Routines

Version 2.0

GEOS Software Development Kit Library

Version 2.0

Routines

Initial Edition, Unrevised and Unexpanded

Geoworks, Inc.
Alameda, CA

Contents

Note: Information about messages is contained within the Objects manual; this manual contains reference entries for each message along with the message's pass and return parameters in C. To find out a message's pass and return parameters for Assembly, see the object class's **.def** file in \PCGEOS\INCLUDE\OBJECTS\*.DEF or \PCGEOS\INCLUDE\*.DEF

Goc Keywords

1

Keywords used in Goc are listed alphabetically on the following pages.

■ @alias

```
@alias(<protoMsg>) <messageDef>;
```

The @alias keyword is used for messages which take conditional parameters in an assembly handler. For example, if the assembly handler takes a word parameter normally and a dword only if a certain flag is set, you will need to have two C messages with the two different parameters. The @alias keyword allows this. Its arguments are shown below:

protoMsg	The name of the existing message that the alias will reference.
messageDef	The new message definition. This is a standard message definition as would follow the @message keyword.

```
@message void MSG_MY_MSG(word par);
@alias(MSG_MY_MSG) void MSG_MY_SECOND_MSG(dword par);
```

See Also: @message

■ @call

```
<ret> = @call [,<flags>] [{<cast_ret>}] \
               <obj>::[{<cast_par>}]<msg>(<param>*);
```

The @call keyword sends the specified message to the specified object and makes the caller wait until the message is processed before continuing. The arguments of @call are shown below:

ret	A variable for receiving the return value of the message as defined by @message. This has the same usage as a typical function return value.
flags	Flags that determine how the message affects the recipient's event queue. The allowable flags are shown below. (The comma is required before each flag.)
cast_ret	A message to cast the message return value to. When Goc determines what type of value should be returned, it uses the return value of the *cast_ret* message if this field is used. The curly braces are required around this field.
obj	The name of the recipient object, or a variable representing the optr of the recipient.
cast_par	A message to cast the message parameters to. When Goc determines what type of values should be passed to the message, should be returned, it uses the parameters of the *cast_par*

message if this field is used. The curly braces are required around this field.

msg	The name of the message to be sent, or an expression representing the message number. If an expression is used, you must cast the message to a certain type with the *cast* parameter.
param	Expressions or constants passed to the message. Parameters passed to messages are specified in the same way as if they were being passed directly to a function or routine in C.

The flags allowed with @call are shown below:

forceQueue
This flag will cause the message to be placed in the recipient's event queue, even if it could have been handled by a direct call.

checkDuplicate
This flag makes the kernel check if a message of the same name is already in the recipient's event queue. For this flag to work, *forceQueue* must also be passed. Note that due to implementation constraints, events will be checked from last to first rather than from first to last.

checkLastOnly
This flag works like *checkDuplicate*, above, except that it checks only the last message in the event queue.

replace This flag modifies *checkDuplicate* and *checkLastOnly* by superseding the duplicate (old) event with the new one. The new event will be put in the duplicate's position in the event queue. If a duplicate is found but the *replace* flag is not passed, the duplicate will be dropped and the new event will be put at the end of the queue.

insertAtFront
This puts the message at the front of the recipient's event queue. Note that this flag will supersede the *replace* flag.

canDiscardIfDesperate
This flag indicates that this event may be discarded if the system is running extremely low on handles and requires more space immediately.

Additionally, @call alows the use of several special expressions in place of the recipient:

self Send the message to the object issuing the @call command. e.g.

```
@call self::MSG_VIS_DRAW(flags, gstate);
```

Routines

process	Send the message to the object's associated Process object. e.g.

```
@call process::MSG_HELLO_RESPOND();
```

application	Send the message to the object's associated GenApplication object.

```
attr = @call application::MSG_GEN_GET_ATTRIBUTES();
```

@genParent	Send the message to the object's generic parent in a generic object tree.
@visParent	Send the message to the object's visible parent in a visible object tree.

If you need to send a message to an object's superclass, use the @callsuper keyword rather than @call.

```
gstate = @call myObj::MSG_META_CUT();
retVal = @call {MSG_MY_MSG} myObj::MSG_OTHER_MSG();
retVal = @call myObj::MSG_MY_MSG(10, param1);
```

See Also: @send, @callsuper, @message, @method, @object

■ @callsuper

```
<ret> = @callsuper [{<cast_ret>}] \
                <obj>::<class>::[{<cast_par>}]<msg>(<param>*) [<flags>]+;
@callsuper;
```

The @callsuper keyword does two things: The most useful is to pass a received message on to the superclass to ensure default behavior is preserved; the second, and less used, acts just like @call but sends the message to the recipient's superclass rather than the recipient's class. This is rarely used but can be used if only default behavior is required of the message. Its arguments are shown below:

ret	Same as @call.
obj	Same as @call.
cast_ret	Same as @call.
class	The name of the object's superclass that should receive the message. It is possible to send the message to the highest superclass.
cast_par	Same as @call.
msg	Same as @call.
param	Same as @call.

flags Same as @call.

```
(void) @callsuper myObj::MySupClass::MSG_MY_MSG();
```

See Also: @call, @send, @message, @method

■ @chunk

@chunk <*type*> <*name*> [= <*init*>];

The @chunk keyword declares a resource chunk containing data of some kind. Data can be of any GEOS or C data type or structure, including a string of characters. The chunk must be declared between the resource delimiters @start and @end. Its arguments are described below:

type	The data type or structure type of the chunk.
name	The name of the chunk—how it will be referenced by other entities.
init	Initializer data, if any, to initialize the chunk to.

If you will need to access the chunk from another executable file, you must declare it in the other file with @extern. Objects are declared with @object.

```
typedef struct {
    int     a;
    int     b;
} MyStruct;

char    MyString[8];

@start MyDataResource, data, notDetachable;
@chunk word         MyWordChunk;
@chunk MyStruct     MyMSChunk = {5, 10};
@chunk MyString     MyStringChunk = “My string”;
@end;
```

See Also: @start, @end, @object, @extern

■ @chunkArray

@chunkArray <*stype*> <*aname*> [= {<*init*>}];

The @chunk keyword declares a Chunk Array, a special kind of chunk. Only uniform-element-size chunk arrays may be declared with this keyword. It has the following arguments:

stype	This is the type of each element in the Chunk Array. It may be any standard C or Goc type, or any derived type.
aname	This is the name of the Chunk Array.

init	You may declare the initializer values for a chunk array. If you do not set any initial values, the Chunk Array will be created with no elements.

```
@chunkArray int     someints;
@chunkArray dword   somedwords = {123456789,
                                  6021023,
                                  31415926};
```

See Also: @chunk, @elementArray

■ @class

```
@class   <cname>, <super> [, master [, variant]];
```

The @class keyword begins a class definition. All instance data and messages for the class are declared between @class and @endc. The arguments of @class are listed below:

cname	Name of the class being declared.
super	Name of the superclass.
master	Use of this term designates this class as a master class.
variant	Use of this term designates this class as a variant class.

```
@class MyTriggerClass, GenTriggerClass;
@endc
@class MyMasterVarClass, MetaClass, master, variant;
@endc
```

See Also: @endc, @classdecl

■ @classdecl

```
@classdecl <cname> [, neverSaved];
```

The @classdecl keyword defines a given class structure in memory. Every new class that will be used by an application must appear in an @classdecl declaration. The arguments for this keyword are shown below:

cname	The name of the class being declared.
neverSaved	Using this term indicates that objects of this class are never saved along with state information.

```
@classdecl MyTriggerClass;
@classdecl MyProcessClass, neverSaved;
```

See Also: @class

■ @composite

```
@instance @composite <iname> = <linkFieldName>;
```

The @composite keyword appears as a subcommand of @instance. It is a type of instance data—it indicates that an object of this class can have several children and that the @composite instance data field will be an optr to the first of its children. The arguments of the @composite keyword are given below:

iname	The name of the instance data field.
linkFieldName	The name of the @link instance data field for this class.

```
@class GenTrigWithKidsClass, GenTriggerClass;
    /* GI_link is the GenClass sibling link field. */
    @instance @composite GTWKI_comp = GI_link;
@endc
```

See Also: @instance, @link

■ @default

```
<fname> = @default [<op> [~]<attr>]*; /* to use default value of
                                         instance data field */
@default <varRoot> = <super>; /* to specify default superclass of
                                a variant class */
@default <fname> = <value>; /* to specify a default value for an instance
               data field defined by a superclass. */
<fname> = @default;
```

The @default keyword can be used in several ways: to specify the default calue of an instance data field, to represent the default value of an object's instance data field, or to specify the default superclass of a variant class. It may also be used when defining a class to specify a default value to use with an instance data field defined by a superclass.

The @default keyword is most commonly used when modifying default instance data values of bitfield-type fields. The defaults are set in the @class definition and may be modified in the @object declaration. The arguments of @default are shown below:

fname	The name of the instance data field. Typically, this is a record.
op	A bitwise operator character. If setting certain bits, use the OR operator (|); if removing certain bits, use the AND operator (&).

Routines

attr	The name of the attribute bit to set or remove. If removing attribute bits, place the logical NOT character (~) in front of the attribute.

```
@object GenPrimaryClass MyPrimary {
    GI_states = @default & ~GENS_MAXIMIZED;
    GI_attrs = @default | GENA_TARGETABLE;
}
```

The @default keyword can also be used to specify the default superclass of a variant class. In this case, it has the following arguments:

varRoot	The name of the variant class, with the word "Class" removed. (e.g. the root of "MyVariantClass" is "MyVariant".)
super	The default superclass for this variant class.

To specify a class' default value for an instance data field when that instance data field is defined by a superclass, @default has the following arguments:

fname	The name of the instance data field.
value	The class' default value for the field.

To represent an object's default value for an instance data field, @default has the following arguments:

fname	The name of the instance data field.

See Also: @object, @instance, @class

■ @define

@define <*mname*>[(<*pdef*>)] <*macro*>

The @define directive defines a Goc macro. You can define C macros with the #define directive; macros that use Goc operators, keywords, or code must be defined with @define. Similarly, macros defined with @define must be later used with the "@" marker preceeding them; otherwise, the Goc processor will scan over the macro and will not evaluate it. The arguments of @define are listed below:

mname	The macro name. This can be used later as @*mname* to invoke the macro.
pdef	The optional parameter definition, as with C macros.
macro	The macro.

```
@define MyChunk(a) @chunk char[] a = “text”;
@define MyText(a,b) @chunk char[] a = “b”;

/* You can later use these macros as follows: */
```

```
@MyChunk(Text1)
@MyText(Text2, newText)
/* This will evaluate to the following: */
@chunk char[] Text1 = “text”;
@chunk char[] Text2 = “newText”;
```

@deflib

`@deflib <libName>`

Most Goc libraries will have a **.goh** header file. This file should begin with a @deflib directive. This will see to it that no library header file is included more than once in a given compilation. The file must end with an @endlib directive. The @deflib directive takes the following argument:

libName This is the name of the header file, with the **.goh** extension stripped off. For example, if the library's header file is **hellolib.goh**, the file would begin with

```
@deflib      hellolib
```

See Also: @endlib

@dispatch

`@dispatch [noFree] [{<cast>}] <nObj>::<nMsg>::<event>;`

The @dispatch keyword sends a previously-encapsulated message to the specified object. This keyword is analogous to @send; use @dispatchcall if the event must be processed immediately. The encapsulated event must have been defined with @record. The arguments of @dispatch are given below:

noFree A flag indicating the event will not be freed after it is handled.

cast A message to cast the parameters to.

nObj An override recipient object for the event. Encapsulated messages can store recipients; this will override the stored value. If no override is desired, specify this as *null*.

nMsg An override message to be sent. Encapsulated messages can store the message number to be sent; this will override the stored value. If no override is desired, specify this as *null*.

event The name of the encapsulated event, defined earlier with @record.

```
@dispatch null::null::myEvent;
@dispatch newObject::null::myEvent;
@dispatch null::MSG_NEW_MSG::myEvent;
```

Routines

See Also: @record, @send, @dispatchcall

■ @dispatchcall

`<ret> = @dispatchcall [noFree] [{<cast>}] <nObj>::<nMsg>::<event>;`

The @dispatchcall keyword sends a previously-encapsulated message to the specified object. This keyword is analogous to @call; use @dispatch if the event can be sent with no return values. The encapsulated event must have been defined with @record. The arguments of @dispatchcall are given below:

ret	A variable to receive the returned value.
noFree	A flag indicating the event will not be freed after it is handled.
cast	A message to cast the parameters and return value to.
nObj	An override recipient object for the event. Encapsulated messages can store recipients; this will override the stored value. If no override is desired, specify this as *null*.
nMsg	An override message to be sent. Encapsulated messages can store the message number to be sent; this will override the stored value. If no override is desired, specify this as *null*.
event	The name of the encapsulated event, defined earlier with @record.

```
retVal = @dispatchcall null::null::myEvent;
retVal = @dispatchcall newObject::null::myEvent;
(void) @dispatchcall null::MSG_NEW_MSG::myEvent;
```

See Also: @record, @send, @dispatchcall

■ @elementArray

`@elementArray <stype> <aname> [= {<init>}];`

The @chunk keyword declares a Element Array, a special kind of Chunk Array. It has the following arguments:

stype	This is the type of each element in the Element Array. It may be any standard C or Goc type, or any derived type.
aname	This is the name of the Element Array.
init	You may declare the initializer values for a chunk array. If you do not set any initial values, the Element Array will be created with no elements.

See Also: @chunk, @chunkArray

■ @end

`@end      <segname>`

The @end keyword denotes the end of a resource block definition that had been started with @start. Its one argument is the name of the resource segment.

```
@start MenuResource;
@end
```

See Also: @start, @header, @object, @chunk

■ @endc

`@endc`

The @endc keyword denotes the end of a class definition begun with @class. It has no arguments.

See Also: @class

■ @endif

`@endif`

The @endif directive denotes the end of a block of conditionally-compiled code. It is used with @if, @ifdef, and @ifndef.

See Also: @if, @ifdef, @ifndef

■ @endlib

`@endlib`

Most Goc libraries will have a **.goh** header file. This file should end with an @endlib directive. This will see to it that no library header file is included more than once in a given compilation. The file must begin with an @deflib directive.

See Also: @deflib

■ @exportMessages

`@exportMessages <expname>, <num>;`

The @exportMessages keyword sets aside a number of message spots so the messages may be declared elsewhere. This allows users of the class to declare messages that are guaranteed to be unique across all subclasses. Exported messages are declared with the @importMessage keyword. The arguments of @exportMessages are shown below:

expname	Name of the range being exported.

■ Routines

num Number of message spots to be exported.

```
@exportMessages MetaUIMessages, 50;
@exportMessages MyExportedMessages, 12;
```

See Also: @importMessage, @reserveMessages, @message

■ @extern

```
@extern  <type> <name>;
@extern  method <cname>, <manme>+
```

The @extern keyword allows code in a given compilation session to access objects, chunks, monikers, and methods defined in another compilation session. The compiler will assume the element exists and will be linked by the Glue linker. If Glue is unable to locate and link the external resource element, it will respond with an error. The arguments of @extern are given below:

type The type of resource element being referenced. This must be one of *object*, *chunk*, *visMoniker*, or *method*.

name The name of the element being referenced.

```
@extern chunk MyChunk;
@extern object MyBlueTrigger;
@extern visMoniker GAL_visMoniker;
```

If @extern is being used to declare a method which is in a different file from the class declaration, it has the following arguments:

cname The name of the class for which the method is defined.

mname+ The name of the message which invokes the method. As with normal method declarations, there must be at least one message which invokes the method.

Some confusion has arisen about when to use **@extern**. The following notes will hopefully prove useful.

Classes

Your class' definition should not be broken up over files. If you wish to keep your class definition in a file separate from your other code, this file should be a **.goh** file.

If your class is *declared* (@classdecl) in a file other than where it is *defined* (@class), then the declaring file should **@include** the defining file.

Normally the declaring file contains all method definitions for the class. If any method definitions are in another file, then both files will need an **@extern** keyword like so:

In file containing class declaration:

```
 @extern method MyClass, MSG_MY_DO_SOMETHING;
```

In file containing method code:

```
@extern method MyClass,
                  MSG_MY_DO_SOMETHING(word myArg)
{ /* Method code here */ }
```

Object Trees

All objects declared in a static tree (e.g. your application's generic tree) should be in the same source file. If they are in different files, then they may be joined into a single tree only by dynamically adding objects from one file as children to objects of the other.

Note that if one file contains a tree of objects, then you may incorporate the whole tree by simply dynamically adding the top object in the file to the main tree. You won't have to add each object individually.

If an object declared in one source file will send a message to an object in another source file, you must include an **@extern** line in the source file containing the sending object:

```
@extern object ReceivingObjectName;
```

The message itself should be sent in the following manner (with variations possible if you will be using **@call**, passing arguments, or what have you):

```
optr ROOptr;
ROOptr = GeodeGetOptrNS(@ReceivingObjectName);
@send ROOptr::MSG_DO_SOMETHING(0, 0);
```

See Also: @chunk, @object, @visMoniker

■ gcnList

gcnList(<*manufID*>,<*lname*>) = <*oname*> [, <*oname*>]*;

The gcnList keyword, which does not have the keyword marker @ preceeding it, puts the listed objects onto the specified notification list. GCN lists are specified by both manufacturer ID and list type. The arguments of the gcnList keyword are given below:

manufID	The manufacturer ID number of the GCN list type. Often this will be MANUFACTURER_ID_GEOWORKS.
lname	The list type, or list name, of the GCN list.
oname	A listing of all the objects that will be included on the GCN list. Separate objects with commas.

```
@object GenApplicationClass HelloApp = {
    GI_comp = HelloPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) =
                                    HelloPrimary;
}
```

See Also: @object

■ @genChildren

`@send @genChildren::<msg>(<params>);`

Any composite object in a generic object tree (therefore a subclass of **GenClass**) can send a message that will be dispatched at once to all of its children. Note that any message sent with @genChildren as the destination must be dispatched with the **@send** keyword and therefore can have no return value and can not pass pointers in its parameters.

■ @genParent

`[@send | @call]@genParent::<msg>(<params>);`

Any composite object in a generic object tree (therefore a subclass of **GenClass**) can use the @genParent address to send a message to its generic parent. This can be used with either @send or @call.

■ @gstring

`@gstring <gsname> = {[<command> [, <command>]+]}`

The @gstring keyword lets you declare a GString in Goc source code.

gsname	The name of the chunk which will contain the GString.
command	This may be any command which could be put in a GString.

■ @header

`@header <type> [= <init>];`

The @header keyword sets the header of an object or data resource segment to a custom structure. The structure must begin with an LMemBlockHeader or ObjLMemBlockHeader. The arguments of @header are given below:

type	The name of the structure set as the new header type.
init	Any initializer data for the fields added to your structure.

```
typedef struct {
    LMemBlockHeader  meta;
    int     a;
    int     b;
} MyLMemBlockHeader;

@start MyDataResource, data, notDetachable;

@header MyLMemBlockHeader = 10, 12;

@end;
```

See Also: @start, @end, @object, @chunk

■ @if

`@if (<cond>)`

The @if directive denotes the beginning of a conditionally-compiled block of code. If the expression detailed in *cond* equates to *true*, then the code between the @if directive and the first corresponding @endif directive will be compiled with the rest of the code.

cond	The expression determining whether the code is to be compiled or not. This expression is based on numerical values, names of macros, and Boolean operators (|| and &&).

```
@if 0
       /* code not compiled */
@endif

@if MyMacro
       /* code compiled if MyMacro is defined */
@endif

@if defined(MyMacro) || MY_CONDITION
       /* code compiled either if MyMacro is defined or
        * if MY_CONDITION is non-zero */
@endif
```

See Also: @ifdef, @ifndef, @endif

Routines

■ @ifdef

`@ifdef <`*`item`*`>`

The @ifdef directive is similar to the @if directive in use, except the condition it evaluates is based solely on whether the *item* is defined or not (if *item* is defined, the following code is compiled).

See Also: @if, @ifndef, @endif

■ @ifndef

`@ifndef <`*`item`*`>`

The @ifndef directive is similar to the @ifdef directive in use, except the condition it evaluates is based solely on whether *item* is not defined (if *item* is not defined, the following code is compiled).

See Also: @if, @ifdef, @endif

■ @importMessage

`@importMessage <`*`expname`*`>, <`*`messageDef`*`>;`

The @importMessage keyword declares a message with a reserved message number set aside earlier by @exportMessages. The arguments of this keyword are given below:

expname	Name of the range exported with @exportMessages.
messageDef	Standard message definition—exactly the same as would follow the @message keyword for message declaration.

```
@importMessage MyExportedMessages, word MSG_MY_MSG(
                                  byte param1, byte param2);
```

See Also: @exportMessages, @reserveMessages, @message

■ @include

`@include <`*`fname`*`>`

The @include directive is used to include Goc files into a code file. It is similar to the #include directive in C. Its only argument is a file name (*fname*) enclosed in either angled brackets or quotation marks. If you use quotation marks, the compiler will look first in the file's own directory; if you use angled brackets, it will look first in the standard include directories.

```
@include <stdapp.goh>
@include <uitsctrl.goh>
@include “Art/mkrGenDoc”
```

■ @instance

@instance <*insType*> <*iname*> = <*default*>;

The @instance keyword declares an instance data field for a class. This keyword will appear between the class delimeters @class and @endc. Its arguments are shown below:

insType	The data type of the instance data field. Must be a valid C data type or data structure. (Note—special types may also be used; see discussion below.)
iname	The name of the instance data field.
default	The default value of the field if it is not declared explicitly in the instance of the class.

The Goc preprocessor allows the use of several special types of instance data fields. To use these special types, insert the proper keyword (type name) in place of the *insType* argument above and do not include a default value for the field (*default*). The possible special types and their meanings are given in the list below (see the individual keyword entries for more detail):

@composite
This field will point to the first child in an object hierarchy. Note that this keyword has a special format. Rather than being allowed a default value, set the *default* argument in the declaration to be the same as the name of the corresponding @link field. This is important; otherwise, your program will not compile properly.

@link This field will point to the next sibling object in an object hierarchy or will point to the parent.

@visMoniker
This field will contain a visual moniker or a pointer to a visual moniker resource chunk.

@kbdAccelerator
This field will contain a keyboard accelerator character.

Note that if you want to declare instance data fields for variable-sized data, you should use the @vardata keyword rather than @instance.

```
@instance int               myInteger = 10;

typedef struc {
    int     a;
    int     b;
} MyStruc;
@instance MyStruc           strucField = {7, 11};
```

Routines

```
@instance @visMoniker      GI_moniker;
@instance @link            VI_link;
@instance @composite       VCI_comp = VI_link;
@instance @kbdAccelerator GI_kbdAcc;
```

See Also: @vardata, @visMoniker, @link, @composite, @kbdAccelerator

■ @kbdAccelerator

`@instance @kbdAccelerator <`*`iname`*`>;`

The @kbdAccelerator keyword follows @instance to create an instance data field that will contain a keyboard accelerator. The *iname* argument is the name of the instance data field.

```
@instance @kbdAccelerator GI_kbdAcc;
```

See Also: @instance

■ @link

`@instance @link <`*`iname`*`>;`

The @link keyword follows @instance to define a link instance data field pointing to the object's next sibling in the object hierarchy. The *iname* argument is the name of the instance data field. Note that the name of the link field must be set as the default value of the corresponding @composite field.

```
@instance @link GI_link;
@instance @composite GI_comp = GI_link;
```

See Also: @instance, @composite

■ @message

`@message <`*`retType`*`> <`*`mname`*`>([@stack] <`*`param`*`>*);`

The @message keyword defines a message and its parameters and return values. This keyword will appear within a class definition (i.e., between @class and @endc). The message defined with @message will automatically be valid for the class for which it is defined as well as for subclasses of that class. The arguments of this keyword are shown below:

retType	The data type of the value returned by this message. This must be a standard C or GEOS data type or pointer.
mname	The name of the message. Typically, this will be the prefix "MSG_" followed by a shortened version of the class name, followed by a short name for the message.

@stack — This keyword may be used if the message might be sent from assembly language code instead of Goc. It indicates that the arguments will be passed on the stack; the handler will pop them off the stack in reverse order from the way they are listed in the declaration.

*param** — The parameters for this message, of which there may be none or several. All the parameters must appear inside the parentheses. Parameters are defined in a similar manner as for functions and routines; each one consists of a data type followed by the name of the parameter of that type.

```
@message void MSG_TRIGGER_PUSHED(int push1);
```

```
@message word MSG_MY_MSG(byte firstParam, word secParam,
                                  long thirdParam);
```

See Also: @method, @reserveMessages, @exportMessages, @importMessage, @record

■ @method

@method [<*hname*>,] <*cname*>, <*mname*>+ [{<code>}];

The @method keyword begins definition of a method (message handler). Its arguments are listed below:

hname — The method name, if any. If no method name is given, one will be created by removing “Class” from the class name and “MSG_” from the message name and concatenating the two.

cname — The name of the class to which the method belongs. Each method belongs to only one class.

mname+ — The name(s) of the message(s) handled by this method. There must be at least one message which invokes this method. There may be more than one, as long as they all have the same parameters.

code — Goc procedural code to handle the message. If there is no code, *hname* is assumed to be the name of an existing routine which should be used as the method.

```
@method   MyClass, MSG_MY_MSG {
    /* method code goes here */
}
```

```
@method   MyClassMethod, MyClass, MSG_MY_MSG {
    /* method code goes here */
}
```

See Also: @message

Routines

■ @noreloc

```
@noreloc <iname>;
```

The @noreloc keyword specifies that an instance data field (defined in the previous program statement) is not relocatable. Normally optr fields are assumed to be relocatable and will be automatically relocated by the system when shutting down and coming back from a shutdown; by means of the @noreloc, this automatic behavior can be turned off for a given field.

■ @object

```
@object  <class> <name> <flags>* = {
    [<fieldName> = <init>;]*
    [<varName> [= <init>];]*
}
```

The @object keyword defines an object in an object resource block. It must appear between @start and @end. Its arguments are defined below:

class	The name of the class of the object.
name	The name of the object.
flags	Flags associated with the object; currently only *ignoreDirty* is supported. When set, this flag indicates that changes to the object should not be saved to a state file. Note, however, that *ignoreDirty* should *never* be set for generic objects.
fieldName	The name of an instance data field defined with @instance. Any number of such fields may be specified.
varName	The name of an instance data field defined with @vardata. Any number of such fields may be specified.
init	Initializer data for a normal instance data field or for the extra data of a variable data field. If a variable data field has no extra data, no initializer should be specified.

Many fields may be specified in the object declaration. Each field reference must be defined in a class in the object's class ancestry. Additionally, not all fields must be set. If a field is not specified within the @object declaration, the field will be set to its default value as defined by the class.

```
@start MyObjectResource;

@object GenTriggerClass MyTrigger ignoreDirty = {
    GI_visMoniker = “MyTrigger’s Moniker”;
}
```

```
@object GenApplicationClass MyApp = {
    GI_comp = MyPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) =
                                           MyPrimary;
}
@object GenPrimaryClass MyPrimary = {
    GI_comp = MyMenu, MyInteraction, MyView;
    GI_visMoniker = “My Primary’s Moniker”;
}
@object MyClass NewObject = {
    NO_instance1 = 1;
    NO_instance6 = ‘C’;
}
@end
```

See Also: @start, @end, @extern, @class, @instance, @vardata

■ @optimize

`@optimize`

This directive may be placed at the top of a **.goh** file. The directive instructs Goc to generate a specially processed **.poh** file which contains all the information of the **.goh** file, but is somewhat faster to compile. This **.poh** file is automatically regenerated if the corresponding **.goh** file has been changed since the last compilation.

■ @protominor

`@protominor <prototypeName>`

When creating a new version of an existing library, use the **@protominor** keyword to declare new messages and variable data fields for a class. Suppose your original class declaration looked like so:

```
@class MyClass, SuperClass;
      @message void MSG_M_DO_THIS(void);
      @vardata void TEMP_M_DONE_FLAG;
@endc
```

Having released this version of your class, you wished to release another version in which this class handled another message. You wanted to specify that this new message would only work with this new version of the library. This would be set up like so:

```
@class MyClass, SuperClass;
      @message void MSG_M_DO_THIS(void);
      @vardata void TEMP_M_DONE_FLAG;

      @protominor MyVersion20
      @message void MSG_M_DO_THAT(void);
@endc
```

To do the equivalent version control with routines, use the **incminor** .gp file directive.

■ @prototype

`@prototype <`*`messageDef`*`>;`

The @prototype keyword allows multiple messages to have the same pass and return parameters. Use @prototype to define the pass and return values, then use @message to declare the messages that have these parameters. The messages defined with @message will have different message numbers and will invoke different methods. The *messageDef* argument is a standard message definition.

```
@prototype word MSG_MY_PROTO(byte param1);

@message(MSG_MY_PROTO) MSG_MY_MSG;
@message(MSG_MY_PROTO) MSG_MY_SECOND_MSG;
```

See Also: @alias, @message

■ @record

`<`*`event`*`> = @record <`*`obj`*`>::<`*`msg`*`>(<`*`param`*`>*);`

The @record keyword encapsulates an event for later use with @dispatch or @dispatchcall. The arguments of @record are as follows:

event	The name of the event. This name will be used with @dispatch and @dispatchcall later.
obj	The name of the object, or an expression representing the object that will receive the message. This may be set to *null* to indicate that the recipient will be determined when the message is sent.
msg	The name of the message, or an expression representing the message that will be sent. This may be set to *null* to indicate that the message will be determined when it it sent.
param	This is a list of parameters that will be sent with the message when it is dispatched.

```
myEvent = @record myObj::MSG_VIS_VUP_CREATE_GSTATE();
```

See Also: @dispatch, @dispatchcall, @call, @send

■ @reloc

```
@reloc   <iname>, [(<count>, <struct>)] <ptrType>;
@reloc   <iname>, <fn>, [(<count>, <struct>)] <ptrType>;
```

The @reloc keyword designates an instance data field that contains data requiring relocation on startup. Note that this does not include instance fields declared with the @composite and @link fields, but it does include any handle or pointer fields you may have. Note that there are two different formats for the use of @reloc. The first represents a normal instance field; the second represents a variable data instance field (see @vardata). This is *not* used with @instance or @vardata but stands alone.

The arguments of @reloc are shown below:

iname	The name of the relocatable instance data field.
count	If the instance variable is an array of relocatable data or structures containing relocatable fields, this is the number of elements in the array.
struct	If the relocatable data is an array of structures, this represents the name of the field within each structure that requires relocation.
ptrType	This is the type of pointer in the relocatable field. This must be one of *optr* (object pointer), *ptr* (far pointer), or *handle.*
fn	This is the name of the field within the extra data of the variable data. If no extra data will be associated with this relocatable field, then put a zero (0) rather than a field name.

```
@reloc MO_myHandle, handle;
@reloc MO_myVarHandle, 0, handle;
@reloc MO_myTable, (10, MyStruct), ptr;
```

See Also: @instance, @vardata

■ _reloc

```
@method [<hname>,] <cname>, _reloc { <code>};
```

The _reloc keyword is used to write relocation handlers for classes, if you need to relocate-unrelocate instance data when it's either read in or saved to state.

The arguments of _reloc are show below:

Routines

code	Code to execute when the object block is loaded in or saved out to state., in which case instance data may need to be relocated or unrelocated by hand.

■ @reserveMessages

```
@reserveMessages <number>;
```

The @reserveMessages keyword reserves the given number of message spots. Messages are numbered sequentially according to the order of their declaration; this keyword allows one or more numbers to be skipped in the numbering process, allowing application upgrades without making earlier versions obsolete. The single argument is the number of message spots to skip.

```
        @reserveMessages 25;
```

See Also: @exportMessages, @importMessage, @message

■ @send

```
@send    [<flags>+] [(<cast_ret>)] <obj>::[{<cast_par>}]<msg>(<param>*);
```

The @send keyword sends a given message to the specified object. The message will be sent and the sender's thread will continue executing without waiting for a response. If return values or synchronization is important, use the @call keyword. The parameters of @send are shown below:

flags	Flags that determine how the message affects the recipient's event queue. The allowable flags are shown below. (The comma is required before each flag.)
cast_ret	A message to cast the message return value to. When Goc determines what type of value should be returned, it uses the return value of the *cast_ret* message if this field is used. The curly braces are required around this field.
obj	The name of the recipient object, or an optr to the object.
cast_par	A message to cast the message parameters to. When Goc determines what type of values should be passed to the message, should be returned, it uses the parameters of the *cast_par* message if this field is used. The curly braces are required around this field.
msg	The name of the message to be sent, or an expression representing the message number. If an expression is used, you must cast the message to a certain type with the *cast* argument .
param	Expressions or constants passed to the message. Parameters passed to messages are specified in the same way as if they were

being passed directly to a function or routine in C. Note that pointers may *not* be passed with @send but handles may; if you must pass a pointer, use @call.

The flags allowed with @send are shown below:

forceQueue
This flag will cause the message to be placed in the recipient's event queue, even if it could have been handled by a direct call.

checkDuplicate
This flag makes the kernel check if a message of the same name is already in the recipient's event queue. For this flag to work, *forceQueue* must also be passed. Note that due to implementation constraints, events will be checked from last to first rather than from first to last.

checkLastOnly
This flag works like *checkDuplicate*, above, except that it checks only the last message in the event queue.

replace This flag modifies *checkDuplicate* and *checkLastOnly* by superseding the duplicate (old) event with the new one. The new event will be put in the duplicate's position in the event queue. If a duplicate is found but the *replace* flag is not passed, the duplicate will be dropped and the new event will be put at the end of the queue.

insertAtFront
This puts the message at the front of the recipient's event queue. Note that this flag will supersede the *replace* flag.

canDiscardIfDesperate
This flag indicates that this event may be discarded if the system is running extremely low on handles and requires more space immediately.

```
@send, forceQueue MyObj::MSG_MY_MSG(10, x);
@send MyObj::MSG_SET_ATTR(attributesParam);
```

See Also: @call, @callsuper, @message, @method

■ @specificUI

```
<fname>  = [@specificUI] <mod>* <key>;
```

The @specificUI keyword is used when setting a keyboard accelerator instance field in an object declaration. It tells the UI to allow the use of the keystrokes specified, even if they are normally reserved for the specific UI. The keyword itself takes no arguments; those shown are for the **GenClass** instance data field *GI_kbdAccelerator*. These are

fname	The name of the instance data field defined with @instance.
mod	Modifier keys; must be one or more of *control*, *ctrl*, *shift*, *alt*.
key	The accelerator character. Must be either a numeric value of a keyboard key or a letter enclosed in single quotation marks.

```
@object MyClass MyObject {
    GI_kbdAccelerator = ctrl shift ‘k’;
}
```

See Also: **GenClass**, @kbdAccelerator, @instance

■ @stack

```
@message <retType> <mname>([@stack] <param>*);
```

This keyword may be used if the message might be sent from assembly language code instead of Goc. It indicates that the arguments will be passed on the stack; the handler will pop them off the stack in reverse order from the way they are listed in the declaration.

See Also: @message

■ @start

```
@start   <segname> [, <flags>];
```

The @start keyword indicates the beginning of a resource block. The end of the block is denoted by the keyword @end. The arguments of @start are listed below:

segname	The name of the resource segment.
flags	Optional flags. The flag *data*, when set, indicates the block will be a data resource rather than an object resource. The flag *notDetachable*, when set, indicates the block should not be saved to a state file.

```
@start MenuResource;
@end

@start MyDataResource, data, notDetachable;
@end
```

See Also: @end, @header, @object, @chunk

■ @uses

`@uses <`*`class`*`>;`

If you know that a variant class will always be resolved to be a subclass of some particular class, you can declare this with the @uses keyword. This will let the variant class define handlers for the “used” superclass. The keyword uses the following argument:

class	A class which will always be a superclass of the defined variant class.

Warnings: You must make sure that the variant class’s inheritance is always resolved such that the used class is one of its ancestor classes.

See Also: @class

■ @vardata

`@vardata <`*`type`*`> <`*`vname`*`>;`

The @vardata keyword creates a vardata data type for a class. Each type created with @vardata can be simply the name of the type, or it can have additional data (a single structure). The arguments of @vardata are given @defaultbelow:

type	This is the data type or structure type of the data field. If no extra data is to be associated with this field, then put the word *void* in place of a type.
vname	This is the name of the variable data instance field.

```
@vardata     dword        MY_FIRST_VAR_DATA;

typedef struc {
    int     a;
    int     b;
} MyStruc;

@vardata     MyStruc      MY_SECOND_VAR_DATA;
@vardata     void         MY_THIRD_VAR_DATA;
```

See Also: @vardataAlias, @instance

■ @vardataAlias

`@vardataAlias (<`*`origName`*`>) <`*`newType`*`> <`*`newName`*`>;`

The @vardataAlias keyword allows you to set up variable data fields with varying amounts of extra data. That is, a single variable data field in the instance chunk could have two different sizes and two different names. The arguments of @vardataAlias are listed below:

origName	The name of the original variable data field defined with @vardata.
newType	The new type or structure associated with this variable data field. If no extra data is to be associated with this alias, then put the word *void* instead of a type.
newName	The new name of the variable data field that uses the new type.

```
/* defined in GenTriggerClass */
@vardata word ATTR_GEN_TRIGGER_ACTION_DATA;
/* A special GenTrigger that uses a different data
 * type is defined in the application: */
@object GenTriggerClass MyTrigger = {
    GTI_actionMsg = MSG_MY_APPS_MESSAGE;
    GTI_destination = process;
    @vardataAlias (ATTR_GEN_TRIGGER_ACTION_DATA)
                    dword ATTR_MY_TRIGGER_SPECIAL_DATA;
```

See Also: @vardata, @instance

■ @visChildren

`@send @visChildren::<msg>(<params>);`

Any composite object in a visible object tree (therefore a subclass of **VisCompClass**) can send a message that will be dispatched at once to all of its children. Note that any message sent with @visChildren as the destination must be dispatched with the **@send** keyword and therefore can have no return value.

■ @visParent

`@send @visParent::<msg>(<params>);`

Any object in a visible tree can use **@visParent** as the destination of an **@call** command. The message will be sent to the object's parent in the visible object tree. The remainder of the command is the same as a normal **@call**.

■ @visMoniker

`@instance @visMoniker <iname>;`

The @visMoniker keyword follows @instance to create an instance data field for a visual moniker. The *iname* argument is the name of the instance data field.

```
@instance @visMoniker GI_visMoniker;
```

See Also: @instance, **GenClass**

Routines

Routines

Parameters File Keywords

2

Keywords used in the **.gp** file of an geode are shown in alphabetical order in this section. These keywords define how the Glue linker will link the geode.

■ appobj

appobj <*name*>

The **appobj** field indicates the name of the application object. All geodes with *appl* set under **type** (see above) must have an **appobj** entry. The *name* argument should be the name of the object of **GenApplicationClass** specified in the application's **.goc** file.

■ class

class <*name*>

The **class** field specifies the name of the object class to be bound to the geode's process thread. This field has significance only if **process** is specified in the geode's **type** field (see below). This should be the same as the **ProcessClass** object designated in the **.goc** file (see the Hello World sample for an example of this connection). Note that this class binding will only be for the geode's first (primary) thread.

■ driver

driver <*name*> [noload]

This field specifies another driver that is used by this geode. The *noload* flag indicates that the used driver does not need to be loaded when the geode is first launched. Most applications and libraries will not use exported routines from drivers, so few geodes will use this field. (Notable exceptions are those geodes that access serial and parallel ports—those geodes will include the serial or parallel driver.)

■ entry

entry <*name*>

This field is used by library geodes. The *name* argument is the name of the library routine to be called by the kernel when the library is loaded or unloaded and when a program using the library is loaded or unloaded.

■ exempt

`exempt <library-name>`

If you wish to exempt a certain library from Glue's platform checking, call it out with the exempt keyword. Glue will not complain if you then use parts of the library not normally available with platforms named in your **platform** statement.

■ export

export <*name*> [as <*name2*>]

This field identifies routines usable by geodes other than the one being compiled; these routines are "exported" for use by other programs. Both forms create entry point symbols for the routines. The first *name* argument must be the actual name of the routine. If the second, optional, *name2* argument is included, then other programs will call that routine using the second name rather than the original. This allows a routine to have a different global name than that used by its creator geode.

This field is also used to export classes defined in a **.goc** or **.goh** file. See Hello World for an example of this usage.

■ incminor

incminor [<name>]

The **incminor** directive is used at the end of a library's **.gp** file before new routines are added (after a release of the library has already been made). After this release, new **export** and **publish** directives will be put after this incminor directive. The **incminor** directive causes two things: First, the geode's minor protocol number gets incremented by one. Second, any geode that uses your library will depend only on the higher minor protocol number if it actually uses one or more of the entry points exported after the **incminor** directive.

Any number of incminor directives may be used in a given **.gp** file. The major and the base minor numbers still come from a **.rev** file, if one exists.

The *name* argument is optional; it may be used in conjunction with the protominor compiler directive. Glue will know that the structures marked with the protominor label should be associated with the revision represented by the incminor directive.

■ library

library <*name*> [noload]

This field specifies another library that is used by this geode. The *noload* flag indicates that the used library does not need to be loaded when the geode is first launched (though symbolic information will be loaded in any case). Note that every geode must have the line

```
library geos
```

included in the **.gp** file. Most will also have the following line:

```
library ui
```

Routines

Any number of used libraries may be specified.

■ load

```
load      <name> ["<class>"] as [<name2>] [<align>] [<combine>]\
             ["<class2>"]
```

The **load** field is used when you want to alter the way a segment is linked for your geode. This is especially useful, for example, when integrating another company's runtime routines into your application or library; their segments may correspond to specifications other than yours.

Every segment read in has a given name, class, alignment, and combination type. These are described below (the **load** parameters appear after):

name This is the actual name of the segment being loaded in. Segments with the same name are treated as one continuous segment.

class Segments with the same class name are always loaded together into memory regardless of their order in the geode's source code. Class names in the load directive must always be enclosed in quotation marks.

align This specifies the alignment type of the segment—on what type of address the segment can start. Possible alignment settings are byte, word, double word, paragraph, and page.

combine Segments with the same name may appear in different code modules. The *combine* parameter specifies how these segments are to be combined when loaded. The combine type may be one of the following (see your assembly reference manual for more information): COMMON, PRIVATE, PUBLIC, STACK, or RESOURCE.

The parameters for load are listed below. Only the first is necessary, to inform Glue which segment is to undergo the alterations. For an example of using the load statement, see below.

name This represents the actual original name of the segment. It is a necessary parameter so Glue knows which segment's linkage is to be altered.

class This is the original class name of the segment. It must be enclosed in quotation marks if given. If you do not need to change the class, this parameter is unnecessary.

name2 This is the new name of the segment, if any.

align This specifies the new align type of the segment, if any.

combine This specifies the new combine type of the segment, if any.

class2 This specifies a new class name for the segment, if any is required. If you do not need to change the class, this parameter is unnecessary. The new class must be in quotation marks.

Examples:

```
load _NAME_ “CODE” as CODE word public
load _NAME_ “CODE” as DATASEG para common “DATA”
```

■ longname

`longname` *“<string>”*

The **longname** field designates a 32-character name for the geode. This name will be displayed with the geode’s icon by GeoManager; all geodes should be given a long name.

■ name

`name` *<pname>.<ext>*

The **name** field in the parameters file gives the geode a permanent name which will be used by both the Glue linker and the Swat debugger. Every geode must have a permanent name. Note that the *pname* argument must be no more than eight characters, and the *ext* argument must be no more than four. Additionally, the *ext* argument may not be “appl,” as that is reserved.

When Glue is linking an error-checking geode, it drops the fourth character of *ext* and adds “ec” to the end of *pname*.

■ nosort

`nosort`

This keyword should appear before the list of resources. Normally glue will sort the geode’s resources to optimize their arrangement. This keyword turns off that sorting. If you will generate .GYM (generic symbol) files for your geode, you should use the nosort option, as it will be important that all versions of your geode order their resources in the same way. If you won’t generate .GYM files, you probably don’t want to use this option.

■ platform

`platform` *<name>*

The platform directive specifies that the Geode is compatible with the named system. This gives a sign of how backwards-compatible the application is. If multiple platforms are specified, Glue will make sure that the major protocol numbers for each of the libraries it finds within the platforms match. Having done that, it will use the smallest minor protocol number it can find for each library to ensure compatibility across all platforms.

If a reference is ever made to an entry point in a library that would cause the executable to depend upon a later version of the library than specified in the platform file, glue will complain. For example, if the specified platoform used GrObj version 534.1 and glue found a reference to an entry point that didn't exist until GrObj 534.3 (ie., an entry point exported following 3 'incminor's in grobj.gp), glue will spit out an error message like:

```
error: file “somegeode.gp”, line 59: Usage of
NewGrObjRoutine requires grobj minor protocol 3, but
platform files only allow minor protocol 1
```

If the new routine happens to be a “published” routine, glue will copy it into the geode in an effort to avoid the error.

■ publish

```
publish <name>
```

Normally, If a geode is required to run (via platform specifications) with a version of a library that doesn't contain one of the entry points required by the geode, glue will notify the user of the inconsistency, and the link will fail. However, if that entry point happens to be a published routine, glue will actually copy the routine into the geode and switch the call over to the newly copied routine to remove the dependency on the library routine. Glue does this by copying any routines marked “publish” in a library's .gp file into the .ldf file, then copying them out into whatever other geodes needs when those geodes are linked. Routines are marked “publish” by replacing the word “export” with the word “publish” in the .gp file, like so:

```
publish PublishedRoutinei
```

The published routines appear in .ldf files in individual segments named after the routine (e.g. _PUBLISHED_PublishedRoutine), each containing a routine, also named after the published routine (e.g., _PUBLISHED__PUBLISHED_PublishedRoutine) You'll know one of these routines has been linked into your geode by examining the resource summary output by glue:

```
Resource                          Size # Relocs
-----------------------------------------------
CoreBlock                         0     0
dgroup                            240   8
_PUBLISHED_GROBJCALCCORNERS       53    1
_PUBLISHED_GROBJBODYPROCESSALLGR  94    2
```

```
TEST2_E                     478    27
INTERFACE                   652    1
CHANGETEXTDIALOG            232    1
APPRESOURCE                 416    1
```

■ resource

```
resource <name> (read-only|preload|discardable|fixed|conforming|shared|\
                code|data|lmem|discard-only|swap-only|ui-object|object|\
                no-swap|no-discard)+
```

The **resource** field indicates to Glue that the geode uses the named resource. Not all resources used by a geode must be declared here, however. (Resources are described in more detail in "GEOS Programming," Chapter 5.) Resources must be designated with the proper attributes, all of which are listed below:

(none)	If no attribute is specified, the resource named becomes a private data resource for the geode.
read-only	The resource block may not be modified by the program.
preload	The resource block should be loaded when the geode is first launched.
discardable	The resource block may be discarded from memory if necessary.
fixed	The resource block should reside in fixed memory.
conforming	The resource block, if containing code, may be called from a lower privilege level. If containing data, it may be accessed from a lower privilege level. (This applies only in protected mode and is not currently implemented.)
shared	The resource block may be used by other geodes. (Note: It is an error to specify *code* and *shared* without *read-only*.)
code	The resource block contains executable code.
data	The resource block contains data only. If a data resource is designated *read-only* and not fixed, it is assumed to be discardable.
lmem	The resource block consists of a local memory heap. This implies the attribute *data* (above), though not the condition pertaining to being discardable.
discard-only	The resource block should not be swapped but may be discarded. This is useful for initialization code.
swap-only	The resource block should not be discarded but may be swapped.

Routines

ui-object	The resource block contains objects to be run by the UI. This implies *lmem*, *shared*, and *no-discard*. All blocks for a geode designated *ui-object* will be run in a UI thread created specifically for the geode's UI objects.
object	The resource block contains objects that are to be run by the application's process thread rather than by the UI. This implies *lmem* and *no-discard*.
no-swap	The resource block will not be swapable.
no-discard	The resource block will not be discardable.

Because most resources are code resources, standard code does not have to be declared in the parameters file. Code resources default to *code*, *read-only*, and *shared*. However, if the resource is named in the **.gp** file, the default is overridden in favor of the settings presented. This fact is useful primarily when programming in assembly—in C, code resources are not declared explicitly.

The Hello World sample application uses only standard code resources (undeclared) and UI resources (designated *ui-object*). Some other examples are listed below:

- Shared data
  ```
  resource <name> data shared
  ```
- Initialization code
  ```
  resource <nm> code shared read-only preload no-swap
  ```
- Common code used by several geodes (this is the default)
  ```
  resource <name> code shared read-only
  ```
- Self-modifying code (strongly discouraged)
  ```
  resource <name> code
  ```

■ stack

```
stack     <number>
```

The **stack** field designates the size of the application's stack in bytes. The default stack size is 2000 bytes. This field is not necessary for geodes unless they require a different size stack (the Hello World sample uses a slightly smaller stack size for example only). The **stack** field is valid only for geodes with a process aspect.

■ tokenchars

`tokenchars` *“<string>”*

This is one of two fields that identifies a unique token in GeoManager’s token database file (see **tokenid**, below). The **tokenchars** field must be a string of four characters that identifies the geode’s token. Note that these characters also appear in the geode file’s extended attributes.

■ tokenid

`tokenid` *<number>*

This is the other of two fields that identifies a unique token in GeoManager’s token database file (see **tokenchars**, above). It must be a number corresponding to the programmer’s manufacturer ID number. Note that this number also appears in the geode file’s extended attributes.

■ type

```
type    (process|driver|appl|library)+ [single] [system] [uses-coproc]\
          [needs-coproc] [has-gcm] [c-api]
```

The **type** field in the parameters file designates certain characteristics of the geode being compiled. These attributes correspond to the **GeodeAttrs** type and determine how the Glue linker will put the geode together. The attributes are as follows:

process This attribute indicates the geode has its own thread. Applications should always have process specified in the type field.

driver This attribute indicates the geode has a driver aspect.

appl This attribute indicates the geode has an application aspect.

library This attribute indicates the geode has a library aspect.

single This geode may only have one copy running at a time. Some applications may allow multiple copies to be running at once; they should not specify single as a type attribute.

system This attribute is set for drivers that must be exited specially and must always be exited. For example, a swap driver has special exit conditions that must always be met and is therefore a system driver.

uses-coproc
This attribute is set if the geode will make use of a math coprocessor if one is available. Note that if the geode with this

attribute set is a library, all applications that use the library will inherit the property. This attribute is used to indicate that the coprocessor's state must be saved during a context switch.

needs-coproc	This attribute indicates that the geode must have a math coprocessor to run. (This implies *uses-coproc*, above).
has-gcm	This attribute indicates that the application being compiled has a GCM (appliance) version. This information is used by Welcome to locate all GCM applications.
c-api	This attribute indicates the library entry points are written in C so the kernel must call them with C calling conventions.

■ usernotes

`usernotes` *"<string>"*

This field specifies text to be put in the **.geo** file's usernotes field. The text must be within quotation marks and can be up to 100 characters long. It must contain no line breaks. This can be useful for containing copyright notices in the executable files. The user can read the text in the usernotes by using GeoManager's File/Get Info command.

Routines

Routines

3

All routines in the kernel and the supplied libraries are listed alphabetically in the following pages. In many cases, data structures are listed with certain routines. Global data structures and data types are listed in a later section.

■ ArrayQuickSort()

```
void      ArrayQuickSort(
          void   *array,      /* Pointer to start of array */
          word   count,       /* Number of elements in array */
          word   elementSize, /* Size of each element (in bytes) */
          word   valueForCallback, /* Passed to callback routine */
          QuickSortParameters *parameters);
```

This routine sorts an array of uniform-sized elements. It uses a modified QuickSort algorithm, using an insertion sort for subarrays below a certain size; this gives performance of $O(n \log n)$. The routine calls a callback routine to actually compare elements.

ArrayQuickSort() is passed five arguments: A pointer to the first element of the array, the number of elements in the array, the size of each element in bytes, a word of data (which is passed to all callback routines), and a pointer to a **QuickSortParameters** structure.

Before **ArrayQuickSort()** examines or changes any element, it calls a locking routine specified by the **QuickSortParameters** structure. This routine locks the element, if necessary, and takes any necessary prepatory steps. Similarly, after **ArrayQuickSort()** is finished with a routine, it calls an unlocking routine in the **QuickSortParameters**. Each of these routines is passed a pointer to the element and the word of callback data which was passed to **ArrayQuickSort()**.

The sort routine does not compare elements. Rather, it calls a comparison callback routine specified by the **QuickSortParameters**. This callback routine should be declared _pascal. Whenever **ArrayQuickSort()** needs to compare two elements, it calls the callback routine, passing the addresses of the elements and the *valueForCallback* word which was passed to **ChunkArraySort()**. The callback routine's return value determines which element will come first in the sorted array:

- ◆ If element *el1* ought to come before *el2* in the sorted array, the callback routine should return a negative integer.
- ◆ If element *el1* ought to come after *el2* in the sorted array, the callback routine should return a positive integer.
- ◆ If it doesn't matter whether *el1* comes before or after *el2* in the array, the callback routine should return zero.

Include: chunkarr.h

Tips and Tricks: You may need to sort an array based on different criteria at different times. The simplest way to do this is to write one general-purpose callback routine and have the *valueForCallback* word determine how the sort is done. For example, the same callback routine could sort the array in ascending or descending order, depending on the *valueForCallback*.

Be Sure To: Lock the array on the global heap (unless it is in fixed memory).

Warnings: Do not have the callback routine do anything which might invalidate pointers to the array. For example, if the array is in a chunk, do not resize the chunks or allocate other chunks in the same LMem heap.

See Also: QuickSortParameters, ChunkArraySort()

■ BlockFromTransferBlockID

```
VMBlockHandle BlockFromTransferBlockID(id);
          TransferBlockID id;
```

This macro extracts the VMBlockHandle from a **TransferBlockID**.

■ BlockIDFromFileAndBlock

```
TransferBlockID BlockIDFromFileAndBlock(file, block);
          VMFileHandle file;
          VMBlockHandle block;
```

This macro creates the dword type **TransferBlockID** from a VMFileHandle and a VMBlockHandle.

■ bsearch()

```
extern void *_pascal bsearch(
          const void          *key,
          const void          *array,
          word                count,
          word                elementSize,
          PCB(int, compare, (const void *, const void *)));
```

This is a standard binary search routine. The callback routine must be declared _pascal.

■ calloc()

```
void *    calloc(
          word                n,         /* number of structures to allocate */
          size_t              size);     /* size of each structure in bytes */
```

The **malloc()** family of routines is provided for Standard C compatibility. If a geode needs a small amount of fixed memory, it can call one of the routines. The kernel will allocate a fixed block to satisfy the geode's **malloc()** requests; it will allocate memory from this block. When the block is filled, it will allocate another fixed malloc-block. When all the memory in the block is freed, the memory manager will automatically free the block.

When a geode calls **calloc()**, it will be allocated a contiguous section of memory large enough for the specified number of structures of the specified size. The memory will be allocated out of its malloc-block, and the address of the start of the memory will be returned. The memory will be zero-initialized. If the request cannot be satisfied, **calloc()** will return a null pointer. The memory is guaranteed not to be moved until it is freed (with **free()**) or resized (with **realloc()**). When GEOS shuts down, all fixed blocks are freed, and any memory allocated with **calloc()** is lost.

Tips and Tricks: You can allocate memory in another geode's malloc-block by calling **GeoMalloc()**. However, that block will be freed when the other geode exits.

Be Sure To: Request a size small enough to fit in a malloc-block; that is, the size of the structure times the number of structures requested must be somewhat smaller than 64K.

Warnings: All memory allocated with **calloc()** is freed when GEOS shuts down.

See Also: malloc(), free(), GeoMalloc(), realloc()

■ CCB()

```
#define CCB(return_type, pointer_name, args) \
          return_type _cdecl (*pointer_name) args
```

This macro is useful for declaring pointers to functions that use the C calling conventions. For example, to declare a pointer to a function which is passed two strings and returns an integer, one could write

```
CCB(int, func_ptr, (const char *, const char *));
```

which would be expanded to

```
int _cdecl (*func_ptr) (const char *, const char *);
```

See Also: PCB()

■ CellDeref()

```
void *    CellDeref(
          optr   CellRef);
```

This routine translates an optr to a cell into the cell's address. The routine is simply a synonym for **LMemDeref()**.

■ CellDirty()

Section 19.4.2.2 of the Concepts book

```
void     CellDirty(
         void *            ptr);  /* pointer to anywhere in locked cell */
```

This routine marks a cell as “dirty”; i.e., the cell will have to be copied from memory back to the disk.

Include: cell.h

Tips and Tricks: All the cells in an item block are marked dirty at once; thus, you can call this routine just once for several cells in the same item block. Only the segment portion of the pointer is significant; thus, you can pass a pointer to anywhere in the cell. This is useful if you have incremented the pointer to the cell.

■ CellGetDBItem()

Section 19.4.2.2 of the Concepts book

```
DBGroupAndItem CellGetDBItem(
         CellFunctionParameters *cfp,
         word              row,      /* Get handles of cell in this row */
         byte              column);  /*...and this column */
```

All cells are stored as ungrouped DB items. If you wish to manipulate the cells with standard DB routines, you will need to know their handles. The routine is passed the address of the **CellFunctionParameters** and the row and column indices of the desired cell. It returns the **DBGroupAndItem** value for the specified cell. If there is no cell at the specified coordinates, it returns a null **DBGroupAndItem**. The routine does not lock the cell or change it in any way.

Include: cell.h

See Also: DBGroupAndItem

■ CellGetExtent()

Section 19.4.2.2 of the Concepts book

```
void     CellGetExtent(
         CellFunctionParameters *cfp,
         RangeEnumParams *   rep); /* write boundaries in REP_bounds field */
```

This routine returns the boundaries of the utilized portion of the cell file. The routine is passed the address of the cell file’s **CellFunctionParameters** structure.) It writes the results into the *REP_bounds* field of the passed **RangeEnumParams** structure. The index of the first row to contain cells is written into *REP_bounds.R_top*; the index of the last occupied row is written to *REP_bounds.R_bottom*; the index of the first occupied column is written to *REP_bounds.R_left*; and the index of the last occupied row is written to

REP_bounds.R_right. If the cell file contains no cells, all four fields will be set to –1 .

Include: cell.h

■ CellLock()

Section 19.4.2.2 of the Concepts book

```
void *   CellLock(
         CellFunctionParameters* cfp,
         word                    row,      /* Lock cell in this row... */
         word                    column);  /* ... and this column */
```

This routine is passed the address of the **CellFunctionParameters** of a cell file, and the row and column indices of a cell. It locks the cell and returns a pointer to it.

Include: cell.h

See Also: CellLockGetRef()

■ CellLockGetRef()

Section 19.4.2.2 of the Concepts book

```
void *   CellLockGetRef(
         CellFunctionParameters* cfp,
         word                    row,      /* Lock cell in this row... */
         word                    column,   /* ... and this column */
         optr *                  ref);     /* Write handles here */
```

This routine is passed the address of the **CellFunctionParameters** of a cell file, and the row and column indices of a cell. It locks the cell and returns a pointer to it. It also writes the locked cell's item-block and chunk handles to the optr. If the cell moves (e.g. because another cell is allocated), you can translate the optr structure into a pointer by passing it to **CellDeref()**.

Include: cell.h

Warnings: The optr becomes invalid when the cell is unlocked.

See Also: CellGetDBItem(). CellLock()

■ CellReplace()

```
void      CellReplace{
          CellFunctionParameters *cfp,
          word               row,   /* Insert/replace cell at this row... */
          word               column,   /* ... and this column */
          const void *       cellData, /* Copy this data into the new cell */
          word               size);    /* Size of new cell (in bytes) */
```

This routine is used for creating, deleting, and replacing cells in a cell file. To create or replace a cell, set *cellData* to point to the data to copy into the new cell, and set *size* to the length of the cell in bytes, and *row* and *column* the cell's coordinates. (As usual, *cfp* is a pointer to the cell file's **CellFunctionParameters** structure.) Any pre-existing cell at the specified coordinates will automatically be freed, and a new cell will be created.

To delete a cell, pass a *size* of zero. If there is a cell at the specified coordinates, it will be freed. (The *cellData* argument is ignored.)

Include: cell.h

Warnings: If a cell is allocated or replaced, pointers to all ungrouped items (including cells) in that VM file may be invalidated. The **CellFunctionParameters** structure must not move during the call; for this reason, it may not be in an ungrouped DB item. Never replace or free a locked cell; if you do, the cell's item block will not have its lock count decremented, which will prevent the block from being unlocked.

■ CellUnlock()

```
void      CellUnlock(
          void * ptr); /* pointer to anywhere in locked cell */
```

This routine unlocks the cell pointed to by *ptr*. Note that a cell may be locked several times. When all locks on all cells in an item-block have been released, the block can be swapped back to the disk.

Include: cell.h

Tips and Tricks: The DB manager does not keep track of locks on individual items; instead, it keeps a count of the total number of locks on all the items in an item-block. For this reason, only the segment address of the cell is significant; thus, you can pass a pointer to somewhere within (or immediately after) a cell to unlock it. This is useful if you have incremented the pointer to the cell.

Be Sure To: If you change the cell, dirty it (with **CellDirty()**) *before* you unlock it.

Routines

■ CFatalError()

```
void      CFatalError(
          word   code)
```

This routine generates a fatal error. It stores an error code passed for use by the debugger.

■ ChunkArrayAppend()

```
void *    ChunkArrayAppend(
          optr   array,                /* optr to chunk array */
          word   elementSize)          /* Size of new element (ignored if
                                        * elements are uniform-sized) */
```

This routine adds a new element to the end of a chunk array. It automatically expands the chunk to make room for the element and updates the **ChunkArrayHeader**. It returns a pointer to the new element.

One of the arguments is the size of the new element. This argument is significant if the array contains variable-sized elements. If the elements are uniform-sized, this argument is ignored. The array is specified with an optr.

Include: chunkarr.h

Be Sure To: Lock the block on the global heap (if it is not fixed).

Warnings: This routine resizes the chunk, which means it can cause heap compaction or resizing. Therefore, all existing pointers to within the LMem heap are invalidated.

See Also: ChunkArrayInsertAt(), ChunkArrayDelete(), ChunkArrayResize()

■ ChunkArrayAppendHandles()

```
void *    ChunkArrayAppendHandles(
          MemHandle          mh,          /* Handle of LMem heap's block */
          ChunkHandle        ch,          /* Handle of chunk array */
          word               size)        /* Size of new element (ignored if
                                           * elements are uniform-sized) */
```

This routine is exactly like **ChunkArrayAppend()**, except that the chunk array is specified by its global and local handles instead of by an optr.

Include: chunkarr.h

Be Sure To: Lock the block on the global heap (if it is not fixed).

Warnings: This routine resizes the chunk, which means it can cause heap compaction or resizing. Therefore, all existing pointers to within the LMem heap are invalidated.

See Also: ChunkArrayInsertAt(), ChunkArrayDelete(), ChunkArrayResize()

■ ChunkArrayCreate()

```
ChunkHandle ChunkArrayCreate(
          MemHandle mh,       /* Handle of LMem heap's block */
          word   elementSize, /* Size of each element (or zero if elements are
                               * variable-sized) */
          word   headerSize,  /* Amount of chunk to use for header (or zero for
                               * default size) */
          ObjChunkFlags ocf);
```

This routine sets up a chunk array in the specified LMem heap. The heap must have already been initialized normally. The routine allocates a chunk and sets up a chunk array in it. It returns the chunk's handle. If it cannot create the chunk array, it returns a null handle.

If the chunk array will have uniform-size elements, you must specify the element size when you create the chunk array. You will not be able to change this. If the array will have variable-sized elements, pass an element size of zero.

The chunk array always begins with a **ChunkArrayHeader**. You can specify the total header size; this is useful if you want to begin the chunk array with a special header containing some extra data. However, the header must be large enough to accommodate a **ChunkArrayHeader**, which will begin the chunk. If you define a header structure, make sure that its first element is a **ChunkArrayHeader**. Only the chunk array code should access the actual **ChunkArrayHeader**. If you pass a *headerSize* of zero, the default header size will be used (namely, **sizeof(ChunkArrayHeader)**). If you pass a non-zero *headerSize*, any space between the **ChunkArrayHeader** and the heap will be zero-initialized.

To free a chunk array, call **LMemFree()** as you would for any chunk.

Include: chunkarr.h

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

Warnings: Results are unpredictable if you pass a non-zero *headerSize* argument which is smaller than **sizeof(ChunkArrayHeader)**. Since the routine allocates a chunk, it can cause heap compaction or resizing; all pointers to within the block are invalidated.

■ ChunkArrayCreateAt()

```
ChunkHandle ChunkArrayCreateAt(
          optr   array,       /* Create chunk array in this chunk */
          word   elementSize, /* Size of each element (or zero if elements are
                               * variable-sized) */
          word   headerSize,  /* Amount of chunk to use for header (or zero for
                               * default size) */
          ObjChunkFlags ocf);
```

This routine is exactly like **ChunkArrayCreate()**, except that you specify the chunk which will be made into a chunk array. The chunk is specified with an optr. Note that any data already existing in the chunk will be overwritten.

Warnings: The chunk may be resized, which invalidates all pointers to within the LMem heap.

Include: chunkarr.h

■ ChunkArrayCreateAtHandles()

```
ChunkHandle ChunkArrayCreateAtHandles(
          MemHandle          mh,
          ChunkHandle        ch,
          word               elementSize,
          word               headerSize,
          ObjChunkFlags      ocf);
```

This routine is exactly like **ChunkArrayCreate()**, except that the chunk is specified with its global and chunk handles instead of with an optr.

Tips and Tricks: If you pass a null chunk handle, a new chunk will be allocated.

Warnings: The chunk may be resized, which would invalidate all pointers to within the LMem heap.

Include: chunkarr.h

■ ChunkArrayDelete()

```
void      ChunkArrayDelete(
          optr   array,                  /* optr to chunk array */
          void * element);               /* Address of element to delete */
```

This routine deletes an element from a chunk array. It is passed the address of that element, as well as the optr of the array.

Since the chunk is being decreased in size, the routine is guaranteed not to cause heap compaction or resizing.

Include: chunkarr.h

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

Tips and Tricks: Only the chunk handle portion of the optr is significant; the memory block is determined from the pointer to the element.

Warnings: The addresses of all elements after the deleted one will change. No other addresses in the block will be affected. If the address passed is not the address of an element in the array, results are undefined.

See Also: ChunkArrayAppend(), ChunkArrayInsertAt(), ChunkArrayResize(), ChunkArrayZero()

■ ChunkArrayDeleteHandle()

```
void      ChunkArrayDeleteHandle(
          ChunkHandle         ch,          /* Handle of chunk array */
          void *              el);         /* Address of element to delete */
```

This routine is exactly like **ChunkArrayDelete()**, except that the chunk array is specified with its chunk handle instead of with an optr. The global memory handle is not needed, as the memory block is implicit in the pointer to the element.

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

Include: chunkarr.h

■ ChunkArrayDeleteRange()

```
void      ChunkArrayDeleteRange(
          optr   array,           /* optr to chunk array */
          word   firstElement,    /* index of first element to delete */
          word   count);          /* # of elements to delete (-1 to delete to
                                   * end of array) */
```

This routine deletes several consecutive elements from a chunk array. The routine is passed the optr of the chunk array, the index of the first element to delete, and the number of elements to delete. The routine is guaranteed not to cause heap compaction or resizing; thus, pointers to other elements in the array will remain valid.

■ ChunkArrayElementResize()

```
void      ChunkArrayElementResize(
          optr   array,              /* optr to chunk array */
          word   element,            /* Index of element to resize */
          word   newSize);           /* New size of element, in bytes */
```

This routine resizes an element in a chunk array. The chunk array must have variable-sized elements. The routine is passed an optr to the chunk array (which must be locked on the global heap), as well as the index of the element to resize and the new size (in bytes). It does not return anything.

If the new size is larger than the old, null bytes will be added to the end of the element. If the new size is smaller than the old, bytes will be removed from the end to truncate the element to the new size.

Warnings: If the element is resized larger, the chunk array may move within the LMem heap, and the heap itself may move on the global heap; thus, all pointers to within the LMem heap will be invalidated.

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

Include: chunkarr.h

■ ChunkArrayElementResizeHandles()

```
void      ChunkArrayElementResizeHandles(
          Memhandle          mh,          /* Global handle of LMem heap */
          ChunkHandle        ch,          /* Chunk handle of chunk array */
          word               el,          /* Index of element to resize */
          word               ns);         /* New size of element, in bytes */
```

This routine is exactly like **ChunkArrayElementResize()** except that the chunk array is specified with its global and chunk handles, instead of with its optr.

Warnings: If the element is resized to larger than the old, the chunk array may move within the LMem heap, and the heap itself may move on the global heap; thus, all pointers to within the LMem heap will be invalidated.

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

Include: chunkarr.h

■ ChunkArrayElementToPtr()

```
void *   ChunkArrayElementToPtr(
         optr                array,           /* optr to chunk array */
         word                elementNumber,   /* Element to get address of */
         void *              elementSize);    /* Write element’s size here */
```

This routine translates the index of an element into the element’s address. The routine is passed an optr to the chunk array, the index of the element in question, and a pointer to a word-sized variable. It returns a pointer to the element. If the elements in the array are of variable size, it writes the size of the element to the variable pointed to by the *elementSize* pointer. If the elements are of uniform size, it does not do this.

If the array index is out of bounds, the routine returns a pointer to the last element in the array. The routine will also do this if you pass the constant CA_LAST_ELEMENT.

Include: chunkarr.h

Tips and Tricks: If you are not interested in the element’s size, pass a null pointer as the third argument.

Be Sure To: Lock the LMem heap’s block on the global heap (unless it is fixed).

Warnings: The error-checking version fatal-errors if passed the index CA_NULL_ELEMENT (i.e. 0xffff, or -1).

■ ChunkArrayElementToPtrHandles()

```
void *   ChunkArrayElementToPtrHandles(
         Memhandle           mh,              /* Handle of LMem heap’s block */
         ChunkHandle         chunk,           /* Handle of chunk array */
         word                elementNumber,   /* Element to get address of */
         void *              elementSize);    /* Write element’s size here */
```

This routine is just like **ChunkArrayElementToPtr()**, except that the chunk array is specified with its global and chunk handles, instead of with an optr.

Include: chunkarr.h

Tips and Tricks: If you are not interested in the element’s size, pass a null pointer as the fourth argument.

Be Sure To: Lock the LMem heap’s block on the global heap (unless it is fixed).

See Also: ChunkArrayPtrToElement()

Warnings: The error-checking version fatal-errors if passed the index CA_NULL_ELEMENT (i.e. 0xffff, or -1).

■ ChunkArrayEnum()

```
Boolean  ChunkArrayEnum(
         optr              array,    /* optr to chunk array */
         void *            enumData, /* This is passed to callback routine */
         Boolean _pascal (*callback) (void *element, void *enumData));
                /* callback called for each element; returns TRUE to stop */
```

This routine lets you apply a procedure to every element in a chunk array. The routine is passed an optr to the callback routine, a pointer (which is passed to the callback routine), and a pointer to a Boolean callback routine. The callback routine, in turn, is called once for each element in the array, and is passed two arguments: a pointer to an element and the pointer which was passed to **ChunkArrayEnum()**. If the callback routine ever returns *true* for an element, **ChunkArrayEnum** will stop with that element and return *true*. If it enumerates every element without being aborted, it returns *false*.

The callback routine can call such routines as **ChunkArrayAppend(), ChunkArrayInsertAt(),** and **ChunkArrayDelete()**. **ChunkArrayEnum()** will see to it that every element is enumerated exactly once. The callback routine can even make a nested call to **ChunkArrayEnum()**; the nested call will be completed for every element before the outer call goes to the next element. The callback routine should be declared _pascal.

Include: chunkarr.h

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

■ ChunkArrayEnumHandles()

```
Boolean  ChunkArrayEnumHandles(
         MemHandle         mh,       /* Handle of LMem heap's block */
         ChunkHandle       ch,       /* Handle of chunk array */
         void *            enumData, /* Buffer used by callback routine */
         Boolean _pascal (*callback) (void *element, void *enumData));
                /* callback called for each element; returns TRUE to stop */
```

This routine is exactly like **ChunkArrayEnum()**, except that the chunk array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ChunkArrayEnumRange()

```
Boolean  ChunkArrayEnumRange(
         optr   array,        /* optr to chunk array */
         word   startElement, /* Start enumeration with this element */
         word   count,        /* Process this many elements */
         void * enumData,     /* This is passed to the callback routine */
         Boolean _pascal (*callback)/* Return TRUE to halt enumeration */
                (void *element, void *enumData));
```

This routine is exactly like **ChunkArrayEnum()** (described above), except that it acts on a limited portion of the array. It is passed two additional arguments: the index of the starting element, and the number of elements to process. It will begin the enumeration with the element specified (remember, the first element in a chunk array has an index of zero). If the count passed would take the enumeration past the end of the array, **ChunkArrayEnumRange()** will automatically stop with the last element. You can instruct **ChunkArrayEnumRange()** to process all elements by passing a *count* of CA_LAST_ELEMENT.

Include: chunkarr.h

Warnings: The start element must be within the bounds of the array.

See Also: ChunkArrayEnum()

■ ChunkArrayEnumRangeHandles()

```
Boolean  ChunkArrayEnumRangeHandles(
         MemHandle mh,        /* Handle of LMem heap's block */
         ChunkHandle ch,      /* Handle of chunk array */
         word   startElement, /* Start enumeration with this element */
         word   count,        /* Process this many elements */
         void * enumData,     /* This is passed to the callback routine */
         Boolean _pascal (*callback)/* Return TRUE to halt enumeration */
                (void *element, void *enumData));
```

This routine is exactly like **ChunkArrayEnumRange()**, except that the chunk array is specified by its global and chunk handles (instead of with an optr).

■ ChunkArrayGetCount()

```
word     ChunkArrayGetCount(
         optr   array);                 /* optr of chunk array */
```

This routine returns the number of elements in the specified chunk array.

Include: chunkarr.h

Tips and Tricks: It is usually faster to examine the *CAH_count* field of the **ChunkArrayHeader**. This field is the first word of the **ChunkArrayHeader** (and therefore of the chunk). It contains the number of elements in the chunk array.

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed).

See Also: ChunkArrayHeader

■ ChunkArrayGetCountHandles()

```
word      ChunkArrayGetCountHandles(
          MemHandle          mh,          /* Handle of LMem heap's block */
          ChunkHandle        ch);         /* Handle of chunk array */
```

This routine is just like **ChunkArrayGetCount()**, except that the chunk array is specified by its global and local handles (instead of with an optr).

Include: chunkarr.h

■ ChunkArrayGetElement()

```
void      ChunkArrayGetElement(
          optr   array,                   /* optr to chunk array */
          word   elementNumber,           /* Index of element to copy */
          void * buffer);                 /* Address to copy element to */
```

This routine copies an element in a chunk array into the passed buffer. It is your responsibility to make sure the buffer is large enough to hold the element.

Include: chunkarr.h

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed). Make sure the buffer is large enough to hold the element.

See Also: ChunkArrayPtrToElement(), ChunkArrayElementToPtr()

■ ChunkArrayGetElementHandles()

```
void      ChunkArrayGetElementHandles(
          Memhandle          mh,             /* Handle of LMem heap's block */
          ChunkHandle        array,          /* Handle of chunk array */
          word               elementNumber,  /* Index of element to copy */
          void *             buffer);        /* Address to copy element to */
```

This routine is just like **ChunkArrayGetElement()**, except that the chunk array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

Be Sure To: Lock the LMem heap's block on the global heap (unless it is fixed). Make sure the buffer is large enough to hold the element.

See Also: ChunkArrayPtrToElement(), ChunkArrayElementToPtr()

■ ChunkArrayInsertAt()

```
void *   ChunkArrayInsertAt(
         optr              array,        /* Handle of chunk array */
         void *            insertPointer,/* Address at which to insert
                                          * element */
         word              elementSize); /* Size of new element (ignored
                                          * if elements are uniform-sized) */
```

This routine inserts a new element in a chunk array. You specify the location by passing a pointer to an element. A new element will be allocated at that location; thus, the element which was pointed to will be shifted, so it ends up immediately after the new element. The new element will be zero-initialized.

The routine is passed three arguments: the optr of the array, the address where the new element should be inserted, and the size of the new element. (If the array is of uniform-size elements, the size argument will be ignored.)

Include: chunkarr.h

Tips and Tricks: Only the chunk-handle portion of the optr is significant; the memory block is implicit in the pointer to the element.

Be Sure To: Lock the block on the global heap (if it is not fixed).

Warnings: If the address passed is not the address of an element already in the chunk array, results are undefined. The routine may cause heap compaction or resizing; all pointers within the block are invalidated.

See Also: ChunkArrayAppend(), ChunkArrayDelete(), ChunkArrayResize()

■ ChunkArrayInsertAtHandle()

```
void *    ChunkArrayInsertAtHandle(
          ChunkHandle         chunk,       /* Handle of chunk array */
          void *              insertPointer,/* Address at which to insert
                                             * element */
          word                elementSize); /* Size of new element (ignored
                                             * if elements are uniform-sized) */
```

This routine is just like **ChunkArrayInsertAt()**, except that the chunk array is specified by its chunk handle. (The global block is implicit in the pointer passed.)

Include: chunkarr.h

■ ChunkArrayPtrToElement()

```
word      ChunkArrayPtrToElement(
          optr   array,                         /* Handle of chunk array */
          void * element);                      /* Address of element */
```

This routine takes the address of an element in a chunk array, as well as an optr to the array. It returns the element's zero-based index.

Include: chunkarr.h

Tips and Tricks: Only the chunk-handle portion of the optr is significant; the memory block is implicit in the pointer to the element.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: If the address passed is not the address of the beginning of an element, results are unpredictable.

See Also: ChunkArrayElementToPtr()

■ ChunkArrayPtrToElementHandle()

```
word      ChunkArrayPtrToElementHandle(
          ChunkHandle         array,    /* chunk handle of chunk array */
          void *              element); /* Pointer to element to delete */
```

This routine is exactly like **ChunkArrayPtrToElement()**, except that the chunk array is indicated by its chunk handle. (The global block is implicit in the pointer passed.)

■ ChunkArraySort()

```
void      ChunkArraySort(
          optr                array,          /* optr to chunk array */
          word                valueForCallback,/* Passed to callback routine */
          sword _pascal (*callback)  (void *el1,
                                   void * el2,
                                   word valueForCallback))
            /* Sign of return value decides order of elements */
```

This is a general-purpose sort routine for chunk arrays. It does a modified Quicksort on the array, using an insertion sort for subarrays below a certain size; this gives performance of $O(n \log n)$.

The sort routine does not compare elements. Rather, it calls a comparison callback routine passed in the *callback* parameter. Whenever it needs to compare two elements, it calls the callback routine, passing the addresses of the elements and the *valueForCallback* word which was passed to **ChunkArraySort()**. The callback routine should be declared _pascal. The callback routine's return value determines which element will come first in the sorted array:

- ◆ If element *el1* ought to come before *el2* in the sorted array, the callback routine should return a negative integer.
- ◆ If element *el1* ought to come after *el2* in the sorted array, the callback routine should return a positive integer.
- ◆ If it doesn't matter whether *el1* comes before or after *el2* in the sorted array, the callback routine should return zero.

Include: chunkarr.h

Tips and Tricks: You may need to sort an array based on different criteria at different times. The simplest way to do this is to write one general-purpose callback routine and have the *valueForCallback* word determine how the sort is done. For example, the same callback routine could sort the array in ascending or descending order, depending on the *valueForCallback*.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: Do not have the callback routine do anything which might invalidate pointers to the array (such as allocate a new chunk or element).

See Also: ArrayQuickSort()

■ ChunkArraySortHandles()

```
void      ChunkArraySortHandles(
          MemHandle           memHandle,       /* Handle of LMem heap’s block */
          ChunkHandle         chunkHandle,     /* Handle chunk array */
          word                valueForCallback,/* Passed to callback routine */
          sword _pascal(*callback)(void *el1, void * el2, word valueForCallback)
                 /* Sign of return value decides order of elements */
```

This routine is exactly like **ChunkArraySort()** above, except that the chunk array is specified by its global and chunk handles (instead of by an optr).

Include: chunkarr.h

■ ChunkArrayZero()

```
void      ChunkArrayZero(
          optr   array);         /* optr to chunk array */
```

This routine destroys all the elements in an array. It does not affect the extra-space area between the **ChunkArrayHeader** and the elements. It is guaranteed not to cause heap compaction or resizing; thus, pointers to other chunks remain valid.

Include: chunkarr.h

Be Sure To: Lock the block on the global heap (unless it is fixed).

See Also: ChunkArrayDelete()

■ ChunkArrayZeroHandles()

```
void      ChunkArrayZeroHandles(
          MemHandle           mh     /* Global handle of LMem heap */
          ChunkHandle         ch);   /* Chunk handle of chunk array */
```

This routine is exactly like **ChunkArrayZero()** above, except that the chunk array is specified by its global and chunk handles (instead of by an optr).

Include: chunkarr.h

■ ClipboardAbortQuickTransfer()

```
void      ClipboardAbortQuickTransfer(void);
```

This routine cancels a quick-transfer operation in progress. It is typically used when an object involved in a quick-transfer is shutting down or when an error occurs in a quick-transfer. This routine is usually used only by the object or Process which initiated the quick-transfer.

Include: clipbrd.goh

■ ClipboardAddToNotificationList()

```
void    ClipboardAddToNotificationList(
        optr  notificationOD);
```

This routine registers the passed object or process for quick-transfer notification. This routine is typically called from within an object's MSG_META_INITIALIZE handler or within a Process object's MSG_GEN_PROCESS_OPEN_APPLICATION handler. Pass the optr of the object or the geode handle if the Process object should be registered.

Include: clipbrd.goh

See Also: ClipboardRemoveFromNotificationList()

■ ClipboardClearQuickTransferNotification()

```
void    ClipboardClearQuickTransferNotification(
        optr  notificationOD);
```

This routine removes an object or process from quick-transfer notification. It is typically used in the object's MSG_META_DETACH handler or in the Process object's MSG_GEN_PROCESS_CLOSE_APPLICATION to ensure that it is not notified after it has already detached.

Pass the optr of the object specified to receive notification in **ClipboardStartQuickTransfer()** (or the geode handle if a process).

Note that an object may also want to check if a quick-transfer is in progress when detaching and possibly abort it if there is one.

See Also: **clipbrd.goh**

■ ClipboardDoneWithItem()

```
void    ClipboardDoneWithItem(
        TransferBlockID header);
```

This routine is called when an object or Process is done using a transfer item. It relinquishes exclusive access to the item's transfer VM file after the caller had previously called **ClipboardQueryItem()**.

Include: clipbrd.goh

■ ClipboardEndQuickTransfer()

```
void      ClipboardEndQuickTransfer(
          ClipboardQuickNotifyFlags  flags);
```

This routine ends a quick-transfer operation by resetting the pointer image, clearing any quick-transfer region, clearing the quick-transfer item, and sending out any needed notification of the completed transfer.

Pass this routine a record of **ClipboardQuickNotifyFlags**. Pass the value CQNF_MOVE if the operation was completed and was a move; pass CQNF_COPY if the operation was completed and was a copy. If the operation could not be completed (e.g. incompatible data types), pass CQNF_NO_OPERATION or CQNF_ERROR.

The notification sent out by the UI will be in the form of the message MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_CONCLUDED. This message notifies the originator of the transfer item of the type of operation; the originator can then respond if necessary.

Include: clipbrd.goh

■ ClipboardEnumItemFormats()

```
word      ClipboardEnumItemFormats(
          TransferBlockID     header,
          word                maxNumFormats,
          ClipboardFormatID * buffer);
```

This routine returns a list of all the formats supported by the current transfer item. To see whether a particular format is supported, you can use **ClipboardTestItemFormat()** instead.

Pass this routine the following:

header The transfer item header as returned by **ClipboardQueryItem()**.

maxNumFormats
The maximum number of formats that should be returned. You should set your return buffer (see below) large enough to support this size.

buffer A pointer to a locked or fixed buffer into which the formats will be copied. Upon return, the buffer will contain the proper number of **ClipboardFormatID** structures, one for each format available. This buffer should be at least large enough to support the number of formats requested in *maxNumFormats*.

The word return value is the total number of formats returned. This number will be equal to or less than the number passed in *maxNumFormats*. The routine will also return the passed buffer filled with that number of **ClipboardFormatID** structures.

Include: clipbrd.goh

See Also: ClipboardTestItemFormat()

■ ClipboardGetClipboardFile()

```
VMFileHandle ClipboardGetClipboardFile(void);
```

This routine returns the VM file handle of the current default transfer VM file.

Include: clipbrd.goh

■ ClipboardGetItemInfo()

```
optr      ClipboardGetItemInfo(
          TransferBlockID header);
```

This routine returns the source identifier (*CIH_sourceID*) of the current transfer item. Pass the transfer item's header returned by **ClipboardQueryItem()**.

Include: clipbrd.goh

■ ClipboardGetNormalItemInfo()

```
TransferBlockID ClipboardGetNormalItemInfo(void);
```

This routine returns information about the normal transfer item. It returns a **TransferBlockID** dword which contains the VM file handle of the transfer file and the VM block handle of the transfer item's header block.

To extract the file handle from the return value, use the macro **FileFromTransferBlockID()**. To extract the block handle, use the macro **BlockFromTransferBlockID()**.

Include: clipbrd.goh

■ ClipboardGetQuickItemInfo()

```
TransferBlockID ClipboardGetQuickItemInfo(void);
```

This routine returns information about the quick-transfer transfer item. It returns a **TransferBlockID** dword which contains the VM file handle of the transfer file and the VM block handle of the transfer item's header block.

To extract the file handle from the return value, use the macro **FileFromTransferBlockID()**. To extract the block handle, use the macro **BlockFromTransferBlockID()**.

Include: clipbrd.goh

■ ClipboardGetQuickTransferStatus()

```
Boolean  ClipboardGetQuickTransferStatus(void);
```

This routine returns *true* if a quick-transfer operation is in progress, *false* otherwise. It is often called when objects or Processes are shutting down in order to abort any quick-transfers originated by the caller.

Include: clipbrd.goh

■ ClipboardGetUndoItemInfo()

```
TransferBlockID ClipboardGetUndoItemInfo(void);
```

This routine returns information about the undo transfer item. It returns a **TransferBlockID** dword which contains the VM file handle of the transfer file and the VM block handle of the transfer item's header block.

To extract the file handle from the return value, use the macro **FileFromTransferBlockID()**. To extract the block handle, use the macro **BlockFromTransferBlockID()**.

Include: clipbrd.goh

■ ClipboardQueryItem()

```
void     ClipboardQueryItem(
         ClipboardItemFlags     flags,
         ClipboardQueryArgs *   retValues);
```

This routine locks the transfer item for the caller's exclusive access and returns information about the current transfer item. You should call this routine when beginning any paste or clipboard query operation. For operations in which you will change the clipboard's contents, you should instead use the routine **ClipboardRegisterItem()**.

Pass the following values:

flags	A record of **ClipboardItemFlags** indicating the transfer item you want to query. Use CIF_QUICK to get information on the quick transfer item, and pass zero (or TIF_NORMAL) to get information on the normal transfer item.

retValues A pointer to an empty **ClipboardQueryArgs** structure into which return information about the transfer item will be passed. This structure is defined as follows:

```
typedef struct {
      word                CQA_numFormats;
      optr                CQA_owner;
      TransferBlockID     CQA_header;
} ClipboardQueryArgs;
```

The *CQA_header* field of **ClipboardQueryArgs** is used as a pass value to several other clipboard routines. It contains the VM file handle of the transfer VM file and the VM block handle of the transfer item's header block. The *CQA_owner* field is the optr of the object that originated the transfer item. The *CQA_numFormats* field specifies the total number of formats available for this transfer item. To see if a particular format is supported by the transfer item, call the routine **ClipboardTestItemFormat()**.

Be Sure To: You must call **ClipboardDoneWithItem()** when you are done accessing the transfer item. This routine relinquishes your exclusive access to the transfer VM file.

Include: clipbrd.goh

See Also: ClipboardRequestItemFormat(), ClipboardDoneWithItem**()**

■ ClipboardRegisterItem()

```
Boolean   ClipboardRegisterItem(
          TransferBlockID header,
          ClipboardItemFlags flags);
```

This routine completes a change to the transfer item. You should use this routine whenever copying or cutting something into the clipboard or whenever attaching something as the quick-transfer item.

This routine puts the item specified by *header* into the transfer VM file. It frees any transfer item that may already be in the file. Pass this routine the following:

header Header information for the item, consisting of the transfer VM file handle and the VM block handle of the block containing the new transfer item. Create the **TransferBlockID** structure using the macro **BlockIDFromFileAndBlock()**.

flags A record of **ClipboardItemFlags** indicating whether you're registering a clipboard item or a quick-transfer item. The flag CIF_QUICK indicates the item is a quick-transfer item; zero (or TIF_NORMAL) indicates the item is a normal clipboard item.

Include: clipbrd.goh

See Also: ClipboardRequestItemFormat()

■ ClipboardRemoveFromNotificationList()

```
Boolean  ClipboardRemoveFromNotificationList(
         optr   notificationOD);
```

This routine removes an object or Process from the clipboard's change notification list. It is typically called when the object or Process is being detached or destroyed. Pass it the same optr that was added to the notification list with **ClipboardAddToNotificationList**().

This routine returns an error flag: The flag will be *true* if the object could not be found in the notification list, *false* if the object was successfully removed from the list.

Include: clipbrd.goh

See Also: ClipboardAddToNotificationList()

■ ClipboardRequestItemFormat()

```
void     ClipboardRequestItemFormat(
         ClipboardItemFormatID  format,
         TransferBlockID        header,
         ClipboardRequestArgs * retValue);
```

This routine returns specific information about a particular transfer item. Because some of the passed information must be retrieved with **ClipboardQueryItem()**, you must call **ClipboardQueryItem()** before calling this routine.

Pass this routine the following:

format — The manufacturer ID and format type of the new transfer item being put into the transfer VM file. Create the **ClipboardItemFormatID** value with the macro **FormatIDFromManufacturerAndType()**.

header — Header information for the item, consisting of the transfer VM file handle and the VM block handle of the block containing the new transfer item. Create the **TransferBlockID** structure using the macro **BlockIDFromFileAndBlock()** using returned information from **ClipboardQueryItem()**.

retValue — A pointer to an empty **ClipboardRequestArgs** structure that will be filled by the routine. This structure is defined as follows:

```
typedef struct {
      VMFileHandle CRA_file;
      VMChain      CRA_data;
      word         CRA_extra1;
      word         CRA_extra2;
} ClipboardRequestArgs;
```

Upon return, the *CRA_file* field will contain the transfer VM file's VM file handle and the *CRA_data* field will contain the VM block handle of the transfer item's header block. If there is no transfer item, *CRA_data* will be zero.

Include: clipbrd.goh

See Also: ClipboardRegisterItem(), ClipboardQueryItem()

■ ClipboardSetQuickTransferFeedback()

```
void    ClipboardSetQuickTransferFeedback(
        ClipboardQuickTransferFeedback   cursor,
        UIFunctionsActive                buttonFlags);
```

This routine sets the image of the mouse pointer during a quick-transfer operation. Use this routine to provide visual feedback to the user during the quick-transfer. For example, an object that could not accept the quick-transfer item would set the "no operation" cursor while the mouse pointer was over its bounds.

Pass the two following values:

cursor	A value of **ClipboardQuickTransferFeedback** type indicating the type of cursor to set. The possible values are listed below.
buttonFlags	A record of **UIFunctionsActive** flags. These flags are defined in the Input Manager section and deal with user override of the move/copy behavior.

The cursor parameter contains a value of **ClipboardQuickTransferFeedback**. This is an enumerated type that defines the cursor to be set, and it has the following values:

CQTF_MOVE	This sets the cursor to the specific UI's move cursor.
CQTF_COPY	This sets the cursor to the specific UI's copy cursor.
CQTF_CLEAR	This clears the cursor and sets it to the specific UI's modal "no operation" cursor.

Include: clipbrd.goh

Routines

■ ClipboardStartQuickTransfer()

```
Boolean  ClipboardStartQuickTransfer(
         ClipboardQuickTransferFlags          flags,
         ClipboardQuickTransferFeedback       initialCursor,
         word                                 mouseXPos,
         word                                 mouseYPos,
         ClipboardQuickTransferRegionInfo *   regionParams,
         optr                                 notificationOD);
```

This routine signals the beginning of a quick-transfer operation. Typically, an object or process will call this routine in its MSG_META_START_MOVE_COPY handler.

Pass it the following parameters:

flags A record of **ClipboardQuickTransferFlags** indicating whether an addition graphic region will be attached to the cursor and whether the caller wants notification of transfer completion. The flags allowed are listed below, after the parameter list.

initialCursor
The initial cursor to use for visual feedback to the user. It is a value of **ClipboardQuickTransferFeedback**, either CQTF_MOVE or CQTF_COPY. If -1 is passed in this parameter, the initial cursor will be the default no-operation cursor (i.e. the transfer source may not also act as the transfer destination).

mouseXPos This field is used only if CQTF_USE_REGION is passed in *flags*. It is the horizontal position of the mouse in screen coordinates.

mouseYPos This field is used only if CQTF_USE_REGION is passed in *flags*. It is the vertical position of the mouse in screen coordinates.

regionParams
A pointer to a **ClipboardQuickTransferRegionInfo** structure defining the graphical region to be attached to the cursor during the transfer operation. This structure is only required if CQTF_USE_REGION is passed in *flags*. It is defined below.

notificationOD
The optr of the object to be notified upon transfer completion. The object specified will receive the notification messages MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_CONCLUDED and MSG_..._FEEDBACK.

The allowed **ClipboardQuickTransferFlags** are listed below:

CQTF_COPY_ONLY
Source supports copying only (not cutting).

CQTF_USE_REGION
Source has passed the definition of a graphical region which will be attached to the tail of the quick-transfer cursor.

CQTF_NOTIFICATION
Source requires notification of completion of the transfer in order to cut original data or provide other feedback.

If a graphical region is to be attached to the quick-transfer cursor, you must pass a pointer to a **ClipboardQuickTransferRegionInfo** in the *regionParams* parameter. This structure is defined below.

```
typedef struct {
       word   CQTRI_paramAX;
       word   CQTRI_paramBX;
       word   CQTRI_paramCX;
       word   CQTRI_paramDX;
       Point  CQTRI_regionPos;
       dword  CQTRI_strategy;
       dword  CQTRI_region;
} ClipboardQuickTransferRegionInfo;
```

This structure is passed on the stack to the routine. The first four fields represent the region's definition parameters. *CQTRI_regionPos* is a **Point** structure indicating where (in screen coordinates) the region is to be located. *CQTRI_strategy* is a pointer to the region strategy routine. *CQTRI_strategy* should be a video driver strategy. To find out the strategy of the video driver associated with your window, send your object a MSG_VIS_VUP_QUERY with VUQ_VIDEO_DRIVER. Pass the handle thus gained to **GeodeInfoDriver()**, which will return the strategy.

This routine returns an error flag: If a quick-transfer is already in progress, the return will be *true*. If the quick-transfer is successfully begun, the error flag will be *false*.

Include: clipbrd.goh

■ ClipboardTestItemFormat()

```
Boolean   ClipboardTestItemFormat(
          TransferBlockID     header,
          ClipboardFormatID   format);
```

This routine tests whether the given format is supported by the specified transfer item. It returns *true* if the format is supported, *false* if the format is not supported. Pass the following values:

header	A **TransferBlockID** specifying the VM file handle and VM block handle of the transfer item to be checked. This is returned by the routines **ClipboardGetNormalItemInfo()**, **ClipboardGetQuickItemInfo()**, **ClipboardGetUndoItemInfo()**, and **ClipboardQueryItem()**. Most often the proper routine to use is **ClipboardQueryItem()**.
format	A **ClipboardFormatID** specifying the type and manufacturer ID of the format to be checked. It is most appropriate to create this parameter from its individual parts using the macro **FormatIDFromManufacturerAndType()**.

Include: clipbrd.goh

■ ClipboardUnregisterItem()

```
void      ClipboardUnregisterItem(
          optr   owner);
```

This routine restores the transfer item to what it was before the last **ClipboardRegisterItem()**. Pass it the optr of the caller.

Only the object that last registered a transfer item is allowed to unregister it. If the transfer item is owned by a different object, or if there is no transfer item, nothing will be done. If the transfer item is owned by the caller, the transfer item will be unregistered and the clipboard will be restored to its previous state.

Include: clipbrd.goh

■ ConstructOptr()

```
optr      ConstructOptr(
          Handle              han,
          ChunkHandle         ch);
```

This macro constructs an optr type from the given handle (typically a MemHandle) and chunk handle.

See Also: HandleToOptr(), OptrToHandle(), OptrToChunk()

■ DBAlloc()

```
DBItem   DBAlloc(
         VMFileHandle       file,
         DBGroup            group,
         word               size);
```

This routine allocates an item in the specified file and group. It is passed the handles for the file and group which will contain the new item. It returns the new item's item-handle.

Warnings: All pointers to items in the group may be invalidated.

Include: dbase.h

See Also: DBAllocUngrouped()

■ DBAllocUngrouped()

```
DBGroupAndItem DBAllocUngrouped(
         VMFileHandle       file,
         word               size);
```

This routine allocates an ungrouped item in the specified file. It is passed the handle of the file which will contain the new item. It returns the item's **DBGroupAndItem** value.

Warnings: All pointers to ungrouped items may be invalidated.

Include: dbase.h

See Also: DBAlloc()

■ DBCombineGroupAndItem()

```
DBGroupAndItem DBCombineGroupAndItem(
         DBGroup            group,
         DBItem             item);
```

This macro combines group and item handles into a dword-sized **DBGroupAndItem** value.

Include: dbase.h

See Also: DBGroupFromGroupAndItem(), DBItemFromGroupAndItem()

■ DBCopyDBItem()

```
DBItem   DBCopyDBItem(
         VMFileHandle        srcFile,
         DBGroup             srcGroup,
         DBItem              srcItem,
         VMFileHandle        destFile,
         DBGroup             destGroup);
```

This routine makes a duplicate of a DB item in the specified DB file and group. It is passed the file handle, group handle, and item handle of the source item, as well as the file handle and group handle of the destination group. It makes a copy of the DB item and returns its **DBItem** handle.

Warnings: All pointers to items in the destination group may be invalidated.

Include: dbase.h

See Also: VMCopyVMChain()

■ DBCopyDBItemUngrouped()

```
DBGroupAndItem DBCopyDBItemUngrouped(
         VMFileHandle        srcFile,
         DBGroupAndItem      srcID,        /* source item */
         VMFileHandle        destFile);
```

This routine makes a duplicate of a specified DB item. It is passed the file handle and **DBGroupAndItem** value specifying the source item, and the file handle of the destination file. It allocates the item as an ungrouped item in the specified file and returns its **DBGroupAndItem** value.

Tips and Tricks: If the source item is not ungrouped, you can combine the group and item handles into a **DBGroupAndItem** value by calling the macro **DBCombineGroupAndItem()**.

Warnings: All pointers to ungrouped items in the destination file may be invalidated.

Include: dbase.h

See Also: VMCopyVMChain()

■ DBDeleteAt()

```
void      DBDeleteAt(
          VMFileHandle        file,
          DBGroup             group,
          DBItem              item,
          word                deleteOffset,
          word                deleteCount);
```

This routine deletes a sequence of bytes from within an item. It does not invalidate pointers to other items. The routine is passed the file, group, and item handles specifying the item, as well as an offset within the item and a number of bytes to delete. It will delete the specified number of bytes from within the item, starting with the byte at the specified offset.

Include: dbase.h

■ DBDeleteAtUngrouped()

```
void      DBDeleteAtUngrouped(
          VMFileHandle        file,
          DBGroupAndItem      id,
          word                deleteOffset,
          word                deleteCount);
```

This routine is just like **DBDeleteAt()**, except it is passed a **DBGroupAndItem** value instead of separate group and item handles. It does not invalidate pointers to other items.

Include: dbase.h

■ DBDeref()

```
void *    DBDeref(
          optr                *ref);
```

This routine is passed an optr to a locked DB item. The routine returns the address of the item.

Warnings: The optr becomes invalid when the DB item is unlocked.

Include: dbase.h

■ DBDirty()

```
void      DBUnlock(
          const void *          ptr);
```

This routine marks a DB item as dirty; this insures that the VM manager will copy its item-block back to the disk before freeing its memory. The routine is passed a pointer to anywhere within the item.

Tips and Tricks: All the items in an item block are marked dirty at once; thus, you can call this routine just once for several items in the same item block. Only the segment portion of the pointer is significant; thus, you can pass a pointer to anywhere in the item. This is useful if you have incremented the pointer to the item.

Include: dbase.h

■ DBFree()

```
void      DBFree(
          VMFileHandle          file,
          DBGroup               group,
          DBItem                item);
```

This routine frees the specified item. It does not invalidate pointers to other items in the group. It is passed the file, group, and item handles specifying the item; it does not return anything.

Never Use Situations:

Never call **DBFree()** on a locked item. If you do, the item-block's lock count will not be decremented, which will prevent the item block from ever being properly unlocked.

Include: dbase.h

See Also: DBFreeUngrouped()

■ DBFreeUngrouped()

```
void      DBFreeUngrouped(
          VMFileHandle          file,
          DBGroupAndItem        id);
```

This routine frees the specified item. It does not invalidate pointers to other ungrouped items. It is passed the file handle and **DBGroupAndItem** value specifying the item; it does not return anything.

Never Use Situations: Never call **DBFreeUngrouped()** on a locked item. If you do, the item-block's lock count will not be decremented, which will prevent the item block from ever being properly unlocked.

Include: dbase.h

See Also: DBFree()

■ DBGetMap()

```
DBGroupAndItem DBGetmap(
         VMFileHandle       file);
```

This routine returns the **DBGroupAndItem** structure for the passed file's map item. If there is no map item, it returns a null handle.

Include: dbase.h

See Also: DBSetMap(), DBLockMap()

■ DBGroupAlloc()

```
DBGroup  DBGroupAlloc(
         VMFileHandle       file);
```

This routine allocates a new DB group in the specified file and returns its handle. If the group cannot be allocated, **DBGroupAlloc()** returns a null handle.

Include: dbase.h

■ DBGroupFree()

```
void     DBGroupFree(
         VMFileHandle       file,
         DBGroup            group);
```

This routine frees the specified group. This deletes all items and item-blocks associated with the group. It is passed the file and group handle specifying the group. Note that you can free a group even if some of its items are locked; those locked items will also be freed.

Include: dbase.h

■ DBGroupFromGroupAndItem()

```
DBGroup  DBGroupFromGroupAndItem(
         DBGroupAndItem     id);
```

This macro returns the **DBGroup** portion of a **DBGroupAndItem** value.

Include: dbase.h

■ DBInsertAt()

```
void    DBInsertAt(
        VMFileHandle       file,
        DBGroup            group,
        DBItem             item,
        word               insertOffset,
        word               insertCount);
```

This routine inserts bytes at a specified offset within a DB item. The bytes are zero-initialized. It is passed the file, group, and item handles specifying a DB item, as well as an offset within the cell and a number of bytes to insert. It inserts the specified number of bytes beginning at the specified offset; the data which was at the passed offset will end up immediately after the inserted bytes.

Warnings: This routine invalidates pointers to other items in the same group.

Include: dbase.h

■ DBInsertAtUngrouped()

```
void    DBInsertAtUngrouped(
        VMFileHandle       file,
        DBGroupAndItem     id,
        word               insertOffset,
        word               insertCount);
```

This routine is just like **DBInsertAt()**, except it is passed a **DBGroupAndItem** value instead of separate group and item handles.

Warnings: This routine invalidates pointers to other ungrouped items.

Include: dbase.h

■ DBItemFromGroupAndItem()

```
DBItem  DBItemFromGroupAndItem(
        DBGroupAndItem     id);
```

This macro returns the **DBItem** portion of a **DBGroupAndItem** value.

Include: dbase.h

■ DBLock()

```
void *   DBLock(
         VMFileHandle        file,
         DBGroup             group,
         DBItem              item);
```

This routine locks the specified item and returns a pointer to it. It is passed the file, group, and item handles specifying a DB item. If it fails, it returns a null pointer.

Include: dbase.h

See Also: DBLockGetRef(), DBLockUngrouped()

■ DBLockGetRef()

```
void *   DBLockGetRef(
         VMFileHandle        file,
         DBGroup             group,
         DBItem              item,
         optr *              ref);
```

This routine is just like **DBLock()**, except that it writes the item's optr to the passed address.

Include: dbase.h

Warnings: The optr is only valid until the DB item is unlocked.

■ DBLockGetRefUngrouped()

```
void *   DBLockGetRefUngrouped(
         VMFileHandle        file,
         DBGroupAndItem      id,
         optr *              ref);
```

This routine is the same as **DBLockGetRef()**, except that it is passed a **DBGroupAndItem** value.

Include: dbase.h

■ DBLockMap()

```
void *   DBLockMap(
         VMFileHandle        file);
```

This routine locks the specified file's map item and returns its address. To unlock the map item, call **DBUnlock()** normally.

Include: dbase.h

See Also: DBLockMap()

■ DBLockUngrouped()

```
void *    DBLockUngrouped(
          VMFileHandle        file,
          DBGroupAndItem      id);
```

This routine is the same as **DBLock()**, except that it is passed a **DBGroupAndItem** value.

Include: dbase.h

■ DBReAlloc()

```
void      DBReAlloc(
          VMFileHandle        file,
          DBGroup             group,
          DBItem              item,
          word                size);
```

This routine changes the size of a DB item. It is passed the file, group, and item handles specifying the DB item, and a new size for the item (in bytes). If the new size is larger than the old, space will be added to the end of the item; if the new size is smaller than the old, the item will be truncated to fit.

Warnings: If the new size is larger than the old, all pointers to items in the group are invalidated. Space added is not zero-initialized.

Include: dbase.h

■ DBReAllocUngrouped()

```
void      DBReAllocUngrouped(
          VMFileHandle        file,
          DBGroupAndItem      id,
          word                size);
```

This routine is just like **DBReAlloc()**, except it is passed a **DBGroupAndItem** value instead of separate group and item handles.

Warnings: If the new size is larger than the old, all pointers to ungrouped items are invalidated. Space added is not zero-initialized.

Include: dbase.h

■ DBSetMap()

```
void     DBSetMap(
         VMFileHandle        file,
         DBGroup             group,
         DBItem              item);
```

This routine sets the DB map item. You can later retrieve a **DBGroupAndItem** structure identifying this item by calling **DBGetMap()**. The routine is passed the file, group, and item handles specifying the new map item; it does not return anything.

Include: dbase.h

■ DBSetMapUngrouped()

```
void     DBSetMapUngrouped(
         VMFileHandle        file,
         DBGroupAndItem      id);
```

This routine is just like **DBSetMap()**, except it is passed a **DBGroupAndItem** value instead of separate group and item handles.

Include: dbase.h

■ DBUnlock()

```
void     DBUnlock(
         void * ptr); /* address of item to unlock */
```

This routine unlocks the DB item whose address is passed.

Tips and Tricks: Only the segment address of the pointer is significant. Thus, you can pass a pointer to somewhere within an item (or immediately after it) to unlock it.

Be Sure To: If the item has been changed, make sure you call **DBDirty()** *before* you unlock it.

Include: dbase.h

■ DiskCheckInUse()

```
Boolean  DiskCheckInUse(
         DiskHandle          disk);
```

This routine checks if a registered disk is being used. If a file on that disk is open, or if a path on that disk is on some thread's directory stack, the routine will return *true* (i.e. non-zero); otherwise it will return *false* (i.e. zero). Note that a disk may be "in use" even if it is not currently in any drive.

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file (which is guaranteed to be in use).

Include: disk.h

■ DiskCheckUnnamed()

```
Boolean   DiskCheckUnnamed( /* returns true if disk is unnamed */
          DiskHandle         disk);
```

This routine checks if a registered disk has a permanent name. If the disk does not have a name, the routine returns *true* (i.e. non-zero); otherwise it returns *false*. Note that GEOS assigns a temporary name to unnamed disks when they are registered. To find out a disk's temporary or permanent name, call **DiskGetVolumeName()**.

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

See Also: DiskGetVolumeName()

Include: disk.h

■ DiskCheckWritable()

```
Boolean   DiskCheckWritable(
          DiskHandle         disk);
```

DiskCheckWritable() checks if a disk is currently writable. It returns *false* (i.e. zero) if the disk is not writable, whether by nature (e.g. a CD-ROM disk) or because the write-protect tab is on; otherwise it returns *true* (i.e. non-zero).

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

Include: disk.h

■ DiskCopy()

```
DiskCopyError DiskCopy(
          word                source,
          word                dest,
          Boolean _pascal (*callback)
                 (DiskCopyCallback          code,
                  DiskHandle                disk,
                  word                      param));
```

This routine copies one disk onto another. The destination disk must be formattable to be the same type as the source disk. The first two arguments specify the source and destination drive. These drives may or may not be the same. If they are different, they must take compatible disks.

A disk copy requires frequent interaction with the user. For example, the copy routine must prompt the user to swap disks when necessary. For this reason, **DiskCopy()** is passed a pointer to a callback routine. This routine handles all interaction with the user. It must be declared _pascal. Each time it is called, it is passed three arguments. The first is a member of the **DiskCopyCallback** enumerated type; this argument specifies what the callback routine should do. The second argument is a disk handle; its significance depends on the value of the **DiskCopyCallback** argument. The third argument is a word-sized piece of data whose significance depends on the value of the **DiskCopyCallback** argument. Note that either of these arguments may be null values, depending on the value of the **DiskCopyCallback** argument.

The callback routine can abort the copy by returning *true* (i.e. non-zero); otherwise, it should return *false* (i.e. zero). The callback routine is called for several situations, identified by the value of **DiskCopyCallback** associated with them:

CALLBACK_GET_SOURCE_DISK
: The callback routine should prompt the user to insert the source disk into the appropriate drive. The second argument is meaningless for this call. The third argument is the number identifying the drive; use **DriveGetName()** to find the name for this drive.

CALLBACK_GET_DEST_DISK
: The callback routine should prompt the user to insert the destination disk into the appropriate drive. The second argument is meaningless for this call. The third argument is the number identifying the drive.

CALLBACK_REPORT_NUM_SWAPS
: The second argument is meaningless for this call. The third argument is the number of disk swaps that will be necessary. The callback routine may wish to report this number to the user and ask for confirmation.

CALLBACK_VERIFY_DEST_DESTRUCTION
: If the destination disk has already been formatted, the callback routine will be called with this parameter. The callback routine may wish to remind the user that the destination disk will be erased. The second argument is the handle of the destination disk; this is useful if, for example, you want to report the disk's name. The third argument is the destination drive's number. If the callback routine aborts the copy at this time by returning non-zero, the destination disk will not be harmed.

CALLBACK_REPORT_FORMAT_PERCT
: If the destination disk needs to be formatted, **DiskCopy()** will periodically call the callback routine with this parameter. The callback routine may wish to notify the user how the format is progressing. In this case, the second argument will be meaningless; the third parameter will be the percentage of the destination disk which has been formatted. The callback routine may wish to notify the user how the format is progressing.

CALLBACK_REPORT_COPY_PERCT
: While the copy is taking place, **DiskCopy()** will periodically call the callback routine with this parameter. The callback routine may wish to notify the user how the copy is progressing. In this case, the second parameter will be meaningless; the third parameter will be the percentage of the copy which has been completed.

If the copy was successful, **DiskCopy()** returns zero. Otherwise, it returns a member of the **DiskCopyErrors** enumerated type. That type has the following members:

ERR_DISKCOPY_INSUFFICIENT_MEM
: This is returned if the routine was unable to get adequate memory.

ERR_CANT_COPY_FIXED_DISKS

ERR_CANT_READ_FROM_SOURCE

ERR_CANT_WRITE_TO_DEST

ERR_INCOMPATIBLE_FORMATS
: The destination drive must be able to write disks in exactly the same format as the source disk. Note that the source and destination drives may be the same.

ERR_OPERATION_CANCELLED
: This is returned if the callback routine ever returned a non-zero value, thus aborting the copy.

ERR_CANT_FORMAT_DEST

Include: disk.h

■ DiskFind()

```
DiskHandle DiskFind(
          const char *        fname,    /* Null-terminated volume name */
          DiskFindResult *    code);    /* DiskFindResult written here */
```

This routine returns the handle of the disk with the specified name. If there is no registered disk with the specified name, **DiskFind()** returns a null handle. Note that while disk handles are unique, volume names are not; therefore, there may be several registered disks with identical volume names. For this reason, **DiskFind()** writes a member of the **DiskFindResults** enumerated type (described below) into the space pointed to by the *code* pointer.

Structures: **DiskFind()** uses the **DiskFindResults** enumerated type, which has the following values:

DFR_UNIQUE
: There is exactly one registered disk with the specified name; its handle was returned.

DFR_NOT_UNIQUE
: There are two or more registered disks with the specified name; the handle of an arbitrary one of these disks was returned.

DFR_NOT_FOUND
: There are no registered disks with the specified name; a null disk handle was returned.

Tips and Tricks: If you want to find all the disks with a given volume name, call **DiskForEach()** and have the callback routine check each disk’s name with **DiskGetVolumeName()**.

See Also: DiskRegisterDisk()

Include: disk.h

■ DiskForEach() (See Section 17.3.2.2 of the Concepts book)

```
DiskHandle DiskForEach(
          Boolean _pascal (* callback)(DiskHandledisk)) /* callback returns true
                                                  * to cancel */
```

This routine lets you perform an action on every registered disk. It calls the callback routine once for each disk, passing the disk's handle. The callback routine must be declared _pascal. The callback routine can force an early termination by returning *true* (i.e. non-zero). If the callback routine ever returns *true*, **DiskForEach()** terminates and returns the handle of the last disk passed to the callback routine. If the callback routine examines every disk without returning *true*, **DiskForEach()** returns a null handle.

Tips and Tricks: **DiskForEach()** is commonly used to look for a specific disk. The callback routine checks each disk to see if it's the one; if it finds a match, the callback routine simply returns *true*, and **DiskForEach()** returns the disk's handle.

Include: disk.h

■ DiskFormat()

```
FormatError DiskFormat(
          word                driveNumber,
          MediaType           media,       /* Format to this size */
          word                flags,       /* See flags below */
          dword               *goodClusters,   /* These are filled in at the */
          dword               *badClusters,    /* of the format */
          Boolean _pascal (*callback)
                              (word percentDone));/* Return true to cancel */
```

This routine formats a disk to the specified size. When it is finished, it fills in the passed pointers to contain the number of good and bad clusters on the disk. (To find out the size of each cluster, call **DiskGetVolumeInfo()**.) The routine returns a member of the **FormatError** enumerated type (whose members are described below).

DiskFormat() can be instructed to call a callback routine periodically. This allows the application to keep the user informed about how the format is progressing. The callback routine is passed either the percent of the disk which has been formatted, or the cylinder and head currently being formatted. The callback routine must be declared _pascal. The callback routine can cancel the format by returning *true* (i.e. non-zero); otherwise, it should return *false* (i.e. zero).

The third argument passed is a word-length flag field. Currently, only three flags are defined:

DFF_CALLBACK_PERCENT_DONE
: A callback routine should be called periodically. The callback routine should be passed a single argument, namely the percentage of the format which has been done.

DFF_CALLBACK_CYL_HEAD
: A callback routine should be called periodically. The callback routine should be passed a single argument, namely the cylinder head being formatted. If both DFF_CALLBACK_PERCENT_DONE and DFF_CALLBACK_CYL_HEAD are passed, results are undefined. If neither flag is set, the callback routine will never be called; a null function pointer may be passed.

DFF_FORCE_ERASE
: A "hard format" should be done, i.e. the sectors should be rewritten and initialized to zeros. If this flag is not set, **DiskFormat()** will do a "soft format" if possible; it will check the sectors and write a blank file allocation table, but it will not necessarily erase the data from the disk.

DiskFormat() returns a member of the **FormatErrors** enumerated type. If the format was successful, it will return the constant FMT_DONE (which is guaranteed to equal zero). Otherwise, it will return one of the following constants:

```
FMT_DRIVE_NOT_READY
FMT_ERROR_WRITING_BOOT
FMT_ERROR_WRITING_ROOT_DIR
FMT_ERROR_WRITING_FAT
FMT_ABORTED
FMT_SET_VOLUME_NAME_ERROR
FMT_CANNOT_FORMAT_FIXED_DISKS_IN_CUR_RELEASE
FMT_BAD_PARTITION_TABLE
FMT_ERR_NO_PARTITION_FOUND
FMT_ERR_CANNOT_ALLOC_SECTOR_BUFFER
FMT_ERR_DISK_IS_IN_USE
FMT_ERR_WRITE_PROTECTED
FMT_ERR_DRIVE_CANNOT_SUPPORT_GIVEN_FORMAT
FMT_ERR_INVALID_DRIVE_SPECIFIED
FMT_ERR_DRIVE_CANNOT_BE_FORMATTED
FMT_ERR_DISK_UNAVAILABLE
```

Include: disk.h

Routines

■ DiskGetDrive()

```
word      DiskGetDrive(
          DiskHandle          dh);
```

This routine returns the drive number associated with a registered disk. Note that it will do this even if the drive is no longer usable (e.g. if a network drive has been unmapped).

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

See Also: DiskFind(), DiskRegisterDisk()

Include: disk.h

■ DiskGetVolumeFreeSpace()

```
dword     DiskGetVolumeFreeSpace(
          DiskHandle          dh);
```

This routine returns the amount of free space (measured in bytes) on the specified disk. If the disk is, by nature, not writable (e.g. a CD-ROM disk), **DiskGetVolumeFreeSpace()** returns zero and clears the thread's error value. If an error condition exists, **DiskGetVolumeFreeSpace()** returns zero and sets the thread's error value.

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

See Also: DiskGetVolumeInfo()

Include: disk.h

■ DiskGetVolumeInfo()

```
word      DiskGetVolumeInfo(  /* Returns 0 if successful */
          DiskHandle          dh,
          DiskInfoStruct      *info);   /* Routine fills this structure */
```

This routine returns general information about a disk. It returns the following four pieces of information:

- ◆ The size of each disk block in bytes. When space is allocated, it is rounded up to the nearest whole block.
- ◆ The number of free bytes on the disk.
- ◆ The total number of bytes on the disk; this is the total of free and used space.

- The disk's volume name. If the volume is unnamed, the current temporary name will be returned.

The information is written to the passed **DiskInfoStruct**. If an error condition occurs, **DiskGetVolumeInfo()** will return the error code and set the thread's error value; otherwise, it will return zero.

Structures: The routine writes the information to a **DiskInfoStruct**:

```
typedef struct {
      word          DIS_blockSize;
      sdword        DIS_freeSpace;
      sdword        DIS_totalSpace;
      VolumeName    DIS_name;
} DiskInfoStruct;
```

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

Include: disk.h

DiskGetVolumeName()

```
void      DiskGetVolumeName(
          DiskHandle          dh,
          char *              buffer);  /* Must be VOLUME_NAME_LENGTH_ZT bytes
                                          * long */
```

This routine copies the disk's volume name (as a null-terminated string) to the passed buffer. If an error occurs, it sets the thread's error value. If the volume has no name, the routine returns the current temporary name.

Warnings: **DiskGetVolumeName()** does not check the size of the buffer passed. If the buffer is not at least VOLUME_NAME_LENGTH_ZT bytes long, the routine may write beyond its boundaries.

Tips and Tricks: If you pass a standard path constant, this routine will return information about the disk containing the main **geos.ini** file.

See Also: DiskGetVolumeInfo(), DiskSetVolumeName()

DiskRegisterDisk()

```
DiskHandle DiskRegisterDisk(
          word   driveNumber);
```

This routine registers a disk in the specified drive and assigns it a disk handle. (The disk handle persists only to the end of the current session of GEOS.) If the disk already has a handle, **DiskRegisterDisk()** will return it.

If the disk does not have a name, GEOS will assign it a temporary name (such as "UNNAMED1") and display an alert box telling the user what the temporary name is. (This is done only the first time the disk is registered in each session.) Note that the temporary name is not written to the disk; thus, it persists only until the end of the current session of GEOS.

If this routine returns a disk handle, there's a disk in the drive; if it doesn't, there may still be a disk in the drive, but the disk is unformatted.

Tips and Tricks: There is no harm in registering the same disk several times. Thus, if you want to get the disk handle for the disk in a specific drive, you can simply call **DiskRegisterDisk()**.

See Also: DiskRegisterDiskSilently()

Include: disk.h

■ DiskRegisterDiskSilently()

```
DiskHandle DiskRegisterDiskSilently(
          word                driveNumber);
```

This routine is almost identical to **DiskRegisterDisk()** (described immediately above). There is only one difference: If GEOS assigns a temporary name to the disk, it will not present an alert box to the user.

See Also: DiskRegisterDisk()

Include: disk.h

■ DiskRestore()

```
DiskHandle DiskRestore(
          void *              buffer,   /* buffer written by DiskSave() */
          DiskRestoreError _pascal (*callback)
                              (const char         *driveName,
                               const char         *diskName,
                               void               **bufferPtr,
                               DiskRestoreError   error);
```

DiskRestore() examines a buffer written by **DiskSave()** and returns the handle of the disk described by that buffer. If that disk is already registered, **DiskRestore()** will simply return its handle. If the disk is not registered and is not in the drive, **DiskRestore()** will call the specified callback routine. The callback routine should be declared _pascal. The callback routine is passed four arguments:

- ◆ A null-terminated string containing the name of the drive for the disk.

- A null-terminated string containing the disk's volume label.
- A pointer to a variable in the **DiskRestore()** routine. This variable is itself a pointer to the opaque data structure provided by **DiskSave()**. If the callback routine takes any action which causes that structure to move (e.g. if it causes the global or local heap containing the buffer to be shuffled), it should update the pointer in **DiskRestore()**.
- A member of the **DiskRestoreError** enumerated type. This is the error which **DiskRestore()** would have returned if there had not been a callback routine. This is usually DRE_REMOVABLE_DRIVE_DOESNT_HOLD_DISK.

The callback routine should prompt the user to insert a disk. If the callback routine was successful, it should return DRE_DISK_IN_DRIVE (which is guaranteed to be equal to zero). Otherwise, it should return a member of the **DiskRestoreError** enumerated type; usually it will return DRE_USER_CANCELLED_RESTORE. Note that the callback routine will not generally know if the user has inserted a disk; it generally just displays an alert box and returns when the user clicks "OK." After the callback routine returns, **DiskRestore()** registers the disk and makes sure that it's the correct one; if it is not, it calls the callback routine again.

You can pass a null function pointer to **DiskRestore()** instead of providing a callback routine. In this case, **DiskRestore()** will fail if the disk has not been registered and is not currently in the drive.

DiskRestore() returns the handle of the disk. If it fails for any reason, it returns a null handle and sets the thread's error value to a member of the **DiskReturnError** enumerated type. This type has the following members:

DRE_DISK_IN_DRIVE
: This is returned by the callback routine. This is guaranteed to equal zero.

DRE_DRIVE_NO_LONGER_EXISTS
: The disk is associated with a drive which is no longer attached to the system.

DRE_REMOVABLE_DRIVE_DOESNT_CONTAIN_DISK
: The disk is unregistered, and it is not currently in the drive associated with it. If a callback routine was provided, **DiskRestore()** will call it under these circumstances.

DRE_USER_CANCELLED_RESTORE
: This is returned by the callback routine if the user cancels the restore.

Routines

DRE_COULDNT_CREATE_NEW_HANDLE
: **DiskRestore()** was unable to register the disk in the appropriate drive because it couldn't create a new disk handle.

DRE_REMOVABLE_DRIVE_IS_BUSY
: The appropriate drive is busy with a time-consuming operation (e.g. a disk format).

See Also: DiskSave()

Include: disk.h

■ DiskSave()

```
Boolean  DiskSave(
         DiskHandle          disk,
         void *              buffer,      /* data will be written here */
         word *              bufferSize); /* Size of buffer (in bytes) */
```

This routine writes information about a disk in the specified buffer. **DiskRestore()** can use this information to return the disk handle, even in another session of GEOS. The *bufferSize* argument should point to a word containing the size of the buffer (in bytes). If the buffer is large enough, **DiskSave()** will write an opaque data structure into the buffer, and change the value of **bufferSize* to the actual size of the data structure; any extra buffer space can be freed or otherwise used. In this case, **DiskSave()** will return *true* (i.e. non-zero). If the buffer was too small, **DiskSave()** will return *false* (i.e. zero) and write the size needed into **bufferSize*. Simply call **DiskSave()** again with a large enough buffer. If **DiskSave()** failed for some other reason, it will return *false* and set **bufferSize* to zero.

See Also: DiskRestore()

Include: disk.h

■ DiskSetVolumeName()

```
word     DiskSetVolumeName(
         DiskHandle          dh,
         const char *        name);    /* Change the name to this */
```

This routine changes the disk's volume label. If it is successful, it returns zero; otherwise it returns an error code. It also sets or clears the thread's error value appropriately. The following error codes may be returned:

ERROR_INVALID_VOLUME
: An invalid disk handle was passed to the routine.

ERROR_ACCESS_DENIED
: For some reason, the volume's name could not be changed. For example, the volume might not be writable.

ERROR_DISK_STALE
: The drive containing that disk has been deleted. This usually only happens with network drives.

Include: disk.h

■ DosExec()

```
word      DosExec(
          const char *        prog,
          DiskHandle          progDisk,
          const char *        arguments,
          const char *        execDir,
          DiskHandle          execDisk,
          DosExecFlags        flags);
```

This routine shuts down GEOS to run a DOS program. It returns an error code if an error occurs or zero if successful. Its parameters are listed below:

prog	A pointer to a null-terminated character string representing the path of the program to be run. If a null string (not a null pointer), the system's DOS command interpreter will be run. The path string should not contain the drive name.
progDisk	A disk handle indicating the disk on which the program to be executed sits. If zero is passed, the disk on which GEOS resides will be used.
arguments	A pointer to a locked or fixed buffer containing arguments to be passed to the program being run.
execDir	A pointer to a null-terminated character string representing the path in which the program is to be run. The string should not contain the drive name. If a null pointer is passed and *execDisk* is zero, the program will be run in the directory in which GEOS was first started.
execDisk	The disk handle of the disk containing the directory in *execDir*.
flags	A record of **DosExecFlags** indicating whether the DOS program will give a prompt to the user to return to GEOS. The possible flags are DEF_PROMPT, DEF_FORCED_SHUTDOWN, and DEF_INTERACTIVE. For more information, see the entry for **DosExecFlags**.

If there was no error, DosExec() will return zero. Otherwise it will return one of the following error values: ERROR_FILE_NOT_FOUND, ERROR_DOS_EXEC_IN_PROGRESS, ERROR_INSUFFICIENT_MEMORY, or ERROR_ARGS_TOO_LONG.

Include: system.h

■ DriveGetDefaultMedia()

```
MediaType DriveGetDefaultMedia(
          word                driveNumber);
```

This routine returns the default media type for the specified drive. It returns a member of the **MediaType** enumerated type (described in the Data Structures reference). Note that a drive can be used for media types other than the default. For example, a high-density 3.5-inch drive will have a default media type of MEDIA_1M44, but it can read from, write to, and format 3.5-inch disks with size MEDIA_720K.

See Also: DriveTestMediaSupport()

Include: drive.h

■ DriveGetExtStatus()

```
word      DriveGetExtStatus(
          word                driveNumber);
```

This routine is much like **DriveGetStatus()** (described immediately below). However, in addition to returning all of the flags set by **DriveGetStatus()**, it also sets additional flags in the upper byte of the return value. It returns the following additional flags:

DES_LOCAL_ONLY	This flag is set if the device cannot be viewed over a network.
DES_READ_ONLY	This flag is set if the device is read only, i.e. no data can ever be written to a volume mounted on it (e.g., a CD-ROM drive).
DES_FORMATTABLE	This flag is set if disks can be formatted in the drive.
DES_ALIAS	This flag is set if the drive is actually an alias for a path on another drive.
DES_BUSY	This flag is set if the drive will be busy for an extended period of time (e.g., if a disk is being formatted).

If an error condition exists, **DriveGetExtStatus()** returns zero.

See Also: DriveGetStatus()

Include: drive.h

■ DriveGetName()

```
char *    DriveGetName(
          word   driveNumber,  /* Get name of this drive */
          char * buffer,       /* Write name to this buffer */
          word   bufferSize);  /* Size of buffer (in bytes) */
```

This routine finds the name of a specified drive. You should use this name when prompting the user to take any action regarding this drive (e.g. to insert a disk). The routine writes the name, as a null terminated string, to the buffer passed. It returns a pointer to the trailing null. If the drive does not exist, or the buffer is too small, **DriveGetName()** returns a null pointer.

Include: drive.h

■ DriveGetStatus()

```
word      DriveGetStatus(
          word   driveNumber);
```

This routine returns the current status of a drive. The drive is specified by its drive number. The routine returns a word of **DriveStatus** flags. These flags are listed below:

DS_PRESENT	This flag is set if the physical drive exists, regardless of whether the drive contains a disk.
DS_MEDIA_REMOVABLE	This flag is set if the disk can be removed from the drive.
DS_NETWORK	This flag is set if the drive is accessed over a network (or via network protocols), which means the drive cannot be formatted or copied.
DS_TYPE	This is a mask for the lowest four bits of the field. These bits contain a member of the **DriveType** enumerated type.

If an error condition exists, **DriveGetStatus()** returns zero.

See Also: DriveGetExtStatus()

Include: drive.h

■ DriveTestMediaSupport()

```
Boolean  DriveTestMediaSupport(
         word               DriveNumber,
         MediaType          media);      /* Desired disk size */
```

This routine checks whether the specified drive can support disks in the specified size. It returns *true* (i.e. non-zero) if the drive supports the size.

See Also: DriveGetDefaultMedia()

Include: drive.h

Routines

■ EC()

void `EC( line);`

This macro defines a line of code that will only be compiled into the error-checking version of the geode. The *line* parameter of the macro is the actual line of code. When the EC version of the program is compiled, the line will be treated as a normal line of code; when the non-EC version is compiled, the line will be ignored.

■ EC_BOUNDS()

void `EC_BOUNDS(addr);`

This macro adds an address check to the error-checking version of a program. When the EC version of the program is compiled, the address check will be included; when the non-EC version is compiled, the address check will be left out. The *addr* parameter is the address or pointer to be checked.

The macro expands to a call to **ECCheckBounds()** on the specified address or pointer. If the address is out of bounds, the program will stop with a call to **FatalError()**.

See Also: ECCheckBounds()

■ EC_ERROR()

void `EC_ERROR(code);`

This macro inserts a call to **FatalError()** in the error-checking version of the program and does nothing to the non-EC version. When the program gets to this point, it will halt and put up an error message corresponding to the specified error *code*. If a condition should be checked before calling **FatalError()**, you can use EC_ERROR_IF() instead.

■ EC_ERROR_IF()

void `EC_ERROR_IF(test, code);`

This macro inserts a conditional call to **FatalError()** in the error-checking version of a program; it does nothing for the non-EC version. The *test* parameter is a Boolean value that, if *true*, will cause the **FatalError()** call to be made. If *test* is *false*, **FatalError()** will not be called.

■ EC_WARNING()

```
EC_WARNING(word warningCode);
```

This macro generates a warning for the debugger when executed by error-checking code; it has no effect when in non-EC code.

Include: ec.h

■ EC_WARNING_IF()

```
EC_WARNING_IF(<expr>, word warningCode)
```

When this macro is executed in error-checking code, it tests <*expr*>; if <*expr*> is non-zero, it generates a warning with code *warningCode* for the debugger.

In non-EC code, the macro has no effect (and <*expr*> is not evaluated).

Include: ec.h

■ ECCheckBounds()

```
void    ECCheckBounds(
        void   *address);
```

This routine checks to see if the given pointer is within bounds of the block into which it points. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckChunkArray()

```
void    ECCheckChunkArray(
        optr   o);
```

This routine checks the validity of the specified chunk array. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckChunkArrayHandles()

```
void    ECCheckChunkArrayHandles(
        MemHandle mh,
        ChunkHandle ch);
```

This routine checks the validity of the specified chunk array. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckClass()

```
void        ECCheckClass(
            ClassStruct *class);
```

This routine checks that the given pointer actually references a class definition. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckDriverHandle()

```
void        ECCheckDriverHandle(
            GeodeHandle gh);
```

This routine checks that the passed handle actually references a driver. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckEventHandle()

```
void        ECCheckEventHandle(
            EventHandle eh);
```

This routine checks that the passed handle actually references a stored message. If the assertions fail, a fatal error will occur.

■ ECCheckFileHandle()

```
void        ECCheckFileHandle(
            FileHandle file);
```

This routine checks that the passed handle actually is a file handle and references a file. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckGeodeHandle()

```
void        ECCheckGeodeHandle(
            GeodeHandle gh);
```

This routine checks that the passed handle references a loaded geode. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckGStateHandle()

```
void      ECCheckGStateHandle(
          GStateHandle gsh);
```

This routine checks that the passed handle references a GState. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckHugeArray()

```
void      ECCheckHugeArray(
          VMFileHandle        vmFile,
          VMBlockHandle       vmBlock);
```

This routine checks the validity of the passed Huge Array. If the block passed is not the directory block of a Huge Array, the routine fails.

Include: ec.h

■ ECCheckLibraryHandle()

```
void      ECCheckLibraryHandle(
          GeodeHandle gh);
```

This routine checks that the passed handle references a library. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemChunk()

```
void      ECCheckLMemChunk(
          void * chunkPtr);
```

This routine checks the validity of the chunk pointed to by *chunkPtr*. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemHandle()

```
void      ECCheckLMemHandle(
          MemHandle mh);
```

This routine checks that the passed handle is a memory handle and actually references a local memory block. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemHandleNS()

```
void        ECCheckLMemHandleNS(
            MemHandle mh);
```

This routine checks that the passed handle is a local memory handle; unlike **ECCheckLMemHandle()**, however, it does not check sharing violations (when threads are illegally using non-sharable memory). If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemObject()

```
void        ECCheckLMemObject(
            optr   obj);
```

This routine checks the validity of an object to ensure that it is an object stored in an object block. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemObjectHandles()

```
void        ECCheckLMemObjectHandles(
            MemHandle mh,
            ChunkHandle ch);
```

This routine checks the validity of an object to ensure that it is an object stored in an object block. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemOD()

```
void        ECCheckLMemOD(
            optr   o);
```

This routine checks the validity of the given local-memory-based object. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckLMemODHandles()

```
void        ECCheckLMemODHandles(
            MemHandle objHan,
            ChunkHandle objCh);
```

This routine checks the validity of the given local-memory-based object. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckMemHandle()

```
void        ECCheckMemHandle(
            MemHandle mh);
```

This routine checks that the passed handle is a memory handle that references a memory block. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckMemHandleNS()

```
void        ECCheckMemHandleNS(
            MemHandle mh);
```

This routine checks that the passed handle references a memory block; unlike **ECCheckMemHandle()**, however, it will not check for sharing violations (when a thread illegally accesses a non-sharable block). If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckObject()

```
void        ECCheckObject(
            optr   obj);
```

This routine checks the validity of the given locked object. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckObjectHandles()

```
void        ECCheckObjectHandles(
            Memhandle mh,
            ChunkHandle ch);
```

This routine checks the validity of the given locked object. If the assertions fail, a fatal error will occur.

■ ECCheckOD()

```
void        ECCheckOD(
            optr   obj);
```

This routine checks the validity of the given object. Unlike **ECCheckLMemObject()**, however, it allows optrs of Process objects to be specified. If assertions fail, a fatal error will occur.

■ ECCheckODHandles()

```
void        ECCheckODHandles(
            MemHandle objHan,
            ChunkHandle objCh);
```

This routine checks the validity of the given object. Unlike **ECCheckLMemObjectHandles()**, however, it allows processes to be specified. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckProcessHandle()

```
void        ECCheckProcessHandle(
            GeodeHandle gh);
```

This routine checks that the passed handle actually references a process. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckQueueHandle()

```
void        ECCheckQueueHandle(
            QueueHandle qh);
```

This routine ensures the passed handle references an event queue. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckResourceHandle()

```
void        ECCheckResourceHandle(
            MemHandle mh);
```

This routine ensures that the passed handle references a geode resource. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckStack()

```
void        ECCheckStack();
```

This routine checks to make sure the current stack has not overflown (and is not about to). This routine also enforces a 100-byte gap between the stack bottom and the stack pointer. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckThreadHandle()

```
void      ECCheckThreadHandle(
          ThreadHandle th);
```

This routine checks that the passed handle actually references a thread. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECCheckWindowHandle()

```
void      ECCheckWindowHandle(
          WindowHandle wh);
```

This routine checks that the passed handle actually references a window. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECLMemExists()

```
void      ECLMemExists(
          optr   o);
```

This routine checks to see if the specified chunk exists. This routine should be called by applications to check the chunk handle's validity. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECLMemExistsHandles()

```
void      ECLMemExistsHandles(
          MemHandle mh,
          ChunkHandle ch);
```

This routine checks to see if the specified chunk exists. This routine should be called by applications to check the chunk handle's validity. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECLMemValidateHandle()

```
void      ECLMemValidateHandle(
          optr   o);
```

This routine checks that the passed optr points to a local memory chunk. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECLMemValidateHandleHandles()

```
void    ECLMemValidateHandleHandles(
        MemHandle mh,
        ChunkHandle ch);
```

This routine checks that the passed memory and chunk handles actually reference a local memory chunk. If the assertions fail, a fatal error will occur.

Include: ec.h

■ ECLMemValidateHeap()

```
void    ECLMemValidateHeap(
        MemHandle mh);
```

This routine does a complete error-check of the LMem heap. It is used internally and should not be needed by application programmers.

Include: ec.h

■ ECMemVerifyHeap()

```
void    ECMemVerifyHeap()
```

This routine makes sure the global heap is in a consistent state. If the assertions fail, a fatal error will occur. This routine should likely not be called by anything other than the EC kernel.

Include: ec.h

■ ECVMCheckMemHandle()

```
void    ECVMCheckMemHandle(
        MemHandle han);
```

This routine checks that the given memory handle is actually linked to a VM block handle. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECVMCheckVMBlockHandle()

```
void    ECVMCheckVMBlockHandle(
        VMFileHandle file,
        VMBlockHandle block);
```

This routine checks the validity of the given VM file and block handles. If assertions fail, a fatal error will occur.

Include: ec.h

■ ECVMCheckVMFile()

```
void      ECVMCheckVMFile(
          VMFileHandle file);
```

This routine checks the validity of the given VM file handle. If assertions fail, a fatal error will occur.

Include: ec.h

■ ElementArrayAddElement ()

```
word      ElementArrayAddElement(
          optr   arr,            /* Handle of element array */
          void * element,        /* Element to add (if necessary) */
          dword  callBackData,   /* This is passed to the Callback routine */
          Boolean _pascal (*callback) (void *elementToAdd,
                                void *elementFromArray, dword valueForCallback));
```

This routine is used to add elements to an array. It is passed the address of a potential element. It compares the element with each member of an element array. If there are no matches, it adds the element to the array and sets the reference count to one. If there is a match, it increments the reference count of the matching element in the array and returns; it does not add the new element. When you pass the address of an element, make sure you pass the address of the data portion of the element (not the reference-count header).

You can pass a callback routine to **ElementArrayAddElement()**. **ElementArrayAddElement()** will call the callback routine to compare elements and see if they match. The callback routine should be declared _pascal. **ElementArrayAddElement()** passes the callback routine the address of the element you passed it, as well as the address of the data-portion of the element in the array (the part after the **RefElementHeader** structure). If the two elements match (by whatever criteria you use), return *true*; otherwise, return *false*. If you pass a null function pointer, the default comparison routine will be called, which checks to see if every data byte matches.

Include: chunkarr.h

Tips and Tricks: If you know the element is already in the array, you can increment its reference count by calling **ElementArrayAddReference()**.

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayAddReference()

■ ElementArrayAddElementHandles()

```
word      ElementArrayAddElementHandles(
          MemHandle          mh,           /* Global handle of LMem heap */
          ChunkHandle        chunk         /* Chunk handle of element array */
          void *             element,      /* Element to add */
          dword              callBackData,/* Passed to the Callback routine */
          Boolean _pascal (*callback) (void *elementToAdd,
                             void *elementFromArray, dword valueForCallback));
```

This routine is exactly like **ElementArrayAddElement()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

Tips and Tricks: If you know the element is already in the array, you can increment its reference count by calling **ElementArrayAddReference()**.

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayAddReference()

■ ElementArrayAddReference()

```
void      ElementArrayAddReference(
          optr   arr,                    /* optr to element array */
          word   token);                 /* Index number of element */
```

This routine increments the reference count of a member of an element array.

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayAddElement()

■ ElementArrayAddReferenceHandles()

```
void      ElementArrayAddReferenceHandles(
          MemHandle          mh,           /* Handle of LMem heap's block */
          ChunkHandle        ch,           /* Handle of element array */
          word               token);       /* Index number of element */
```

This routine is exactly like **ElementArrayAddReference()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ElementArrayCreate()

```
ChunkHandle ElementArrayCreate(
          MemHandle          mh,           /* Handle of LMem heap’s block */
          word               elementSize,  /* Size of each element, or zero
                                            * for variable-sized */
          word               headerSize);  /* Header size (zero for default) */
```

This routine creates an element array in the indicated LMem heap. It creates an **ElementArrayHeader** structure at the head of the chunk. If you want to leave extra space before the start of the array, you can pass a larger header size; if you want to use the standard header, pass a header size of zero.

You can specify the size of each element. Remember that the first three bytes of every element in an element array are the element’s **RefElementHeader**; structure, which contains the reference count; leave room for this when you choose a size. For arrays with variable-sized elements, pass a size of zero.

Include: chunkarr.h

Tips and Tricks: You may want to declare a structure for array elements; the first component should be a **RefElementHeader**. You can pass the size of this structure to **ElementArrayCreate()**.

If you want extra space after the **ElementArrayHeader**, you may want to create your own header structure, the first element of which is an **ElementArrayHeader**. You can pass the size of this header to **ElementArrayCreate()**, and access the data in your header via the structure.

Be Sure To: Lock the block on the global heap before calling this routine (unless it is fixed). If you pass a header size, make sure it is larger than **sizeof(ElementArrayHeader)**.

■ ElementArrayCreateAt

```
ChunkHandle ElementArrayCreateAt(
          optr   arr,                   /* optr of chunk for array */
          word   elementSize,           /* Size of each element, or zero
                                         * for variable-sized */
          word   headerSize);           /* Header size (zero for default) */
```

This routine is just like **ElementArrayCreate()** above, except that the element array is created in a pre-existing chunk. The contents of that chunk will be overwritten.

Include: chunkarr.h

Warnings: If the chunk isn't large enough, it will be resized. This will invalidate all pointers to chunks in that block.

■ ElementArrayCreateAtHandles

```
ChunkHandle ElementArrayCreateAtHandles(
          MemHandle          mh,           /* Handle of LMem heap */
          ChunkHandle        ch            /* Create array in this chunk */
          word               elementSize,  /* Size of each element, or zero
                                          * for variable-sized */
          word               headerSize);  /* Header size (zero for default) */
```

This routine is exactly like **ElementArrayCreateAt()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

Warnings: If the chunk isn't large enough, it will be resized. This will invalidate all pointers to chunks in that block.

■ ElementArrayDelete()

```
void      ElementArrayDelete(
          optr   arr,                      /* optr to element array */
          word   token);                   /* index of element to delete */
```

This routine deletes an element from an element array regardless of its reference count. The routine is passed the element array's optr and the token for the element to delete.

Note that when an element is removed, it is actually resized down to zero size and added to a list of free elements. That way the index numbers of later elements are preserved.

Include: chunkarr.h

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayRemoveReference()

■ ElementArrayDeleteHandles()

```
void      ElementArrayDeleteHandles(
          MemHandle          mh,           /* Handle of LMem heap */
          ChunkHandle        ch,           /* Chunk handle of element array */
          word               token);       /* Index of element delete */
```

This routine is exactly like **ElementArrayDelete()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayRemoveReference()

■ ElementArrayElementChanged()

```
void      ElementArrayElementChanged(
          optr   arr,                    /* optr to element array */
          word   token,                  /* Index number of element */
          dword  callbackData,           /* This is passed along to callback */
          Boolean _pascal (*callback)/* Returns true if elements identical */
                          (void *   elementChanged,
                           void *   elementToCompare,
                           dword    valueForCallback));
```

This routine checks to see if an element is identical to any other elements in the same element array. This is used after an element has changed to see if it now matches another element. If the element matches another, it will be deleted, and the other element will have its reference count incremented.

The routine is passed an optr to the element array, the token of the element which is being checked, a dword of data (which is passed to the callback routine), and a pointer to a callback comparison routine. The callback routine itself is passed pointers to two elements and the *callbackData* argument passed to **ElementArrayElementChanged()**. The callback routine should be declared _pascal. If the two elements are identical, the callback should return *true* (i.e. non-zero); otherwise, it should return *false*.

If you pass a null function pointer, **ElementArrayElementChanged()** will do a bytewise comparison of the elements.

Include: chunkarr.h

■ ElementArrayElementChangedHandles()

```
void      ElementArrayElementChangedHandles(
          MemHandle           memHandle,    /* Handle of LMem heap’s block */
          ChunkHandle         chunkHandle,  /* Chunk handle of element array */
          word                token,        /* Index number of element */
          dword               callbackData,    /* This is passed along to
                                             * callback */
          Boolean _pascal (*callback)/* Returns true if elements identical */
                              (void *   elementChanged,
                               void *   elementToCompare,
                               dword    valueForCallback));
```

This routine is exactly like **ElementArrayElementChanged()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ElementArrayGetUsedCount()

```
word      ElementArrayGetUsedCount(
          optr   arr,                 /* optr to element array */
          dword  callbackData,        /* This is passed to callback routine */
          Boolean _pascal (*callback)/* return true to count this element */
                              (void * element, dword cbData));
```

This routine counts the number of active elements in an element array; that is, elements which have a reference count of one or greater. It can be instructed to count every element, or every element which matches certain criteria. The routine is passed three parameters: the optr of the chunk array, a dword which is passed to the callback routine, and a callback routine which determines whether the element should be counted. The callback routine,which should be declared _pascal, is passed the dword an a pointer to an element. It should return *true* if the element should be counted; otherwise, it should return *false*. To count every element, pass a null callback pointer.

Include: chunkarr.h

See Also: ElementArrayTokenToUsedIndex(), ElementArrayUsedIndexToToken()

■ ElementArrayGetUsedCountHandles()

```
void      ElementArrayGetUsedCountHandles(
          MemHandle          mh,       /* Handle of LMem heap’s block */
          ChunkHandle        ch,       /* Chunk handle of element array */
          dword  callbackData,       /* This is passed to callback routine */
          Boolean _pascal (*callback)/* return true to count this element */
                             (void * element, dword cbData));
```

This routine is exactly like **ElementArrayGetUsedCount()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ElementArrayRemoveReference()

```
void      ElementArrayRemoveReference(
          optr               arr,         /* optr of element array */
          word               token,       /* Index of element to
                                        * unreference */
          dword              callbackData,/* Passed to callback routine */
          void _pascal (*callback)(void *element, dword valueForCallback));
                 /* Routine is called if element is actually removed */
```

This routine decrements the reference count of the specified element. If the reference count drops to zero, the element will be removed. If an element is to be removed, **ElementArrayRemoveReference()** calls the callback routine on that element. The callback routine should perform any cleanup necessary; it is passed a pointer to the element and the *callbackData* argument. If you pass a null function pointer, no callback routine will be called.

Note that when an element is removed, it is actually resized down to zero size and added to a list of free elements. That way the index numbers of later elements are preserved.

Be Sure To: Lock the block on the global heap before calling (unless it is fixed).

See Also: ElementArrayDelete()

Include: chunkarr.h

■ ElementArrayRemoveReferenceHandles()

```
void    ElementArrayRemoveReferenceHandles(
        MemHandle         mh,       /* Handle of LMem heap */
        ChunkHandle       ch,       /* Chunk handle of element array */
        word              token,    /* Index of element to unreference */
        dword             callbackData,/* Passed to callback routine */
        void _pascal (*callback)(void *element, dword valueForCallback));
              /* Routine is called if element is actually removed */
```

This routine is exactly like **ElementArrayRemoveReference()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ElementArrayTokenToUsedIndex()

```
word    ElementArrayTokenToUsedIndex(
        optr   arr,          /* Handle of element array */
        word   token,        /* Index of element to unreference */
        dword  callbackData,/* Data passed to callback routine */
        Boolean _pascal (*callback)/* Return true to count this element */
                  (void *element, dword cbData));
```

This routine is passed the token of an element array. It translates the token into an index from some non-standard indexing scheme. The indexing scheme can either number the elements from zero, counting only those elements in use (i.e. those with a reference count greater than zero); or it can use a more restrictive scheme. If a callback routine is passed, the callback routine will be called for every used element; it should be declared _pascal and return *true* if the element should be counted. If a null callback pointer is passed, every used element will be counted.

Include: chunkarr.h

■ ElementArrayTokenToUsedIndexHandles()

```
word      ElementArrayTokenToUsedIndexHandles(
          MemHandle           mh,    /* Handle of LMem heap */
          ChunkHandle         ch,    /* Chunk handle of element array */
          word                token, /* Index of element to unreference */
          dword               callbackData, /* Data passed to the
                                         * callback routine */
          Boolean _pascal (*callback)/* Return true to count this element */
                              (void *element, dword cbData));
```

This routine is exactly like **ElementArrayTokenToUsedIndex()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ ElementArrayUsedIndexToToken()

```
word      ElementArrayUsedIndexToToken(
          optr   arr,            /* optr to element array */
          word   index,          /* Find token of element with this index */
          dword  callbackData,   /* This is passed to the callback routine */
          Boolean _pascal (*callback)/* Return true to count this element */
                              (void *element, dword cbData));
```

This routine takes an index into an element array from some non-standard indexing scheme. The routine finds the element specified and returns the element's token. The indexing scheme can either number the elements from zero, counting only those elements in use (i.e. those with a reference count greater than zero); or it can use a more restrictive scheme. If a callback routine is passed, the callback routine will be called for every used element; it should should be declared _pascal return *true* if the element should be counted. If a null callback pointer is passed, every used element will be counted.

If no matching element is found, **ElementArrayUsedIndexToToken()** returns CA_NULL_ELEMENT.

Include: chunkarr.h

■ ElementArrayUsedIndexToTokenHandles()

```
word     ElementArrayUsedIndexToTokenHandles(
         MemHandle          mh,    /* Handle of LMem heap's block */
         ChunkHandle        ch,    /* Handle of element array */
         word               index, /* Find token of element with this index */
         dword              callbackData, /* Data passed to the
                                               * callback routine */
         Boolean _pascal (*callback)/* Return true to count this element */
                    (void *element, dword cbData));
```

This routine is exactly like **ElementArrayUsedIndexToToken()** above, except that the element array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ EvalExpression()

```
int      EvalExpression(
         byte   * tokenBuffer,      /* Pointer to the parsed expression */
         byte   * scratchBuffer,    /* Pointer to the base of a scratch buffer
                                     * consisting of two stacks: an argument
                                     * stack and an operator/function stack */
         byte   * resultsBuffer,    /* Pointer to a buffer to contain the
                                     * result of the evaluation */
         word   bufSize,            /* Size of the scratch buffer */
         CEvalStruct * evalParams); /* Pointer to CEvalStruct structure */
```

This routine evaluates a stream of parser tokens. It is used by the evaluator portion of the parse library and will be used only rarely by applications.

Include: parse.h

■ FatalError()

```
void     FatalError(
         word errorCode);
```

This routine causes a fatal error, leaving *errorCode* for the debugger.

■ FileClose()

```
word     FileClose( /* returns error */
         FileHandle         fh,             /* File to close */
         Boolean            noErrorFlag);   /* Set if app. can't handle
                                             * errors */
```

This routine closes an open byte file. If the routine succeeds, it returns zero. If the routine fails and *noErrorFlag* is *false* (i.e., zero), **FileClose()** returns a member of the **FileError** enumerated type. If the routine fails and *noErrorFlag* is *true* (i.e., non-zero), the routine will fatal-error.

Warnings: The *noErrorFlag* parameter should be *true* only during debugging.

Include: file.h

■ FileCommit()

```
word     FileCommit( /* returns error */
         FileHandle          fh,
         Boolean             noErrorFlag); /* set if can't handle errors */
```

FileCommit() forces the file system to write any cached information about a file to the disk immediately. If it is successful, it returns zero. If it fails, it returns an error code. If the routine fails and *noErrorFlag* is *true* (i.e. non-zero), the routine will fatal-error.

Warnings: The *noErrorFlag* parameter should be *true* only during debugging.

Include: file.h

■ FileConstructFullPath()

```
DiskHandle FileConstructFullPath(
         char                * * buffer,   /* Path string is written here */
         word                bufSize,  /* Length of buffer (in bytes) */
         DiskHandle          disk,     /* Disk or standard path; null for
                                        * current path */
         const char          * tail,   /* Path relative to handle */
         Boolean             addDriveLetter); /* Should path begin with drive
                                                 * name? */
```

This routine translates a GEOS directory specification into a complete path string. It writes the string into the passed buffer. The directory is specified by two arguments: The first, *disk*, is the handle of a disk; this may also be a standard path constant. (If a null handle is passed, the current working directory is used.) The second, *tail*, is a pointer to the character string representing the tail end of the path. **FileConstructFullPath()** appends this relative path to the location indicated by the disk handle. It then constructs a full path string, beginning with that disk's root directory, and writes it to the buffer passed. If *addDriveName* is *true* (i.e. non-zero), the path string will begin with the drive's name and a colon.

Examples: The following call to **FileConstructFullPath()** might yield these results:

Code Display 6-1 Sample call to FileConstructFullPath()

```
/* Here we find out the full path of a subdirectory of the DOCUMENT directory */

        DiskHandle      documentDisk;
        char            pathBuffer[256];        /* long enough for most paths */

        documentDisk = FileConstructFullPath(&pathBuffer,/* pointer to pointer */
                                        256,            /* Length of buffer */
                                        SP_DOCUMENT,    /* This can be a disk or
                                                         * standard path */
                                        "MEMOS\\JANUARY", /* In C strings, the
                                                         * backslash must be
                                                         * doubled */
                                        TRUE);          /* Prepend drive name */

/* If the standard paths are set up in the default configuration, “documentDisk”
 * would be the handle of the main hard drive, and pathBuffer would contain a
 * string like “C:\GEOWORKS\DOCUMENT\MEMOS\JANUARY” */
```

See Also: FileParseStandardPath()

Include: file.h

■ FileCopy()

```
word    FileCopy( /* returns error */
        const char          * source,     /* Source path and file name */
        const char          * dest,       /* Destination path and file name */
        DiskHandle          sourceDisk,   /* These handles may be Standard */
        DiskHandle          destDisk);    /* Path constants, or null to indi-
                                       * cate current working directory */
```

This routine makes a copy of a file. The source and destination are specified with path strings. Each string specifies a path relative to the location specified by the corresponding disk handle. If the handle is a disk handle, the path is relative to that disk’s root. If the disk handle is a standard path constant, the path string is relative to that standard path. If the disk handle is null, the path is relative to the current working directory.

If **FileCopy()** is successful, it returns zero. Otherwise, it returns one of the following error codes:

ERROR_FILE_NOT_FOUND
No such source file exists in the specified directory.

ERROR_PATH_NOT_FOUND
An invalid source or destination path string was passed.

ERROR_ACCESS_DENIED
: You do not have permission to delete the existing copy of the destination file, or the destination disk or directory is not writable.

ERROR_FILE_IN_USE
: Some geode has the existing destination file open.

ERROR_SHORT_READ_WRITE
: There was not enough room on the destination disk.

See Also: FileMove()

Include: file.h

■ FileCreate()

```
FileHandle FileCreate( /* sets thread's error value */
          const char          * name,      /* relative to working directory */
          FileCreateFlags     flags,       /* see below */
          FileAttrs           attributes); /* FileAttrs of new file */
```

This routine creates a byte file. The file may be a DOS file or a GEOS byte file. If the file is successfully opened, **FileCreate()** will return the file's handle; otherwise, it will return a null handle and set the thread's error value.

The second parameter is a word-length **FileCreateFlags** record. The lower byte of this field is a **FileAccessFlags** record. This specifies the file's permissions and exclusions. Note that you must request write or read/write permission when you create a file. The upper byte specifies how the file should be created. It contains the following possible values:

FILE_CREATE_TRUNCATE
: If a file with the given name exists, it should be opened and truncated; that is, all data should be deleted.

FILE_CREATE_NO_TRUNCATE
: If the file exists, it should be opened without being truncated.

FILE_CREATE_ONLY
: If the file exists, the routine should fail and set the thread's error value to ERROR_FILE_EXISTS.

FCF_NATIVE
: This flag is combined with one of the above flags if the file should be created in the device's native format; e.g. if it should be a DOS file instead of a GEOS file. The name passed must be an acceptable native file name. If a GEOS file with the specified name already exists, **FileCreate()** will fail with error

condition ERROR_FILE_FORMAT_MISMATCH. Similarly, if the flag isn't set and a non-GEOS file with this name exists, **FileCreate()** will fail and return this error.

The third parameter, *attributes*, describes the **FileAttrs** record to be set for the new file.

If successful, **FileCreate()** returns the file's handle. If it is unsuccessful, it returns a null handle and sets the thread's error value. The following error values are commonly returned:

ERROR_PATH_NOT_FOUND
A relative or absolute path was passed, and the path included a directory which did not exist.

ERROR_TOO_MANY_OPEN_FILES
There is a limit to how many files may be open at once. If this limit is reached, **FileCreate()** will fail until a file is closed.

ERROR_ACCESS_DENIED
Either the caller requested access which could not be granted (e.g. it requested write access when another geode had already opened the file with FILE_DENY_W), or the caller tried to deny access when that access had already been granted to another geode (e.g. it tried to open the file with FILE_DENY_W when another geode already had it open for write-access).

ERROR_WRITE_PROTECTED
The caller requested write or read-write access to a file in a write-protected volume.

ERROR_FILE_EXISTS
Returned if **FileCreate()** was called with FILE_CREATE_ONLY and a file with the specified name already exists.

ERROR_FILE_FORMAT_MISMATCH
Returned if **FileCreate()** was called with FILE_CREATE_TRUNCATE or FILE_CREATE_NO_TRUNCATE and a file exists in a different format than desired; i.e. you passed FCF_NATIVE and the file already exists in the GEOS format, or vice versa.

Examples: An example of usage is shown below.

Code Display 6-2 Example of FileCreate() usage

```
/* Here we create a DOS file in the current working directory. If the file already
 * exists, we open the existing file and truncate it.
 */

        FileHandle      newFile;

        newFile =       FileCreate("NEWFILE.TXT",
                                        ( (FILE_CREATE_TRUNCATE | FCF_NATIVE)
                                         | (FILE_ACCESS_RW | FILE_DENY_RW)),
                                        0); /* set no attribute bits */
```

See Also: FileCreateTempFile(), FileOpen()

Include: file.h

■ FileCreateDir()

```
word     FileCreateDir( /* Returns error & sets thread’s error value */
         const char * name);       /* Relative path of new directory */
```

This routine creates a new directory. The parameter is a path string; the path is relative to the current directory. The last element of the path string must be the directory to create.

If **FileCreateDir()** is successful, it returns zero and clears the thread’s error value. Otherwise, it returns an error code and sets the thread’s error value. The following errors are returned:

ERROR_PATH_NOT_FOUND
: The path string was in some way invalid; for example, it might have instructed **FileCreateDir()** to create the directory within a directory which does not exist.

ERROR_ACCESS_DENIED
: The thread is not able to create directories in the specified location, or a directory with the specified name already exists.

ERROR_WRITE_PROTECTED
: The volume is write-protected.

See Also: FileDeleteDir()

Include: file.h

■ FileCreateTempFile()

```
FileHandle FileCreateTempFile( /* Sets thread’s error value */
          char              * dir, /* directory, relative to working dir.;
                                    * file name replaces 14 trailing null
                                    * characters upon return */
          FileAttrs          attributes);
```

This routine creates and opens a temporary file in the directory specified. The routine automatically selects a name for the temporary file. No creation flags are needed, since the file will definitely be created anew and will be used only by this geode. The directory string must end with fourteen null bytes (enough to be replaced by the new file’s name).

If **FileCreateTempFile()** is successful, it returns the file’s handle as well as the string passed in *dir*, with the trailing null characters replaced by the file name. If it is unsuccessful, it returns a null handle and sets the thread’s error value to a member of the **FileError** enumerated type.

Tips and Tricks: Temporary files are usually created in a subdirectory of SP_PRIVATE_DATA.

See Also: FileCreate()

Include: file.h

■ FileDelete()

```
word      FileDelete( /* returns error */
          const char * name);    /* path relative to working directory */
```

This routine deletes a file. If it is successful, it returns zero; otherwise, it returns a **FileError**. Common errors include:

ERROR_FILE_NOT_FOUND
No such file exists in the specified directory.

ERROR_WRITE_PROTECTED
The volume is write-protected.

ERROR_PATH_NOT_FOUND
An invalid path string was passed.

ERROR_ACCESS_DENIED
You do not have permission to delete that file.

ERROR_FILE_IN_USE
Some geode has that file open.

Include: file.h

■ FileDeleteDir()

```
word     FileDeleteDir( /* Returns error & sets thread’s error value */
         const char * name);    /* Relative path of directory to delete */
```

This argument deletes an existing directory. The parameter is a string which specifies the directory’s position relative to the current working directory. The last element of the path string must be the name of the directory to delete.

If **FileDeleteDir()** is successful, it returns zero and clears the thread’s error value. Otherwise, it returns an error code and sets the thread’s error value. The following errors are returned:

ERROR_PATH_NOT_FOUND	The directory specified could not be found or does not exist.
ERROR_IS_CURRENT_DIRECTORY	This directory is some thread’s current directory, or else it is on some thread’s directory stack.
ERROR_ACCESS_DENIED	The thread does not have permission to delete the directory.
ERROR_WRITE_PROTECTED	The volume is write-protected.
ERROR_DIRECTORY_NOT_EMPTY	The directory specified is not empty. A directory must be empty before it can be deleted.

See Also: FileCreateDir()

Include: file.h

■ FileDuplicateHandle()

```
FileHandle FileDuplicateHandle( /* Sets thread’s error value */
         FileHandle fh);
```

This routine duplicates the handle of an open file and returns the duplicate handle. The duplicate handle has the same read/write position as the original. Both handles will have to be closed for the file to be closed. If there is an error, **FileDuplicateHandle()** returns a null handle and sets the thread’s error value.

Include: file.h

■ FileEnum()

```
word      FileEnum( /* returns number of files returned */
          FileEnumParams      * params, /* described below */
          MemHandle           * bufCreated,/* FileEnum will allocate a return-
                                           * buffer block & write its handle
                                           * here */
          word                * numNoFit); /* Number of files not handled is
                                           * written here */
```

This routine is used to examine all the files in a directory. The routine can filter the files by whether they have certain extended attributes. It creates a buffer and writes information about the files in this buffer. This routine can be called in many different ways; for full details, see the section “FileEnum()” on page 617 of the Concepts book.

Structures: **FileEnum()** uses several structures and enumerated types. They are shown below; the detailed description of the structures follows.

```
       /* Types, values, and structures passed
        * to the FileEnum() routine: */
typedef enum /* word */ {
       FESRT_COUNT_ONLY,
       FESRT_DOS_INFO,
       FESRT_NAME,
       FESRT_NAME_AND_ATTR
} FileEnumStandardReturnType;
typedef enum /* word */ {
       FESC_WILDCARD
} FileEnumStandardCallback;
       /* Types, values, and structures returned
        * by the FileEnum() routine: */
typedef struct {
       FileAttrs    DFIS_attributes;
       FileDateAndTime DFIS_modTimeDate;
       dword        DFIS_fileSize;
       FileLongName DFIS_name;
       DirPathInfo  DFIS_pathInfo;
} FEDosInfo;
typedef struct _FileEnumCallbackData {
       FileExtAttrDesc FECD_attrs[1];
} FileEnumCallbackData;
typedef struct _FileEnumParams {
       FileEnumSearchFlags FEP_searchFlags;
       FileExtAttrDesc *FEP_returnAttrs;
       word             FEP_returnSize;
       FileExtAttrDesc *FEP_matchAttrs;
       word             FEP_bufSize;
       word             FEP_skipCount;
       word _pascal (*FEP_callback) (struct _FileEnumParams *params,
                              FileEnumCallbackData *fecd,
                              word frame);
      FileExtAttrDesc *FEP_callbackAttrs;
      dword        FEP_cbData1;
      dword        FEP_cbData2;
      word         FEP_headerSize;
} FileEnumParams;
```

Most of the information passed to **FileEnum()** is contained in a **FileEnumParameters** structure. The fields of the structure are as follows:

FEP_searchFlags
This is a byte-length flag field. The flags are of type **FileEnumSearchFlags** (described below). These flags specify which files at the current location will be examined by **FileEnum()**. They also specify such things as whether a callback routine should be used.

FEP_returnAttrs
This is a pointer to an array of **FileExtAttrDesc** structures. The last structure should have its *FEA_attr* field set to FEA_END_OF_LIST. The array specifies what information will be returned by **FileEnum()**. The **FileExtAttrDesc** structure is used in a slightly different way than usual. Every file will have an entry in the return buffer; this entry will contain all the extended attribute information requested. Each **FileExtAttrDesc** structure will specify where in that entry its information should be written. The *FEAD_value* field should contain only an offset value; the extended attribute will be written at that offset into the entry. (You can specify an offset by casting an integer value to type **void ***.) The *FEAD_size* value specifies how long the return value can be. You can also request certain return values by setting *FEP_returnAttrs* to equal a member of the **FileEnumStandardReturnType** (again, by casting the **FileEnumStandardReturnType** value to type **void ***). The **FileEnumStandardReturnType** enumerated type is described later in this section.

FEP_returnSize
This is the size of each entry in the returned buffer. If a standard return type or an array of **FileExtAttrDesc** structures was passed, each entry in the returned buffer will contain all the extended attribute information requested for that file.

FEP_matchAttrs
This is a pointer to an array of **FileExtAttrDesc** structures. The last structure should have its *FEA_attr* field set to

FEA_END_OF_LIST. **FileEnum()** will automatically filter out and ignore all files whose attributes do not match the ones specified by this array. For attributes that are word-sized records, *FEAD_value.offset* holds the bits that must be set, and *FEAD_value.segment* holds the bits that must be clear. For byte-sized flags, *FEAD_value.offset.low* contains the flags that must be set, and *FEAD_value.offset.high* contains flags that must be clear. Byte- and word-sized non-flag values are stored in *FEAD_value.offset*. For all other values, *FEAD_value* holds a pointer to the exact value to match, and *FEAD_size* specifies the length of that value (in bytes). If you do not want to filter out any files in the working directory, or if you will use the callback routine to filter the files, pass a null pointer in this field.

FEP_bufsize

This specifies the maximum number of entries to be returned in the buffer. If you do not want to set a limit, pass the constant FEP_BUFSIZE_UNLIMITED. The buffer will be grown as necessary.

FEP_skipCount

This contains the number of matching files to be ignored before the first one is processed. It is often used in conjunction with *FEP_bufSize* to examine many files a few at a time. For example, if you only wanted to examine ten files at a time, you would set *FEP_bufSize* to ten and *FEP_skipCount* to zero. **FileEnum()** would return the data for the first ten files which match the search criteria. After processing the returned data, if there were any files left over, you could call **FileEnum()** again, this time with *FEP_skipCount* set to ten; **FileEnum()** would handle the next ten matching files and return the data about them. In this way you could walk through all the matching files in the directory. Note that if the **FileEnumSearchFlags** bit FESF_REAL_SKIP is set (in *FEP_searchFlags*), the first files in the directory will be skipped *before* they are tested to see if they match. This is faster, since the match condition won't have to be checked for the first files in the directory.

FEP_callback

This holds a pointer to a Boolean callback routine. The callback routine can check to see if the file matches some other arbitrary criteria. The callback routine is called for any files which match all the above criteria. It should be declared _pascal. It is passed three arguments: a pointer to the **FileEnumParams** structure, a pointer to the current stack frame (which is used by some assembly callback routines), and a pointer to an array of **FileExtAttrDesc** structures. These structures are all the attributes required either for return, matching, or callback (see *FEP_callbackAttrs* below), with the information for the current file filled in; you can search through them directly for the information you want, or you can call **FileEnumLocateAttr()** to search through this array. If the file should be accepted by **FileEnum()**, the callback should return *true*; otherwise it should return *false*. You can also instruct **FileEnum()** to use one of the standard callback routines by passing a member of the **FileEnumStandardCallback** enumerated type. In this case, *FEP_callbackAttrs* is ignored; **FileEnum()** will automatically pass the appropriate information to the callback routine. (Note that if the FESF_CALLBACK bit of the *FEP_searchFlags* field is not set, the *FEP_callback* field is ignored.)

FEP_callbackAttrs
This is a pointer to an array of **FileExtAttrDesc** structures. The last structure should have its *FEA_attr* field set to FEA_END_OF_LIST. The array will be filled in with the appropriate information for each file before the callback routine is called. Note that if the FESF_CALLBACK bit of the *FEP_searchFlags* is not set, the *FEP_callbackAttrs* is ignored. If you do not need any attributes passed to the callback routine, set this field to be a null pointer.

FEP_cbData1, *FEP_cbData2*
These are dword-length fields. Their contents are ignored by **FileEnum()**; they are used to pass information to the callback routine. If you do not call a standard callback routine, you may use these fields any way you wish.

FEP_headerSize
If the flag FESF_LEAVE_HEADER is set, **FileEnum()** will leave an empty header space at the beginning of the return buffer. The size of the header is specified by this field. If FESF_LEAVE_HEADER is clear, this field is ignored.

The first field of the **FileEnumParams** structure, *FEP_searchFlags*, is a word-length record containing **FileEnumSearchFlags**. The following flags are available:

FESF_DIRS Directories should be examined by **FileEnum()**.

FESF_NON_GEOS
Non-GEOS files should be examined by **FileEnum()**.

Routines

FESF_GEOS_EXECS
: GEOS executable files should be examined by **FileEnum()**.

FESF_GEOS_NON_EXECS
: GEOS non-executable files (e.g., VM files) should be examined by **FileEnum()**.

FESF_REAL_SKIP
: If a skip count of *n* is specified, the first *n* files will be skipped regardless of whether they matched the attributes passed. In this case, **FileEnum()** will return the number of files passed through in order to get enough files to fill the buffer; the return value can thus be the real-skip count for the next pass.

FESF_CALLBACK
: **FileEnum()** should call a callback routine to determine whether a file should be accepted.

FESF_LOCK_CB_DATA
: This flag indicates that the **FileEnumParams** fields *FEP_callback1* and *FEP_callback2* are far pointers to movable memory that must be locked before **FileEnum()** is called.

FESF_LEAVE_HEADER
: If set, **FileEnum()** should leave an empty header space at the start of the return buffer. The size of this buffer is specified by the *FEP_headerSize* field.

The **FileEnumStandardReturnType** enumerated type has the following values; they are used in conjunction with the *FEP_returnAttrs* field of the **FileEnumParams** structure.

FESRT_COUNT_ONLY
: **FileEnum()** will not allocate any memory and will not return data about files; instead, it will simply return the number of files which match the specified criteria.

FESRT_DOS_INFO
: **FileEnum()** will return an array of **FEDosInfo** structures. These structures contain basic information about the file: its virtual name, size, modification date, DOS attributes, and path information (as a **DirPathInfo** record).

FESRT_NAME
: **FileEnum()** will return an array of **FileLongName** strings, each one of which is FILE_LONGNAME_BUFFER_SIZE characters long; every one of these will contain a file's virtual name followed by a null terminator.

FESRT_NAME_AND_ATTR
: **FileEnum()** will return an array of **FENameAndAttr** structures, each one of which contains a file's DOS attributes and virtual name.

The **FEDosInfo** structure includes a word-sized record (*DFIS_pathInfo*) which describes the file's position relative to the standard paths. It contains the following fields:

DPI_EXISTS_LOCALLY
: This bit is set if the file exists in a directory under the primary tree.

DPI_ENTRY_NUMBER_IN_PATH
: This is the mask for a seven-bit field whose offset is DPI_ENTRY_NUMBER_IN_PATH_OFFSET.

DPI_STD_PATH
: This is the mask for an eight-bit field whose offset is DPI_STD_PATH_OFFSET. If the file is in a standard path, this field will contain a **StandardPath** constant for a standard path containing the file. This need not be the "closest" standard path; for example, if the file is in the "World" directory, this constant might nevertheless be SP_TOP.

See Also: FileEnumLocateAttr(), FileEnumWildcard()

Include: fileEnum.h

■ FileEnumLocateAttr()

```
void *   FileEnumLocateAttr( /* returns NULL if attr not found */
         FileEnumCallbackData*   fecd,  /* Passed to callback routine */
         FileExtendedAttribute   attr,  /* Search for this attribute */
         const char *            * name);  /* Attribute name (if second
                                            * argument is FEA_CUSTOM) */
```

FileEnum() can be instructed to call a callback routine to decide which files to filter out. This callback routine is passed an array of **FileExtAttrDesc** structures. To find a particular extended attribute in this array, call **FileEnumLocateAttr()**. This routine will find the address of the value of the attribute desired, and return that address. If the attribute is not in the array, **FileEnumLocateAttr()** will return a null pointer.

Include: fileEnum.h

■ FileEnumWildcard()

```
Boolean  FileEnumWildcard(
         FileEnumCallbackData   * fecd,   /* Passed to callback routine */
         word                   frame);   /* Inherited stack frame */
```

This routine is a utility used by **FileEnum()** and is rarely used by applications. It checks to see if the virtual name of the current file (the file currently being evaluated by **FileEnum()**) matches the pattern in the *FEP_cbData1* field of the **FileEnumParams** structure.

The *fecd* parameter is a pointer to the callback data of the **FileEnum()** routine. The frame parameter is a pointer to the **FileEnum()** stack frame: The first dword is the *FEP_cbData1* field, and the second is the *FEP_cbData2* field.

This routine returns *true* (non-zero) if the file name and pattern match. Otherwise, it returns *false*.

Include: fileEnum.h

■ FileFromTransferBlockID()

```
VMFileHandle FileFromTransferBlockID(id);
         TransferBlockID id;
```

This macro extracts a VMFileHandle from a value of type **TransferBlockID**.

■ FileGetAttributes()

```
FileAttrs FileGetAttributes( /* Sets thread's error value */
         const char * path);    /* file's path relative to current
                               * working directory */
```

This routine returns the standard **FileAttrs** attributes for a file. The file may be a GEOS file or a plain DOS file. Note that you can also get a file's attributes by getting the file's FEA_FILE_ATTR extended attribute. If an error occurs, this routine sets the thread's error.

See Also: FileAttrs, FileSetAttributes()

Include: file.h

■ FileGetCurrentPath()

```
DiskHandle FileGetCurrentPath(
          char * buffer,            /* Path string is written here */
          word   bufferSize);       /* Size of buffer in bytes */
```

This routine writes the current path string (without drive specifier) to the buffer provided. If the buffer is too small, it truncates the path to fit. It returns the handle of the disk containing the current path. If the current path was declared relative to a standard path, the standard path constant will be returned.

Include: file.h

■ FileGetDateAndTime()

```
FileDateAndTime FileGetDateAndTime( /* sets thread's error value */
          FileHandle fh);
```

This routine finds out the time a file was last modified. This routine can be called on GEOS or non-GEOS files. Note that you can also find out the modification time of a file by checking the extended attribute FEA_MODIFICATION. If unsuccessful, it sets the thread's error value.

See Also: FileDateAndTime, FileSetDateAndTime()

Include: file.h

■ FileGetDiskHandle()

```
DiskHandle FileGetDiskHandle( /* sets thread's error value */
          FileHandle fh);
```

This routine returns the handle of the disk containing an open file. If unsuccessful, it sets the thread's error value.

Include: file.h

■ FileGetHandleExtAttributes()

```
word      FileGetHandleExtAttributes(
          FileHandle              fh,           /* open file's handle */
          FileExtendedAttribute   attr,         /* attribute to get */
          void                    * buffer,     /* attribute is written here */
          word                    bufSize);     /* length of buffer in bytes */
```

This routine gets one or more extended attributes of an open file. (To get the attributes of a file without opening it, call **FileGetPathExtAttributes()**.) If a single attribute is requested, the attribute will be written in the buffer passed. If several attributes are requested, *attr* should be set to

FEA_MULTIPLE, and *buffer* should point to an array of **FileExtAttrDesc** structures. In this case, *bufSize* should be the number of structures in the buffer, not the length of the buffer.

If **FileGetHandleExtAttributes()** is successful, it returns zero. Otherwise, it returns one of the following error codes:

ERROR_ATTR_NOT_SUPPORTED
The file system does not recognize the attribute constant passed.

ERROR_ATTR_SIZE_MISMATCH
The buffer passed was too small for the attribute requested.

ERROR_ATTR_NOT_FOUND
The file does not have a value set for that attribute.

ERROR_ACCESS_DENIED
You do not have read-access to the file.

Tips and Tricks: Note that the only way to recover a custom attribute is by passing FEA_MULTIPLE, and using a **FileExtAttrDesc** to describe the attribute.

See Also: FileGetPathExtAttributes()

Include: file.h

■ FileGetPathExtAttributes()

```
word      FileGetPathExtAttributes(
          const char             * path,       /* path relative to current
                                              * working directory */
          FileExtendedAttribute  attr,         /* attribute to get */
          void                   * buffer,     /* attribute is written here */
          word                   bufSize);     /* length of buffer in bytes */
```

This routine gets one or more extended attributes of a GEOS file. If a single attribute is requested, the attribute will be written in the buffer passed. If several attributes are requested, *attr* should be set to FEA_MULTIPLE, and *buffer* should point to an array of **FileExtArtrDesc** structures. In this case, *bufSize* should be the number of structures in the buffer, not the length of the buffer.

If **FileGetPathExtAttributes()** is successful, it returns zero. Otherwise, it returns one of the following error codes:

ERROR_ATTR_NOT_SUPPORTED
The file system does not recognize the attribute constant passed.

ERROR_ATTR_SIZE_MISMATCH
The buffer passed was too small for the attribute requested.

ERROR_ATTR_NOT_FOUND
The file does not have a value set for that attribute.

ERROR_ACCESS_DENIED
You do not have read-access to the file.

Tips and Tricks: Note that the only way to recover a custom attribute is by passing FEA_MULTIPLE, and using a **FileExtAttrDesc** to describe the attribute.

See Also: FileGetHandleExtAttributes()

Include: file.h

■ FileLockRecord()

```
word      FileLockRecord( /* returns error */
          FileHandle          fh,
          dword               filePos,  /* lock starting at this position... */
          dword               regLength);  /* lock this many bytes */
```

This routine puts a lock on a part of a byte file. It first checks to make sure that there are no locks that overlap the region specified; if there are, it will fail and return ERROR_ALREADY_LOCKED. If there are no locks, it will place a lock on the region specified and return zero.

Warnings: Locking a region only prevents threads from locking part of the same region; it does not prevent them from reading from or writing to the region. If applications use this mechanism, they have to make sure to call **FileLockRecord** before trying to access a part of a file.

See Also: FileUnlockRecord(), HandleP()

■ FileMove()

```
word      FileMove( /* Returns error */
          const char          * source,    /* source path and file name */
          const char          * dest,      /* destination path and file name */
          DiskHandle          sourceDisk,  /* These handles may be Standard */
          DiskHandle          destDisk);   /* Path constants, or null to indi-
                                         * cate current working directory */
```

This routine moves a file from one location to another. The source and destination are specified with path strings. Each string specifies a path relative to the location specified by the corresponding disk handle. If the handle is a disk handle, the path is relative to that disk's root. If the disk handle is a standard path constant, the path string is relative to that standard path. If the disk handle is null, the path is relative to the current working directory.

If **FileMove()** is successful, it returns zero. Otherwise, it returns one of the following error codes and sets the thread's error value.

ERROR_FILE_NOT_FOUND
: No such source file exists in the specified directory.

ERROR_PATH_NOT_FOUND
: An invalid source or destination path string was passed.

ERROR_ACCESS_DENIED
: You do not have permission to delete the source file, or there is already a file with the same name as the destination file (and you do not have permission to delete it), or the destination disk or directory is not writable.

ERROR_FILE_IN_USE
: Either the source file is in use, or there is already a file with the same name as the destination file, and it is in use.

ERROR_SHORT_READ_WRITE
: There was not enough room on the destination disk.

See Also: FileCopy()

Include: file.h

■ FileOpen()

```
FileHandle FileOpen( /* sets thread's error value */
          const char          * name,   /* relative to working dir */
          FileAccessFlags     flags);   /* Permissions/exclusions */
```

This routine opens a file for bytewise access. The file may be a DOS file or a GEOS byte file. If the file is successfully opened, **FileOpen()** will return the file's handle; otherwise, it will return a null handle and set the thread's error value. Errors typically set by this routine are listed below:

ERROR_FILE_NOT_FOUND
: No file with the specified name could be found in the appropriate directory.

ERROR_PATH_NOT_FOUND
: A relative or absolute path had been passed, and the path included a directory which did not exist.

ERROR_TOO_MANY_OPEN_FILES
: There is a limit to how many files may be open at once. If this limit is reached, **FileOpen()** will fail until a file is closed.

ERROR_ACCESS_DENIED
: Either the caller requested access which could not be granted (e.g. it requested write access when another geode had already opened the file with FILE_DENY_W), or the caller tried to deny access when that access had already been granted to another geode (e.g. it tried to open the file with FILE_DENY_W when another geode already had it open for write-access).

ERROR_WRITE_PROTECTED
: The caller requested write or read-write access to a file in a write-protected volume.

See Also: FileCreate()

Include: file.h

■ FileParseStandardPath()

```
StandardPath FileParseStandardPath(
          DiskHandle          disk,
          const char          ** path);
```

This routine is passed a full path (relative to the passed disk or a standard path, if the disk handle is null) and finds the standard path which most closely contains that path. It updates the pointer whose address is passed so that it points to the trailing portion of the path string. For example, if you pass the path string “\GEOWORKS\DOCUMENT\MEMOS\APRIL”, the pointer would be updated to point to the “\MEMOS\APRIL” portion, and the **StandardPath** SP_DOCUMENT would be returned. If the path passed does not belong to a standard path, the constant SP_NOT_STANDARD_PATH will be returned, and the pointer will not be changed.

Include: file.h

■ FilePopDir()

```
void      FilePopDir();
```

FilePopDir() pops the top directory off the thread’s directory stack and makes it the current working directory.

See Also: FilePushDir()

Include: file.h

■ FilePos()

```
dword    FilePos( /* Sets thread’s error value */
         FileHandle         fh,
         dword              posOrOffset,
         FilePosMode        mode);
```

This routine changes the current file position. The position can be specified in three ways, depending on the value of the *mode* argument:

FILE_POS_START
: The file position is set to a specified number of bytes after the start of the file. Passing this mode with an offset of zero will set the file position to the start of the file.

FILE_POS_RELATIVE
: The file position is incremented by a specified number of bytes; this number may be negative.

FILE_POS_END
: The file position is set to a specified number of bytes after the end of the file; it is usually passed with a negative number of bytes. Passing this mode with an offset of zero will set the file position to the end of the file.

FilePos() returns a 32-bit integer. This integer specifies the absolute file position after the move (relative to the start of the file).

Tips and Tricks: To find out the current file position without changing it, call **FilePos()** with mode FILE_POS_RELATIVE and offset zero.

Include: file.h

■ FilePushDir()

```
void     FilePushDir();
```

FilePushDir() pushes the current working directory onto the thread’s directory stack. It does not change the current working directory.

See Also: FilePopDir()

Include: file.h

■ FileRead()

```
word      FileRead( /* sets thread’s error value */
          FileHandle          fh,        /* handle of open file */
          void                * buf,     /* copy data to this buffer */
          word                count,     /* Length of buffer (in bytes) */
          Boolean             noErrorFlag);    /* Set if app can’t
                                                * handle errors */
```

This routine copies data from a file into memory. It starts copying from the current position in the file. If possible, it will copy enough data to fill the buffer. If **FileRead()** reaches the end of the file, it sets the thread’s error value to ERROR_SHORT_READ_WRITE. In any event, it returns the number of bytes copied. If an error occurs, **FileRead()** returns -1 and sets the thread’s error value (usually to ERROR_ACCESS_DENIED). The current file position will be changed to the first byte after the ones which were read.

If the argument *noErrorFlag* is set to *true* (i.e. non-zero), **FileRead()** will fatal-error if an error occurs (including an ERROR_SHORT_READ_WRITE).

Warnings: Pass *noErrorFlag true* only during debugging.

Include: file.h

■ FileRename()

```
word      FileRename(
          const char * oldName,      /* Relative to working directory */
          const char * newName);     /* Name only, without path */
```

This routine changes a file’s name. It cannot move a file to a different directory; to do that, call **FileMove()**. If the routine is successful, it returns zero; otherwise, it returns a **FileError**. Common errors include

ERROR_FILE_NOT_FOUND
No such file exists in the specified directory.

ERROR_PATH_NOT_FOUND
An invalid path string was passed.

ERROR_ACCESS_DENIED
You do not have permission to delete that file, or it exists on a read-only volume.

ERROR_FILE_IN_USE
Some geode has that file open.

ERROR_INVALID_NAME
The name was not a valid GEOS name; or the file is a non-GEOS file, and the name was not an appropriate native name.

See Also: FileMove()

Include: file.h

■ FileResolveStandardPath()

```
DiskHandle FileResolveStandardPath(
          char              ** buffer,    /* Write path here; update pointer
                                         * to point to end of path */
          word              bufSize,      /* Size of buffer (in bytes) */
          const char *      path,         /* Relative path of file */
          FileResolveStandardPathFlags flags); /* Flags are described below */
```

This routine finds a file relative to the current location, then writes the full path to the file, starting at the root of the disk (*not* at a standard path). It writes the path to the passed buffer, updating the pointer to point to the null at the end of the path string; it also returns the handle of the disk. If it cannot find the file it returns a null path.

Structures: A record of **FileResolveStandardPathFlags** is passed to **FileResolveStandardPath()**. The following flags are available:

FRSPF_ADD_DRIVE_NAME
The path string written to the buffer should begin with the drive name (e.g., "C:\GEOWORKS\DOCUMENT\MEMOS").

FRSPF_RETURN_FIRST_DIR
FileResolveStandardPath() should not check whether the passed path actually exists; instead, it should assume that the path exists in the first directory comprising the standard path, and return accordingly.

Include: file.h

■ FileSetAttributes()

```
word      FileSetAttributes( /* returns error value */
          const char        * path,    /* file's path relative to current
                                       * working directory */
          FileAttrs         attr);     /* new attributes for the file */
```

This routine changes the standard DOS attributes of a DOS or GEOS file. Note that you can also change the attributes of a file by setting the extended attribute FEA_FILE_ATTR.

See Also: FileAttrs, FileGetAttrs()

Include: file.h

■ FileSetCurrentPath()

```
DiskHandle FileSetCurrentPath(
          DiskHandle          disk,     /* May be a standard path constant */
          const char          * path);  /* path string, null-terminated */
```

This routine changes the current path. It is passed two parameters: The first is the handle of the disk containing the new current path (this may be a standard path constant). The second is a null-terminated path string. It is specified with normal DOS conventions: directories are separated by backslashes; a period (".") indicates the current directory; and a pair of periods ("..") indicates the parent of the current directory. The string may not contain wildcard characters.

If *disk* is a disk handle, the path is relative to the root directory of that disk; if *disk* is a standard path constant, the path is relative to the standard path; if it is null, the path is relative to the current working directory. **FileSetCurrentPath()** returns the disk handle associated with the new current path; this may be a standard path constant. If **FileSetCurrentPath()** fails, it returns a null handle.

Include: file.h

■ FileSetDateAndTime()

```
word      FileSetDateAndTime( /* returns error */
          FileHandle          fh,               /* handle of open file */
          FileDateAndTime     dateAndTime);     /* new modification time */
```

This routine changes a file's last-modification time-stamp. This routine can be called on GEOS or non-GEOS files. Note that you can also change the modification time of a file by changing the extended attribute FEA_MODIFICATION. If unsuccessful, this routine returns an error and sets the thread's error value.

See Also: FileDateAndTime, FileGetDateAndTime()

Include: file.h

■ FileSetHandleExtAttributes()

```
word      FileGetPathExtAttributes( /* returns error */
          FileHandle             fh,       /* handle of open file */
          FileExtendedAttribute  attr,     /* attribute to get */
          const void             * buffer, /* attribute is read from here */
          word                   bufSize); /* length of buffer in bytes */
```

This routine sets one or more extended attributes of an open GEOS file. (To set the attributes of a file without opening it, call **FileSetPathExtAttributes()**.) If a single attribute is specified, the attribute's new value will be read from the buffer passed. If several attributes are to be changed, *attr* should be set to FEA_MULTIPLE, and *buffer* should point to an array of **FileExtAttrDesc** structures. In this case, *bufSize* should be the number of structures in the buffer, not the length of the buffer.

If **FileSetHandleExtAttributes()** is successful, it returns zero. Otherwise, it sets the thread's error value and returns one of the following error codes:

ERROR_ATTR_NOT_SUPPORTED
The file system does not recognize the attribute constant passed.

ERROR_ATTR_SIZE_MISMATCH
The buffer passed was the wrong size for the attribute specified.

ERROR_ACCESS_DENIED
The caller does not have write-access to the file.

ERROR_CANNOT_BE_SET
The extended attribute cannot be changed. Such attributes as FEA_SIZE and FEA_NAME cannot be changed with the **FileSet...()** routines.

Tips and Tricks: Note that the only way to create or change a custom attribute is by passing FEA_MULTIPLE, and using a **FileExtAttrDesc** to describe the attribute.

See Also: FileSetPathExtAttributes()

Include: file.h

■ FileSetPathExtAttributes()

```
word      FileSetPathExtAttributes(
          const char             * path,   /* path relative to current
                                            * working directory */
          FileExtendedAttribute   attr,    /* attribute to get */
          const void             * buffer, /* attribute is read from here */
          word                   bufSize); /* length of buffer in bytes */
```

This routine sets one or more extended attributes of a file. If a single attribute is specified, the attribute will be written in the buffer passed. If several attributes are to be changed, *attr* should be set to FEA_MULTIPLE and *buffer* should point to an array of **FileExtAttrDesc** structures. In this case, *bufSize* should be the number of structures in the buffer, not the length of the buffer.

If **FileSetPathExtAttributes()** is successful, it returns zero. Otherwise, it sets the thread's error value and returns one of the following error codes:

ERROR_ATTR_NOT_SUPPORTED
: The file system does not recognize the attribute constant passed.

ERROR_ATTR_SIZE_MISMATCH
: The buffer passed was the wrong size for the attribute specified.

ERROR_ACCESS_DENIED
: **FileSetPathExtAttributes()** returns this if any geode (including the caller) has the file open with "deny-write" exclusive access, or if the file is not writable.

ERROR_CANNOT_BE_SET
: The extended attribute cannot be changed. Such attributes as FEA_SIZE and FEA_NAME cannot be changed with the **FileSet...ExtAttributes()** routines.

Tips and Tricks: Note that the only way to create or change a custom attribute is by passing FEA_MULTIPLE, and using a **FileExtAttrDesc** to describe the attribute.

See Also: FileSetHandleExtAttributes()

Include: file.h

■ FileSetStandardPath()

```
void      FileSetStandardPath(
          StandardPath path);    /* StandardPath to set */
```

This routine changes the current working directory to one of the system's StandardPath directories. Pass a standard path.

Include: file.h

■ FileSize()

```
dword     FileSize(
          FileHandle fh);     /* handle of open file */
```

This routine returns the size of the open file specified.

Include: file.h

■ FileTruncate()

```
word      FileTruncate(
          FileHandle          fh,       /* handle of open file */
          dword               offset);  /* offset at which to truncate */
```

This routine truncates the specified file at the passed offset. The *offset* parameter can also be thought of as the desired file size.

Include: file.h

■ FileUnlockRecord()

```
word      FileUnlockRecord( /* returns error */
          FileHandle          fh,           /* handle of open file
          dword               filePos,      /* Release lock that starts here */
          dword               regLength);   /* and is this long */
```

This routine releases a lock on a part of a byte-file. The lock must have been previously placed with **FileLockRecord()**.

See Also: FileLockRecord(), HandleV()

Include: file.h

■ FileWrite()

```
word      FileWrite( /* sets thread’s error value */
          FileHandle          fh,           /* handle of open file */
          const void          * buf,        /* Copy from here into file */
          word                count,        /* # of bytes to copy */
          Boolean             noErrorFlag); /* Set if can’t handle errors */
```

This routine copies a specified number of bytes from a buffer to the file. The bytes are written starting with the current position in the file; any data already at that location will be overwritten. **FileWrite()** returns the number of bytes written. If **FileWrite()** could not write all the data (e.g. if the disk ran out of space), it will set the thread’s error value to ERROR_SHORT_READ_WRITE and return the number of bytes that were written. If it could not write the data to the file at all (e.g. if you do not have write-access to the file), it will return -1 and set the thread’s error value to ERROR_ACCESS_DENIED. In any event, the file position will be changed to the first byte after the ones written.

If the argument *noErrorFlag* is set to *true* (i.e. non-zero), **FileWrite()** will fatal-error if an error occurs.

Warnings: Pass *noErrorFlag true* only during debugging.

Include: file.h

■ FormatIDFromManufacturerAndType

```
dword     FormatIDFromManufacturerAndType(mfr, type);
          ManufacturerIDs        mfr;
          ClipboardItemFormat    type;
```

This macro takes a manufacturer ID and a format type (e.g. CIF_TEXT) and combines them into a dword argument of the type **ClipboardItemFormatID**.

■ free()

```
void      free(
          void * blockPtr);          /* address of memory to free */
```

The **malloc()** family of routines is provided for Standard C compatibility. The kernel will allocate a fixed block to satisfy the geode’s **malloc()** requests; it will allocate memory from this block. When the block is filled, it will allocate another fixed malloc-block. When all the memory in the block is freed, the memory manager will automatically free the block.

When a geode is finished with some memory it requested from **malloc()**, it should free the memory. That makes it easier for **malloc()** to satisfy memory request. It can free the memory by passing the address which was returned by **malloc()** (or **calloc()** or **realloc()**) when the memory was allocated. All of the memory will be freed.

The memory must be in a malloc-block assigned to the geode calling **free()**. If you want to free memory in another geode's malloc-block, call **GeoFree()**.

Include: stdlib.h

Warnings: Pass exactly the same address as the one returned to you when you allocated the memory. If you pass a different address, **free()** will take unpredictable actions, including possibly erasing other memory or crashing the system.

See Also: calloc(), malloc(), GeoFree(), realloc()

■ FractionOf()

```
word        FractionOf(
            WWFixedAsDWord      wwf);
```

This macro returns the fractional portion of a **WWFixedAsDWord** value.

Include: geos.h

Routines

■ GCNListAdd()

```
Boolean  GCNListAdd(
         optr              OD,          /* optr to add to list */
         ManufacturerID    manufID,     /* manufacturer ID of list */
         word              listType);   /* list type */
```

This routine adds an object pointer (optr) to a GCN list interested in a particular change. The routine must be passed the optr to add, along with the *manufID* and the type of the list to add it to. If no list of the specified manufacturer and type currently exists, a new list will be created.

This routine will return *true* if the optr was successfully added to the GCN list and *false* if the optr could not be added. An optr cannot be added to a GCN list if it currently exists on that list.

Include: gcnlist.goh

■ GCNListAddHandles()

```
Boolean  GCNListAddHandles(
         MemHandle         mh,          /* handle of object to add */
         ChunkHandle       ch,          /* chunk of object to add */
         ManufacturerIDs   manufID,     /* manufacturer ID of list */
         word              listType);   /* list type */
```

This routine is exactly the same as **GCNListAdd()**, except it takes the memory and chunk handles of the object rather than a complete optr.

Include: gcnlist.goh

■ GCNListAddToBlock()

```
Boolean  GCNListAddToBlock(
         optr              OD,          /* optr of list to add */
         ManufacturerID    manufID,     /* manufacturer ID of list */
         word              listType,    /* list type */
         MemHandle         mh,          /* handle of block holding list */
         ChunkHandle       listOfLists);/* chunk of list of lists
                                         * in block */
```

This routine adds a new GCN list to a block containing the GCN lists. Pass it the optr of the chunk containing the new GCN list as well as the list's type and manufacturer ID. Pass also the memory handle and chunk handle of the chunk containing the GCN "list of lists" which will manage the new list.

This routine returns true of the new optr is added to the GCN mechanism, false if it could not be added (if it was already there).

Warnings: This routine may resize chunks in the block, so you should dereference any pointers after calling this routine.

Include: gcnlist.goh

■ GCNListCreateBlock()

```
ChunkHandle GCNListCreateBlock(
          MemHandle mh);         /* handle of the locked LMem block */
```

This routine creates a list of lists for the GCN mechanism. It is rarely, if ever, called by applications. Pass it the handle of the locked LMem block in which the list should be created.

Include: gcnlist.goh

■ GCNListDestroyBlock()

```
void      GCNListDestroyBlock(
          MemHandle          mh,              /* handle of locked block to
                                            * be destroyed */
          ChunkHandle        listOfLists);    /* chunk of list of lists */
```

This routine destroys a GCN list of lists and all the GCN lists associated with it. Pass it the handle of the locked LMem block containing the lists as well as the chunk handle of the chunk containing the list of lists.

Include: gcnlist.goh

■ GCNListDestroyList()

```
void      GCNListDestroyList(
          optr   list);          /* optr of the GCN list to be destroyed */
```

This routine destroys the specified GCN list.

Include: gcnlist.goh

■ GCNListRelocateBlock()

```
void      GCNListRelocateBlock(
          MemHandle          mh,              /* handle of locked LMem block
                                            * containing GCN lists */
          ChunkHandle        listOfLists,     /* chunk of list of lists */
          MemHandle          relocBlock);     /* handle of block containing
                                            * relocation information */
```

This routine relocates the GCN list of lists in the specified block, updating all the optrs stored therein.

Warnings: This routine can resize and/or move the LMem block, so you should dereference pointers after calling it.

Include: gcnlist.goh

■ GCNListRemove()

```
Boolean  GCNListRemove(
         optr                OD,           /* the optr to be removed */
         ManufacturerID      manufID,      /* manufacturer ID of the list */
         word                listType);    /* list type */
```

This routine removes the passed optr from the specified GCN list. The routine must be passed the optr to remove along with the manufacturer ID and list type of the list to remove it from.

This routine will return *true* if the optr was successfully removed from the GCN list and *false* if the optr could not be found on the GCN list and therefore could not be removed.

Include: gcnlist.goh

■ GCNListRemoveFromBlock()

```
Boolean  GCNListRemoveFromBlock(
         optr                OD,           /* optr of GCN list to remove */
         ManufacturerID      manufID,      /* manufacturer of list to remove */
         word                listType,     /* type of list being removed */
         MemHandle           mh,           /* handle of locked LMem block
                                          * containing the list of lists */
         ChunkHandle         listOfLists); /* chunk of list of lists */
```

This routine removes a GCN list from a GCN list block and from the list of lists therein.

Include: gcnlist.goh

■ GCNListRemoveHandles()

```
Boolean  GCNListRemoveHandles(
         MemHandle           mh,
         ChunkHandle         ch,
         ManufacturerID      manufID,
         word                listType);
```

This routine is exactly the same as **GCNListRemove()**, except it specifies the object to be removed via handles rather than an optr.

Include: gcnlist.goh

See Also: GCNListRemove()

■ GCNListSend()

```
word      GCNListSend(
          ManufacturerID      manufID,            /* manufacturer of list */
          word                listType,           /* notification type */
          EventHandle         event,              /* event to be sent to list */
          MemHandle           dataBlock,          /* data block, if any */
          word                gcnListSendFlags);  /* GCNListSendFlags */
```

This routine sends a message to all objects in the specified GCN list. The message is specified in *event*, and the list is specified in *manufID* and *listType*. The message will be sent asynchronously (some time after the change has occurred) by the message queue.

The *dataBlock* parameter contains the memory handle of an extra data block to be sent with the notification, if any; this block should also be specified in the classed event. If no data block is required, pass a NullHandle. If a data block with a reference cound is used, increment the reference count by one before calling this routine; this routine decrements the count and frees the block if the count reaches zero.

The *gcnListSendFlags* parameter is of type **GCNListSendFlags**, which has only one meaningful flag for this routine:

GCNLSF_SET_STATUS
: Causes the message sent to the GCN list to be set as the lists "status." The list's status message is then sent to any object adding itself to the list at a later time. If this flag is set, the event handle in *event* will be returned by the routine. If this flag is not set, the return value will be the number of messages sent out.

Include: gcnlist.goh

■ GCNListSendToBlock()

```
word      GCNListSendToBlock(
          ManufacturerID      manufID,      /* manufacturer id of list */
          word                listType,     /* notification type */
          EventHandle         event,        /* event to be sent to list */
          MemHandle           dataBlock,    /* data block, if any */
          MemHandle           mh,           /* handle of locked LMem block
                                           * containing GCN list of lists */
          ChunkHandle         listOfLists,  /* chunk of list of lists */
          GCNListSendFlags    flags);       /* GCNListSendFlags */
```

This routine sends the specified *event* to the specified list, just as **GCNListSend()**. **GCNListSentToBlock()**, however, specifies a particular instance of the GCN list by specifying the appropriate list of lists in *mh* and *listOfLists*. Other parameters and return values are identical to **GCNListSend()**.

See Also: GCNListSend()

Include: gcnlist.goh

■ GCNListSendToList()

```
void      GCNListSendToList(
          optr                list,         /* optr of GCN list */
          EventHandle         event,        /* event to send to list */
          MemHandle           dataBlock,    /* handle of data block, if any */
          GCNListSendFlags    flags);       /* GCNListSendFlags */
```

This routine sends the specified *event* to the specified GCN *list*. The list is specified explicitly by optr as opposed to by manufacturer ID and type. The event will be sent via the proper queues to all objects registered on the list. After the notification is handled by all notified objects, the event will be freed, as will the data block passed. (If no data block, pass NullHandle in *dataBlock*)

The *flags* parameter can have one flag, GCNLSF_SET_STATUS. If this flag is set, the event passed will be set as the list's status message.

Include: gcnlist.goh

See Also: GCNListSend()

■ GCNListSendToListHandles()

```
void      GCNListSendToListHandles(
          MemHandle           mh,            /* handle of list’s block */
          ChunkHandle         ch,            /* chunk of list */
          EventHandle         event,         /* event to send to list */
          MemHandle           dataBlock,     /* handle of data block, if any */
          GCNListSendFlags    flags);        /* GCNListSendFlags */
```

This routine is exactly the same as **GCNListSendToList()**; the list is specified not by optr, however, but by a combination of its global and chunk handles.

See Also: GCNListSendToList()

Include: gcnlist.goh

■ GCNListUnRelocateBlock()

```
Boolean   GCNListUnRelocateBlock(
          MemHandle           mh,            /* handle of the locked lmem block
                                           * containing the list of lists */
          ChunkHandle         listOfLists,   /* chunk of the list of lists */
          MemHandle           relocBlock);   /* handle of block containing
                                           * relocation/unrelocation info */
```

This routine unrelocates the specified list of lists, updating all the optrs according to the information in *relocBlock*. This routine is rarely, if ever, used by applications; it is used primarily by the UI when shutting down to a state file.

It returns *true* if the specified list of lists has no lists saved to state and therefore is simply destroyed. The return value is *false* if the list of lists is saved to the state file normally.

Include: gcnlist.goh

■ GenCopyChunk()

```
word      GenCopyChunk(
          MemHandle           destBlock,/* handle of locked LMem block into
                                     * which chunk will be copied */
          MemHandle           blk,      /* handle of locked source LMem block */
          ChunkHandle         chnk,     /* chunk handle of chunk to be copied */
          word                flags);   /* CompChildFlags */
```

This is a utility routine that copies one LMem chunk into a newly created chunk. The routine will allocate the new chunk in the block passed in *destBlock* and will return the chunk handle of the new chunk. It is used primarily by the UI to duplicate generic object chunks.

The source chunk is specified by the global handle *blk* and the chunk handle *chnk*. The *flags* parameter contains a record of **CompChildFlags**, of which only the CCF_MARK_DIRTY flag is meaningful. If this flag is set, the new chunk will be marked dirty.

Warnings: This routine may resize and/or move chunks and blocks, so you must dereference pointers after calling it.

Include: genC.goh

■ GenFindObjectInTree()

```
optr    GenFindObjectInTree(
        optr   startObject,     /* optr of object at which to start search */
        dword  childTable);     /* pointer to table of bytes, each indicating
                               * the position of the child at the given
                               * level; -1 is the end of the table */
```

This utility routine finds the object having the optr *startObject* in the generic tree. Applications will not likely need this routine.

The childTable parameter points to a table of bytes, each byte representing the child number to be found at each level. The first byte indicates the child of startObject to get; the second byte indicates the child to get at the next level; the third byte indicates the child to get at the next level, and so on. A byte of -1 indicates the end of the table. The object found will be returned.

Include: genC.goh

■ GenInsertChild()

```
void    GenInsertChild(
        MemHandle         mh,             /* handle of parent */
        ChunkHandle       chnk,           /* chunk of parent */
        optr              childToAdd,     /* optr of new child */
        optr              referenceChild, /* optr of reference child */
        word              flags);         /* CompChildFlags */
```

This utility routine adds a child object to a composite object. It is used almost exclusively by the UI for generic objects—applications will typically use MSG_GEN_ADD_CHILD.

See Also: MSG_GEN_ADD_CHILD

Warnings: This routine may move or resize chunks and/or object blocks; therefore, you must dereference pointers after calling it.

Include: genC.goh

■ GenProcessAction()

```
void      GenProcessAction(
          MemHandle           mh,    /* handle of object calling the routine */
          ChunkHandle         chnk,  /* chunk of object calling the routine */
          word                mthd,        /* message to send to actionOptr */
          word                dataCX,      /* data to pass in CX register */
          word                dataDX,      /* data to pass in DX register */
          word                dataBP,      /* data to pass in BP register */
          optr                actionOptr); /* object to receive mthd */
```

This utility routine sends the action message specified in *mthd* to the action object specified in *actionOptr*. It is typically used by the UI and generic objects and corresponds to the **GenClass** message MSG_GEN_OUTPUT_ACTION.

Warnings: This routine may move or resize chunks and/or object blocks; therefore, you must dereference pointers after calling it.

See Also: MSG_GEN_OUTPUT_ACTION

Include: genC.goh

■ GenProcessGenAttrsAfterAction()

```
void      GenProcessGenAttrsAfterAction(
          MemHandle           mh,    /* handle of object calling the routine */
          ChunkHandle         chnk); /* chunk of object calling the routine */
```

This utility routine processes various attributes for a generic object after the object's action message has been sent. It is used almost exclusively by the generic UI after MSG_GEN_OUTPUT_ACTION or **GenProcessAction()**.

Warnings: This routine may move or resize chunks and/or object blocks; therefore, you must dereference pointers after calling it.

Include: genC.goh

■ GenProcessGenAttrsBeforeAction()

```
void      GenProcessGenAttrsBeforeAction(
          MemHandle           mh,    /* handle of object calling the routine */
          ChunkHandle         chnk); /* chunk of object calling the routine */
```

This utility routine processes various attributes for a generic object before the object's action message has been sent. It is used almost exclusively by the generic UI before MSG_GEN_OUTPUT_ACTION or **GenProcessAction()**.

Warnings: This routine may move or resize chunks and/or object blocks; therefore, you must dereference pointers after calling it.

Include: genC.goh

■ GenProcessUndoGetFile()

```
VMFileHandle GenProcessUndoGetFile();
```

This routine returns the handle of the file that holds the process' undo information.

Include: Objects/gProcC.goh

■ GenProcessUndoCheckIfIgnoring()

```
Boolean GenProcessUndoCheckIfIgnoring();
```

This routine returns *true* if the process is currently ignoring actions.

Include: Objects/gProcC.goh

■ GenRemoveDownwardLink()

```
void     GenRemoveDownwardLink(
         MemHandle          mh,       /* handle of calling object */
         ChunkHandle        chnk,     /* chunk of calling object */
         word               flags);   /* CompChildFlags */
```

This utility routine removes a child from the generic tree, preserving the child's upward link and usability flags. It is called primarily by the generic UI and is rarely used by applications. The flags parameter specifies whether the object linkage should be marked dirty by passing the CCF_MARK_DIRTY flag.

Warnings: This routine may move or resize chunks and/or object blocks; therefore, you must dereference pointers after calling it.

Include: genC.goh

■ GenSetUpwardLink()

```
void        GenSetUpwardLink(
            MemHandle           mh,         /* handle of calling object */
            ChunkHandle         chnk,       /* chunk of calling object */
            optr                parent);    /* optr of calling object’s parent */
```

This utility routine converts the child/parent link to an upward-only link. Pass the handle and chunk of the locked child object and the optr of the parent composite.

Include: genC.goh

■ GeodeAllocQueue()

```
QueueHandle GeodeAllocQueue();
```

This routine allocates an event queue which can then be attached to a thread with **ThreadAttachToQueue()**. It returns the queue’s handle if one is allocated; it will return zero otherwise. This routine is used outside the kernel only in exceptional circumstances.

Be Sure To: You must free the queue when you are done with it; use **GeodeFreeQueue()**.

Include: geode.h

■ GeodeDuplicateResource()

```
MemHandle GeodeDuplicateResource(
            MemHandle mh);      /* handle of geode resource to duplicate */
```

This routine reads a resource from a geode into a newly-allocated block (allocated by this routine). Any relocations on the resource to itself are adjusted to be the duplicated block. The handle of the duplicated block is returned.

Include: resource.h

■ GeodeFind()

```
GeodeHandle GeodeFind(
          const char          * name,       /* geode’s permanent name */
          word                numChars,     /* number of characters to match:
                                          * 8 for name, 12 for name.ext */
          GeodeAttrs          attrMatch,    /* GeodeAttrs that must be set */
          GeodeAttrs          attrNoMatch);/* GeodeAttrs that must be off */
```

This routine finds a geode given its permanent name, returning the geode handle if found. If the geode can not be found, a null handle will be returned. Pass it the following:

name	A pointer to the null-terminated permanent name of the geode.
numChars	The number of characters to match before returning. Pass GEODE_NAME_SIZE to match the permanent name, (GEODE_NAME_SIZE + GEODE_EXT_SIZE) to match the name and extension.
attrMatch	A record of **GeodeAttrs** the subject geode must have set for a positive match.
attrNoMatch	A record of **GeodeAttrs** the subject geode must have cleared for a positive match.

Include: geode.h

■ GeodeFindResource()

```
word      GeodeFindResource(
          FileHandle          file,         /* geode’s executable file */
          word                resNum,       /* resource number to find */
          word                resOffset,    /* offset to resource */
          dword               * base);      /* pointer to second return value */
```

This routine locates a resource within a geode’s executable (**.geo**) file. It returns the size of the resource as well as the base position of the first byte of the resource in the file (pointed to by *base*). Pass the following:

file	The file handle of the geode’s executable file.
resNum	The number of the resource to be found.
resOffset	The offset within the resource at which to position the file’s read/write position.

base A pointer to a dword value to be filled in by the routine. This value will be the base offset from the beginning of the file to the first byte of the resource.

Structures: A geode's executable file is laid out as shown below.

```
0:     Geode file header
1:     Imported Library Table
2:     Exported Routine Table
3:     Resource Size Table
4:     Resource Position Table
5:     Relocation Table Size Table
6:     Allocation Flags Table
7+:    application resources
```

Include: geode.h

■ GeodeFlushQueue()

```
void    GeodeFlushQueue(
        QueueHandle         source,   /* source queue to flush */
        QueueHandle         dest,     /* queue to hold flushed events */
        optr                obj       /* object to handle flushed events */
        MessageFlags        flags);   /* MF_INSERT_AT_FRONT or zero */
```

This routine flushes all events from one event queue into another, synchronously. Pass it the following:

source The queue handle of the source queue (the one to be emptied).

dest The queue handle of the destination queue that will receive the flushed events.

obj The object that will handle flushed events that were destined for the process owning the source queue. If the process owning the destination queue should be used, pass the destination queue handle in the handle portion of the optr and a null chunk handle.

flags A record of **MessageFlags**. The only meaningful flag for this routine is MF_INSERT_AT_FRONT, which should be set to flush source queue's events to the front of the destination queue. If this flag is not passed, events will be appended to the queue.

Include: geode.h

■ GeodeFreeQueue()

```
void        GeodeFreeQueue(
            QueueHandle qh);        /* handle of queue being freed */
```

This routine frees an event queue allocated with **GeodeAllocQueue()**. Any events still on the queue will be flushed as with **GeodeFlushQueue()**. You must pass the handle of the queue to be freed.

Include: geode.h

■ GeodeFreeDriver()

```
void        GeodeFreeDriver(
            GeodeHandle gh);        /* handle of the driver */
```

This routine frees a driver geode that had been loaded with **GeodeUseDriver()**. Pass it the geode handle of the driver as returned by that routine.

Include: driver.h

■ GeodeFreeLibrary()

```
void        GeodeFreeLibrary(
            GeodeHandle gh);        /* handle of the library */
```

This routine frees a library geode that had been loaded with **GeodeUseLibrary()**. Pass it the geode handle of the library.

Include: library.h

■ GeodeGetAppObject()

```
optr        GeodeGetAppObject(
            GeodeHandle gh);        /* handle of the application geode */
```

This routine returns the optr of the specified geode's GenApplication object. The geode should be an application. Pass zero to get the optr of the caller's application object.

Include: geode.h

■ GeodeGetCodeProcessHandle()

```
GeodeHandle GeodeGetCodeProcessHandle();
```

This routine returns the geode handle of the geode that owns the block in which the code which calls this routine resides.

Include: geode.h

■ GeodeGetDefaultDriver()

```
GeodeHandle GeodeGetDefaultDriver(
          GeodeDefaultDriverType type);    /* type of default driver to get */
```

This routine returns the default driver's geode handle for the type passed. The type must be one of the values of **GeodeDefaultDriverType**, which includes GDDT_FILE_SYSTEM (0), GDDT_KEYBOARD (2), GDDT_MOUSE (4), GDDT_VIDEO (6), GDDT_MEMORY_VIDEO (8), GDDT_POWER_MANAGEMENT(10), and GDDT_TASK(12).

Include: driver.h

■ GeodeGetInfo()

```
word      GeodeGetInfo(
          GeodeHandle          gh,       /* handle of the subject geode */
          GeodeGetInfoType     info,     /* type of information to return */
          void                 * buf);   /* buffer to contain returned info */
```

This routine returns information about the specified geode. The geode must be loaded already. The meaning of the returned word depends on the value passed in *info*; the **GeodeGetInfoType** is shown below. Pass the following:

gh	The geode handle of the geode.
info	The type of information requested; this should be one of the values listed below.
buf	A pointer to a locked or fixed buffer which will contain returned information for various types requested.

GeodeGetInfoType has the following enumerations (only one may be requested at a time):

GGIT_ATTRIBUTES
Get the geode's attributes. The return value will be a record of **GeodeAttrs** corresponding to those attributes set for the geode. Pass a null buffer pointer.

GGIT_TYPE
Get the type of the geode. The returned value will be a value of **GeosFileType** indicating the type of file storing the geode. Pass a null buffer pointer.

GGIT_GEODE_RELEASE
Get the release number of the geode. The returned word will be the size of the buffer pointed to by *buf*, and the buffer will contain the **ReleaseNumber** structure of the geode.

GGIT_GEODE_PROTOCOL
: Get the protocol level of the geode. The returned word will be the size of the buffer pointed to by *buf*, and the buffer will contain the **ProtocolNumber** structure of the geode.

GGIT_TOKEN_ID
: Get the token identifier of the geode. The returned word will be the size of the buffer pointed to by *buf*, and the buffer will contain a **GeodeToken** structure containing the token characters and token ID of the geode's token.

GGIT_PERM_NAME_AND_EXT
: Get the permanent name of the geode, with the extension characters. The returned word will be the size of the buffer pointed to by *buf*, and the buffer will contain a null-terminated character string representing the geode's permanent name (as set in its geode parameters file). Note that the buffer must be at least 13 bytes.

GGIT_PERM_NAME_ONLY
: Get the permanent name of the geode without the extension characters. The returned word will be the size of the buffer pointed to by buf, and the buffer will contain the null-terminated character string representing the geode's permanent name. The buffer must be at least nine bytes.

Include: geode.h

■ GeodeGetOptrNS()

```
optr      GeodeGetOptrNS(
          optr   obj);
```

This routine unrelocates an optr, changing the virtual-segment handle to an actual global handle.

Include: resource.h

■ GeodeGetProcessHandle()

```
GeodeHandle GeodeGetProcessHandle();
```

This routine returns the geode handle of the current executing process (i.e. the owner of the current running thread). Use it when you need to pass your application's geode handle or Process object's handle to a routine or message.

Include: geode.h

■ GeodeGetUIData()

```
word      GeodeGetUIData(
          GeodeHandle         gh);
```

Include: geode.h

■ GeodeInfoDriver()

```
DriverInfoStruct  * GeodeInfoDriver(
          GeodeHandle gh); /* handle of the driver to get information about */
```

This routine returns information about the specified driver geode. Pass the geode handle of the driver as returned by **GeodeUseDriver()**. It returns a pointer to a **DriverInfoStruct** structure, shown below.

```
typedef struct {
    void            (*DIS_strategy)();
    DriverAttrs     DIS_driverAttributes;
    DriverType      DIS_driverType;
} DriverInfoStruct;
```

For full information on this structure, see the **DriverInfoStruct** reference entry.

Include: driver.h

■ GeodeInfoQueue()

```
word      GeodeInfoQueue(
          QueueHandle qh);        /* queue to query */
```

This routine returns information about a specific event queue. Pass the handle of the queue; for information about the current process' queue, pass a null handle. This routine returns the number of events (or messages) currently in the queue.

Include: geode.h

■ GeodeLoad()

```
GeodeHandle GeodeLoad(
          const char *        name,         /* file name of geode */
          GeodeAttrs          attrMatch,    /* GeodeAttrs that must be set */
          GeodeAttrs          attrNoMatch,  /* GeodeAttrs that must be clear */
          word                priority,     /* priority of the loaded geode */
          dword               appInfo,      /* special load information */
          GeodeLoadError *    err);         /* returned error value */
```

This routine loads the specified geode from the given file and then executes the geode based on its type. It returns the geode handle of the loaded geode if successful; if unsuccessful, the returned value will be NullHandle and the *err* pointer will point to an error value. Pass this routine the following:

name	A pointer to the name of the geode's file. This is a null-terminated character string that represents the full path of the file (or a path relative to the current working directory).
attrMatch	A record of **GeodeAttrs** that must be set in the specified geode for the load to be successful.
attrNoMatch	A record of **GeodeAttrs** that must be cleared in the specified geode for the load to be successful. (That is, each bit which is set in *attrNoMatch* must be clear in the geode's **GeodeAttrs** field.)
priority	If the subject geode is a process, this is the priority at which its process thread will run.
appInfo	Two words of data to be passed directly to the loaded geode. For libraries and drivers, this should be a far pointer to a null-terminated string of parameters.
err	A pointer to an empty **GeodeLoadError** which will hold any returned error values.

Warnings: If you load a geode dynamically with **GeodeLoad()**, you must be sure to free it when you are done with **GeodeFree()**.

Include: geode.h

See Also: **UserLoadApplication()**

■ GeodeLoadDGroup

```
void    GeodeLoadDGroup(
        MemHandle           mh);
```

This routine forces the **dgroup** segment into the data-segment register.

Include: resource.h

■ GeodePrivAlloc()

```
word        GeodePrivAlloc(
            GeodeHandle         gh,          /* handle of the owner of the
                                            * newly-allocated private data */
            word                numWords);   /* number of words to allocate */
```

This routine allocates a string of contiguous words in all geodes' private data areas; each set of words will be owned by the geode specified in *gh*. The data allocated can be accessed with **GeodePrivWrite()** and **GeodePrivRead()** and must be freed with **GeodePrivFree()**. The return value will be the offset to the start of the allocated range, or zero if the routine could not allocate the space.

Each geode has a block of private data the is accessed using the **GeodePriv...()** routines. A specific geode's private data block is expanded only when a valid **GeodePrivWrite()** is performed for the geode. Space is "allocated" in the data blocks of all geodes (loaded or yet-to-be loaded) simultaneously via a call to **GeodePrivAlloc()**. Data that have never been written are returned as all zeros.

Include: geode.h

■ GeodePrivFree()

```
void        GeodePrivFree(
            word   offset,          /* offset returned by GeodePrivAlloc() */
            word   numWords);       /* number of words to free */
```

This routine frees a group of contiguous words from all geodes' private data areas. The space must previously have been allocated with **GeodePrivAlloc()**. Pass the offset to the words as returned by **GeodePrivAlloc()** as well as the number of words to be freed.

Include: geode.h

■ GeodePrivRead()

```
void        GeodePrivRead(
            GeodeHandle         gh,       /* handle of owner of private data */
            word                offset,   /* offset returned by
                                        * GeodePrivAlloc() */
            word                numWords, /* number of words to read */
            word                * dest);  /* pointer to buffer into which data
                                        * will be copied */
```

This routine reads a number of words from the geode's private data area. Pass the following:

gh The geode handle of the owner of the private data to be read.

offset The offset to the private data as returned by **GeodePrivAlloc()**.

numWords The number of words to read.

dest A pointer to a locked or fixed buffer into which the words should be read. It must be at least *numWords* words long.

Include: geode.h

■ GeodePrivWrite()

```
void        GeodePrivWrite(
            GeodeHandle         gh,        /* handle of owner of private data */
            word                offset,    /* offset returned by
                                            * GeodePrivAlloc() */
            word                numWords,  /* number of words to be written */
            word                * src);    /* buffer containing data */
```

This routine writes a number of words into a geode's private data area. The area being written must have been allocated previously with **GeodePrivAlloc()**. Pass the following:

gh The geode handle of the owner of the private data space.

offset The offset to begin writing to, as returned by **GeodePrivAlloc()**.

numWords The number of words to be written. This should be no more than had been previously allocated.

src A pointer to the locked or fixed buffer containing the data to be written.

Include: geode.h

■ GeodeSetDefaultDriver()

```
void        GeodeSetDefaultDriver(
            GeodeDefaultDriverType type,  /* type of default driver to set */
            GeodeHandle            gh);   /* driver to set as the default */
```

This routine sets the default driver for the indicated driver type. Pass the type of default driver in *type* and the handle of the driver in *gh*. The type must be a value of **GeodeDefaultDriverType**, which includes GDDT_FILE_SYSTEM (0), GDDT_KEYBOARD (2), GDDT_MOUSE (4), GDDT_VIDEO (6), GDDT_MEMORY_VIDEO (8), GDDT_POWER_MANAGEMENT(10), GDDT_TASK(12).

Include: driver.h

■ GeodeSetUIData()

```
void      GeodeSetUIData(
          GeodeHandle         gh,
          word                data)
```

■ GeodeUseDriver()

```
GeodeHandle GeodeUseDriver(
          const  char         * name,      /* file name of driver to load */
          word                protoMajor,  /* expected major protocol */
          word                protoMinor,  /* expected minor protocol */
          GeodeLoadError      * err);      /* pointer to returned error */
```

This routine dynamically loads a driver geode given the driver's file name. It returns the geode handle of the driver if successful; if unsuccessful, it returns an error code of type **GeodeLoadError** pointed to by *err*. Pass this routine the following:

name	A pointer to the driver's null-terminated full path and file name.
protoMajor	The expected major protocol of the driver. If zero, any protocol is acceptable.
protoMinor	The expected minor protocol of the driver.
err	A pointer to a **GeodeLoadError** in which any error values will be returned.

Tips and Tricks: It is much easier to automatically load the drivers you need by noting them in your geode parameters file.

Be Sure To: If you use **GeodeUseDriver()** to dynamically load a driver, you must also use **GeodeFreeDriver()** to free it when you are done using it.

Include: driver.h

■ GeodeUseLibrary()

```
GeodeHandle GeodeUseLibrary(
          const char *        name,         /* file name of library to load */
          word                protoMajor,   /* expected major protocol */
          word                protoMinor,   /* expected minor protocol */
          GeodeLoadError *    err);         /* pointer to returned error */
```

This routine dynamically loads a library geode when given the library's file name. (The library must be in the thread's working directory.) It returns the geode handle of the loaded library if successful; if unsuccessful, it returns an error code (**GeodeLoadError**) pointed to by *err*. Pass this routine the following parameters:

name	A pointer to the library's null-terminated file name.
protoMajor	The expected major protocol of the library. If zero, any protocol is acceptable.
protoMinor	The expected minor protocol of the library.
err	A pointer to a **GeodeLoadError** which will contain any returned error values.

Be Sure To: If you dynamically load a library with **GeodeUseLibrary()**, you must manually free it when finished, with **GeodeFreeLibrary()**.

Include: library.h

■ GeoFree()

```
void      * GeoFree(
          void                * blockPtr,   /* address of memory to free */
          GeodeHandle         geodeHan);    /* owner of block to be used */
```

The routine **malloc()** can free only memory in the malloc-block belonging to the calling geode. If you want to free memory in another geode's malloc-block, call **GeoFree()**. Passing a null **GeodeHandle** will make **GeoMalloc()** act on memory in the calling geode's malloc-block.

Include: geode.h

Warnings: Pass exactly the same address as the one returned to you when you allocated the memory. If you pass a different address, **GeoFree()** will take unpredictable actions, including possibly erasing other memory or crashing the system.

See Also: free()

■ GeoMalloc()

```
void        * GeoMalloc(
            size_t              blockSize,    /* # of bytes to allocate*/
            GeodeHandle         geodeHan,     /* Owner of block to be used */
            word                zeroInit);    /* Zero-initialize memory? */
```

The routine **malloc()** automatically allocates memory in the malloc-block belonging to the calling geode. It does not zero-initialize the memory. If you want to zero-initialize the memory, or want to allocate it in another geode's malloc-block, call **GeoMalloc()**. Pass *true* (i.e., non-zero) in *zeroInit* to zero-initialize the memory.

Passing a null **GeodeHandle** will make **GeoMalloc()** allocate the memory in the calling geode's malloc-block. If "zeroInit" is true, the memory will be initialized to null bytes; otherwise, the memory will be left uninitialized.

Include: geode.h

Warnings: All memory allocated with **malloc()** is freed when GEOS shuts down.

See Also: malloc()

■ GeoReAlloc()

```
void        * GeoReAlloc(
            void                * blockPtr,   /* address of memory to resize */
            size_t              newSize,      /* New size in bytes */
            GeodeHandle         geodeHan);    /* Owner of block to be used */
```

The routine **realloc()** can resize only memory in the malloc-block belonging to the calling geode. If you want to resize memory in another geode's malloc-block, call **GeoReAlloc()**. Passing a null **GeodeHandle** will make **GeoReAlloc()** act on memory in the calling geode's malloc-block.

If the block is resized larger, the new memory will not be zero-initialized. Resizing a block smaller will never fail. If **GeoReAlloc()** fails, it will return a null pointer (zero). If you pass a *newSize* of zero, the passed block pointer is freed and the return pointer is a null pointer.

Include: geode.h

Warnings: Pass exactly the same address as the one returned to you when you allocated the memory. If you pass a different address, **GeoReAlloc()** will take unpredictable actions, including possibly erasing other memory or crashing the system.

See Also: realloc()

■ GrApplyRotation()

```
void      GrApplyRotation(
          GStateHandle        gstate,   /* GState to alter */
          WWFixedAsDWord      angle);   /* degrees counterclockwise */
```

Apply a rotation to the GState's transformation matrix.

Include: **graphics.h**

■ GrApplyScale()

```
void      GrApplyScale(
          GStateHandle        gstate,   /* GState to alter */
          WWFixedAsDWord      xScale,   /* new x scale factor */
          WWFixedAsDWord      yScale);  /* new y scale factor */
```

Apply a scale factor to the GState's transformation matrix.

Include: **graphics.h**

■ GrApplyTransform()

```
void      GrApplyTransform(
          GStateHandle        gstate,   /* GState to draw to */
          const TransMatrix   *tm);     /* transformation matrix to apply */
```

Apply a transformation, expressed as a transformation matrix, to a GState's coordinate system.

Include: **graphics.h**

■ GrApplyTranslation()

```
void      GrApplyTranslation(
          GStateHandle        gstate,   /* GState to alter */
          WWFixedAsDWord      xTrans,   /* translation in x */
          WWFixedAsDWord      yTrans);  /* translation in y */
```

Apply a translation to the GState.

Include: **graphics.h**

■ GrApplyTranslationDWord()

```
void      GrApplyTranslationDWord(
          GStateHandle        gstate,   /* GState to alter */
          sdword              xTrans,   /* extended translation in x */
          sdword              yTrans);  /* extended translation in y */
```

Apply a 32-bit integer extended translation to the GState.

Include: **graphics.h**

■ GrBeginPath()

```
void      GrBeginPath(
          GStateHandle        gstate,   /* GState to alter */
          PathCombineType     params);  /* path parameters */
```

Starts or alters the path associated with a GState. All graphics operations that are executed until **GrEndPath()** is called become part of the path.

Depending on the value of the *params* field, the new path may replace the old path, or may be combined with the old path by intersection or union.

Include: **graphics.h**

■ GrBeginUpdate()

```
void      GrBeginUpdate(
          GStateHandle gstate);  /* GState to draw to */
```

Called by an application to signal that it is about to begin updating the exposed region. This routine is normally called as part of a MSG_META_EXPOSED handler. Blanks out the invalid area.

Include: win.h

■ GrBitBlt()

```
void      GrBitBlt(
          GStateHandle        gstate,      /* GState to draw to */
          sword               sourceX,     /* original x origin */
          sword               sourceY,     /* original y origin */
          sword               destX,       /* new x origin */
          sword               destY,       /* new y origin */
          word                width,       /* width of area */
          word                height,      /* height of area */
          BLTMode             mode);       /* draw mode (see below) */
```

Transfer a bit-boundary block of pixels between two locations in video memory. This routine is useful for animation and other applications which involve moving a drawing around the screen.

Structures:

```
typedef enum /* word */ {
    BLTM_COPY,   /* Leave source region alone */
    BLTM_MOVE,   /* Clear & invalidate source rect */
    BLTM_CLEAR   /* Clear source rectangle */
} BLTMode;
```

Include: **graphics.h**

■ GrBrushPolyline()

```
void    GrBrushPolyline(
        GStateHandle        gstate,   /* GState to draw to */
        const Point         * points, /* array of Point structures to draw */
        word                numPoints,/* number of points in array */
        word                brushH,   /* brush height */
        word                brushW);  /* brush width */
```

Draw a brushed connected polyline. Note that this routine ignores the GState's line width, and instead uses a brush height and width, measured in pixels.

Include: **graphics.h**

■ GrCharMetrics()

```
dword   GrCharMetrics(
        GStatehandle        gstate,   /* GState to get metrics for */
        GCM_info            info,     /* information to return */
        word                ch);      /* character of type Chars */
```

Returns metric information for a single character of a font. This information is used to determine the drawing bounds for a character. To find out how wide a character is (how much space to leave for it if drawing a line of text character-by-character), use **GrCharWidth()** instead.

Structures:

```
typedef enum {
    GCMI_MIN_X,         /* return = value << 16 */
    GCMI_MIN_X_ROUNDED, /* return = value */
    GCMI_MIN_Y,         /* return = value << 16 */
    GCMI_MIN_Y_ROUNDED, /* return = value << 16 */
    GCMI_MAX_X,         /* return = value << 16 */
```

```
        GCMI_MAX_X_ROUNDED, /* return = value << 16 */
        GCMI_MAX_Y,         /* return = value << 16 */
        GCMI_MAX_Y_ROUNDED  /* return = value << 16 */
    } GCM_Info;
```

See Also: GrCharWidth()

Include: font.h

■ GrCharWidth()

```
dword     GrCharWidth( /* Returns width << 16 */
          GStateHandle        gstate,    /* GState to query */
          word                ch);       /* character of type Chars */
```

Return the width of a single character. Note that this routine does not take into account track kerning, pairwise kerning, space padding, or other attributes that apply to multiple characters.

Include: **graphics.h**

■ GrCheckFontAvailID()

```
FontID    GrCheckFontAvailID(
          FontEnumFlags       flags,
          word                family,
          FontID              id);
```

See if font (identified by ID) exists.

Include: **graphics.h**

■ GrCheckFontAvailName()

```
FontID    GrCheckFontAvailName(
          FontEnumFlags       flags,
          word                family,
          const char          * name);
```

See if font (identified by name) exists.

Include: **graphics.h**

■ GrClearBitmap()

```
void      GrClearBitmap(
          GStateHandle gstate);      /* GState to affect */
```

Clear out the content of a bitmap. Note that the part of the bitmap actually cleared depends on the bitmap mode. For the normal mode, the data portion of the bitmap is cleared. If the bitmap is in BM_EDIT_MASK mode, then the mask is cleared and the data portion is left alone.

Include: **graphics.h**

■ GrCloseSubPath()

```
void      GrCloseSubPath(
          GStateHandle gstate);  /* GState to affect */
```

Geometrically closes the currently open path segment. Note that you must still call **GrEndPath()** to end the path definition.

Include: **graphics.h**

■ GrComment()

```
void      GrComment(
          GStateHandle        gstate,   /* GState to affect */
          const void          * data,   /* comment string */
          word                size);    /* Size of data, in bytes */
```

Write a comment out to a graphics string.

Include: **graphics.h**

■ GrCopyGString()

```
GSRetType GrCopyGString(
          GStateHandle        source,   /* GState from which to get GString */
          GStateHandle        dest,     /* GState to which to copy GString */
          GSControl           flags);   /* flags for the copy */
```

Copy all or part of a Graphics String. The **GSControl** record can have the following flags:

```
GSC_ONE            /* just do one element */
GSC_MISC           /* return on MISC opcode */
GSC_LABEL          /* return on GR_LABEL opcode */
GSC_ESCAPE         /* return on GR_ESCAPE opcode */
GSC_NEW_PAGE       /* return when we get to a NEW_PAGE */
GSC_XFORM          /* return on TRANSFORMATIONopcode */
GSC_OUTPUT:        /* return on OUTPUT opcode */
GSC_ATTR           /* return on ATTRIBUTE opcode */
GSC_PATH           /* return on PATH opcode */
```

The return value can be any one of **GSRetType**, a byte-size field:

```
GSRT_COMPLETE
GSRT_ONE
GSRT_MISC
GSRT_LABEL
```

```
GSRT_ESCAPE
GSRT_NEW_PAGE
GSRT_XFORM
GSRT_OUTPUT
GSRT_ATTR
GSRT_PATH
GSRT_FAULT
```

Include: **gstring.h**

■ GrCreateBitmap()

```
VMBlockHandle GrCreateBitmap(
          BMFormat            initFormat,  /* color fomat of bitmap */
          word                initWidth,   /* initial width of bitmap */
          word                initHeight,  /* initial height of bitmap */
          VMFileHandle        vmFile,      /* VM file to hold bitmap's data*/
          optr                exposureOD,  /* optr to get MSG_META_EXPOSED */
          GStateHandle        * bmgs);     /* Draws to this GState
                                            * will draw to the bitmap */
```

This routine allocates memory for a bitmap and creates an off-screen window in which to hold the bitmap. This routine takes the following arguments:

initFormat	The depth of the bitmap's color.
initWidth	Bitmap's width.
initHeight	Bitmap's height.
vmFile	File to hold the bitmap data; the routine will allocate a block within this file.
exposureOD	Object which will receive the "exposed" message when the bitmap's window is invalidated. If this argument is zero, then no exposed message will be sent. Remember that an off-screen window is created to house the bitmap. When this window is first created, it will be invalid, and it is conceivable that later actions could cause it to become invalid again. On these occasions, the object specified by this argument will receive a MSG_META_EXPOSED.
bmgs	The GStateHandle pointed to by this argument can start out as null; the routine will use it to return the GState by which the bitmap can be drawn to. Any graphics routines which draw to this returned GState will be carried out upon the bitmap.

The routine returns a **VMBlockHandle**, the handle of the block within the passed VM file which contains the bitmap's data. The block will be set up as the first block of a HugeArray. Its header area will be filled with the following:

Complex Bitmap Header
: This is a **CBitmap** structure which contains some basic information about the bitmap.

Editing Mode
: These flags can change how the bitmap is being edited.

Device Information Block
: This internal structure contains information about and used by the video driver. (Don't worry that you don't know the size of this structure; remember that the CBitmap structure contains the offsets of the bitmap and palette data areas.)

Pallette Information (optional)
: If the bitmap has its own pallette, this is where the palette data will be stored; it will consist of an array of 3-byte entries. Depending on how many colors the bitmap supports, there may be 16 or 256 entries in this array.

The bitmap's raw data is in the VM block, but outside of the header area.

Include: **graphics.h**

■ GrCreateGString()

```
GStateHandle GrCreateGString(
          Handle              han,      /* memory, stream, or VM file handle */
          GStringType         hanType,  /* type of handle in han parameter */
          word                * gsBlock);   /* returned for GST_MEMORY and
                                         * GST_VMEM types only */
```

Open a graphics string and start redirecting graphics orders to the string. The hanType parameter must be GST_MEMORY, GST_STREAM, or GST_VMEM.

Include: **gstring.h**

■ GrCreatePalette()

```
word      GrCreatePalette( /* Returns # of entries in color table
                             * or 0 for monochrome or 24-bit */
          GStateHandle gstate);
```

Create a color mapping table and associate it with the current window. Initialize the table entries to the default palette for the device.

Include: **graphics.h**

■ GrCreateState()

```
GStateHandle GrCreateState(
          WindowHandle win);     /* Window in which GState will be active */
```

Create a graphics state (GState) block containg default GState information.

If zero is passed, then the GState created will have no window associated with it.

Include: **graphics.h**

■ GrDeleteGStringElement()

```
void      GrDeleteGStringElement(
          GStateHandle        gstate,   /* GState containing GString */
          word                count);   /* number of elements to delete */
```

Delete a range of GString elements from the GString in the passed GState.

Include: **graphics.h**

■ GrDestroyBitmap()

```
void      GrDestroyBitmap(
          GStateHandle        gstate,   /* GState containing bitmap */
          BMDestroy           flags);   /* flags for removing data */
```

Free the bitmap and disassociate it with its window. Depending on the passed flag, the bitmap's data may be freed or preserved. Thus, it is possible to remove the GString used to edit the bitmap while maintaining the bitmap in a drawable state.

Structures:

```
typedef ByteEnum BMDestroy;
/*     BMD_KILL_DATA,
       BMD_LEAVE_DATA */
```

Include: **graphics.h**

■ GrDestroyGString()

```
void      GrDestroyGString(
          Handle              gstring,  /* Handle of GString */
          GStateHandle        gstate,   /* NULL, or handle of another
                                         * gstate to free*/
          GStringKillType     type);    /* Kill type for data removal */
```

Destroys a GString. Depending on the **GStringKillType** argument, this either constitutes removing the GState from the GString data; or freeing both the GState and the GString's data. If you have been drawing the GString to a GState, you should pass the GState's handle as *gstate*, and this routine will do some cleaning up.

Structures:

```
typedef ByteEnum GStringKillType;
/*     GSKT_KILL_DATA,
       GSKT_LEAVE_DATA */
```

Include: **gstring.h**

■ GrDestroyPalette()

```
void      GrDestroyPalette(
          GStateHandle gstate);  /* GState of palette to destroy */
```

Free any custom palette associated with the current window.

Include: **graphics.h**

■ GrDestroyState()

```
void      GrDestroyState(
          GStateHandle gstate);  /* GState to be destroyed */
```

Free a graphics state block.

Include: **graphics.h**

■ GrDrawArc()

```
void      GrDrawArc(
          GStateHandle        gstate,      /* GState to draw to */
          sword               left,        /* bounds of box outlining arc */
          sword               top,
          sword               right,
          sword               bottom,
          word                startAngle,  /* angles in degrees
          word                endAngle,     * counter-clockwise */
          ArcCloseType        arcType);    /* how the arc is closed */
```

Draw an arc along the ellipse that is specified by a bounding box, from the starting angle to the ending angle.

Include: **graphics.h**

■ GrDrawArc3Point()

```
void      GrDrawArc3Point(
          GStateHandle       gstate,   /* GState to draw to */
          const ThreePointArcParams *params);
```

Draw a circular arc, given three points along the arc; both endpoints and any other point on the arc.

Include: **graphics.h**

■ GrDrawArc3PointTo()

```
void      GrDrawArc3PointTo(
          GStateHandle       gstate,   /* GState to draw to */
          const ThreePointArcToParams *params);
```

As **GrDrawArc3Point()**, above, except that the current position is automatically used as one of the endpoints.

Include: **graphics.h**

■ GrDrawBitmap()

```
void      GrDrawBitmap(
          GStateHandle       gstate,          /* GState to draw to */
          sword              x,               /* x starting point */
          sword              y,               /* y starting point */
          const  Bitmap      * bm,            /* pointer to the bitmap */
          Bitmap * _pascal (*callback) (Bitmap *bm));/* NULL for no callback */
```

Draw a bitmap. Note that if the bitmap takes up a great deal of memory, it is necessary to manage its memory when drawing. If the bitmap resides in a **HugeArray** (true of any bitmap created using **GrCreateBitmap()**), then calling **GrDrawHugeBitmap()** will automatically take care of memory management. Otherwise, you may wish to provide a suitable callback routine. This routine should be declared _pascal and is passed a pointer into the passed bitmap and is expected to return a pointer to the next slice. This allows the bitmap to be drawn in horizontal bands, or swaths.

Include: **graphics.h**

■ GrDrawBitmapAtCP()

```
void      GrDrawBitmapAtCP(
          GStateHandle        gstate,          /* GState to draw to */
          const  Bitmap       * bm,            /* pointer to the bitmap */
          Bitmap * (*callback) (Bitmap *bm));  /* NULL for no callback */
```

This routine is the same as **GrDrawBitmap()**, above, except that the bitmap is drawn at the current position.

Include: **graphics.h**

■ GrDrawChar()

```
void      GrDrawChar(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* x position at which to draw */
          sword               y,        /* y position at which to draw */
          word                ch);      /* character of type Chars */
```

Draw a character at the given position with the current text drawing attributes.

Include: **graphics.h**

■ GrDrawCharAtCP()

```
void      GrDrawCharAtCP(
          GStateHandle        gstate,   /* GState to draw to */
          word                ch);      /* character of type Chars */
```

Draw a character at the current position with the current text drawing attributes.

Include: **graphics.h**

■ GrDrawCurve()

```
void      GrDrawCurve(
          GStateHandle        gstate,      /* GState to draw to */
          const  Point        *points);    /* array of four Points */
```

Draw a Bezier curve.

Include: **graphics.h**

■ GrDrawCurveTo()

```
void      GrDrawCurveTo(
          GStateHandle        gstate,      /* GState to draw to */
          const  Point        *points); /* array of three Points */
```

Draw a Bezier curve, using the current postion as the first point.

Include: **graphics.h**

■ GrDrawEllipse()

```
void      GrDrawEllipse(
          GStateHandle        gstate,   /* GState to draw to */
          sword               left,     /* bounding box bounds */
          sword               top,
          sword               right,
          sword               bottom);
```

Draw an ellipse, defined by its bounding box.

Include: **graphics.h**

■ GrDrawGString()

```
GSRetType GrDrawGString(
          GStateHandle        gstate,          /* GState to draw to */
          Handle              gstringToDraw,   /* GString to draw */
          sword               x,               /* point at which to draw */
          sword               y,
          GSControl           flags,           /* GSControl record */
          GStringElement      * lastElement);  /* pointer to empty structure */
```

Draw a graphics string. The passed control flag allows drawing to stop upon encountering certain kinds of drawing elements. If this causes the drawing to stop in mid-string, then the routine will provide a pointer to the next **GStringElement** to be played.

- ◆ You must provide a GState to draw to. You may wish to call **GrSaveState()** on the GState before drawing the GString (and call **GrRestoreState()** afterwards). If you will draw anything else to this GState after the GString, you must call **GrDestroyGString()** on the GString, and pass this GState's handle as the gstate argument so that **GrDestroyGString()** can clean up the GState.
- ◆ You must provide a GString to draw. The GString must be properly loaded (probably by means of **GrLoadGString()**).
- ◆ You can provide a pair of coordinates at which to draw the GString. The graphics system will translate the coordinate system by these

coordinates before carrying out the graphics commands stored in the GString.

- ◆ You can provide a **GSControl** argument which requests that the system stop drawing the GString when it encounters a certain type of GString element. If the GString interpreter encounters one of these elements, it will immediately stop drawing. The GString will remember where it stopped drawing. If you call **GrDrawGString()** with that same GString, it will continue drawing where you left off.
- ◆ You must provide a pointer to an empty **GStringElement** structure. **GrDrawGString()** will return a value here when it is finished drawing. If the GString has stopped drawing partway through due to a passed **GSControl**, the returned **GStringElement** value will tell you what sort of command was responsible for halting drawing. For instance, if you had instructed **GrDrawGString()** to halt on an 'output' element (GrDraw...() or GrFill...() commands), then when **GrDrawGString()** returns, you would check the value returned to see what sort of output element was present.

Include: **gstring.h**

■ GrDrawGStringAtCP()

```
GSRetType GrDrawGStringAtCP(
        GStateHandle        gstate,          /* GState to draw to */
        GStringeHandle      gstringToDraw,   /* GString to draw */
        GSControl           flags,           /* GSControl flags */
        GStringElement      * lastElement);  /* last element to draw */
```

Draw a graphics string as **GrDrawGString()**, above, except that drawing takes place at the current position.

- ◆ You must provide a GState to draw to. You may wish to call **GrSaveState()** on the GState before drawing the GString (and call **GrRestoreState()** afterwards). If you will draw anything else to this GState after the GString, you must call **GrDestroyGString()** on the GString, and pass this GState's handle as the gstate argument so that **GrDestroyGString()** can clean up the GState.
- ◆ You must provide a GString to draw. The GString must be properly loaded (probably by means of **GrLoadGString()**).
- ◆ You can provide a **GSControl** argument which requests that the system stop drawing the GString when it encounters a certain type of GString element. If the GString interpreter encounters one of these elements, it will immediately stop drawing. The GString will remember where it stopped drawing. If you call **GrDrawGString()** with that same GString, it will continue drawing where you left off.

- You must provide a pointer to an empty **GStringElement** structure. **GrDrawGString()** will return a value here when it is finished drawing. If the GString has stopped drawing partway through due to a passed **GSControl**, the returned **GStringElement** value will tell you what sort of command was responsible for halting drawing. For instance, if you had instructed **GrDrawGString()** to halt on an 'output' element (GrDraw...() or GrFill...() commands), then when **GrDrawGString()** returns, you would check the value returned to see what sort of output element was present.

Include: **gstring.h**

■ GrDrawHLine()

```
void      GrDrawHLine(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x1,       /* first horizontal coordinate */
          sword               y,        /* vertical position of line */
          sword               x2);      /* second horizontal coordinate */
```

Draw a horizontal line.

Include: **graphics.h**

■ GrDrawHLineTo()

```
void      GrDrawHLineTo(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x);       /* ending horizontal coordinate */
```

Draw a horizontal line starting from the current position.

Include: **graphics.h**

■ GrDrawHugeBitmap()

```
void      GrDrawHugeBitmap(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x         /* Point at which to draw */
          sword               y,
          VMFileHandle        vmFile,   /* VM File holding HugeArray */
          VMBlockHandle       vmBlk);   /* VM block of HugeArray */
```

Draw a bitmap that resides in a HugeArray.

Include: **graphics.h**

See Also: **GrDrawBitmap()** , **GrDrawHugeBitmapAtCP()**, **GrDrawHugeImage()**

■ GrDrawHugeBitmapAtCP()

```
void      GrDrawHugeBitmapAtCP(
          GStateHandle        gstate,   /* GState to draw to */
          VMFileHandle        vmFile,   /* VM file containing HugeArray */
          VMBlockHandle       vmBlk);   /* VM block containing HugeArray */
```

As **GrDrawHugeBitmap()**, above, except that the bitmap is drawn at the current position.

Include: **graphics.h**

See Also: **GrDrawBitmapAtCP()**, **GrDrawHugeBitmap()**

■ GrDrawHugeImage()

```
void      GrDrawHugeImage(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x         /* point at which to draw */
          sword               y,
          ImageFlags          flags,
          VMFileHandle        vmFile,   /* VM file holding HugeArray */
          VMBlockHandle       vmBlk);   /* VM block holding HugeArray */
```

Draw a bitmap that resides in a **HugeArray**. Note that the bitmap will be drawn on an assumption of one device pixel per bitmap pixel. The bitmap will not draw rotated or scaled. Depending on the value of the flags argument, the bitmap may be expanded so that a square of device pixels displays each bitmap pixel.

Structures:

```
          typedef ByteFlags ImageFlags;
          /* The following flags be be combined using | and &:
                 IF_IGNORE_MASK,
                 IF_BORDER
          * The flags should be combined with one ImageBitSize:
                 IF_BITSIZE */
          #define IBS_1 0
          #define IBS_2 1
          #define IBS_4 2
          #define IBS_8 3
          #define IBS_16 4
```

Include: **graphics.h**

See Also: **GrDrawImage()** , **GrDrawHugeBitmapAtCP()**

■ GrDrawImage()

```
void      GrDrawImage(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x         /* point at which to draw */
          sword               y,
          ImageFlags          flags,
          const Bitmap        * bm);    /* pointer to bitmap */
```

Draw a bitmap. Note that the bitmap will be drawn on an assumption of one device pixel per bitmap pixel. The bitmap will not draw rotated or scaled. Depending on the value of the flags argument, the bitmap may be expanded so that a square of device pixels displays each bitmap pixel.

Structures:

```
typedef ByteFlags ImageFlags;
/* The following flags be be combined using | and &:
       IF_IGNORE_MASK,
       IF_BORDER
* The flags should be combined with one ImageBitSize:
       IF_BITSIZE */
#define IBS_1 0
#define IBS_2 1
#define IBS_4 2
#define IBS_8 3
#define IBS_16 4
```

Include: **graphics.h**

See Also: **GrDrawHugeImage()** , **GrDrawBitmap()**

■ GrDrawLine()

```
void      GrDrawLine(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x1,       /* First coordinate of line */
          sword               y1,
          sword               x2,       /* Second coordinate of line */
          sword               y2);
```

Draw a line.

Include: **graphics.h**

See Also: **GrDrawLineTo()**, **GrDrawHLine()**, **GrDrawVLine()**

■ GrDrawLineTo()

```
void      GrDrawLineTo(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* Second coordinate of line */
          sword               y);
```

Draw a line starting from the current position.

Include: **graphics.h**

See Also: **GrDrawLine()**, **GrDrawHLineTo()**, **GrDrawVLineTo()**

■ GrDrawPath()

```
void      GrDrawPath(
          GStateHandle gstate);  /* GState to draw to */
```

Draws the stroked version of the current path, using the current graphic line attributes.

Include: **graphics.h**

■ GrDrawPoint()

```
void      GrDrawPoint(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* Coordinates of point to draw */
          sword               y);
```

Draw a pixel.

Include: **graphics.h**

■ GrDrawPointAtCP()

```
void      GrDrawPointAtCP(
          GStateHandle gstate);  /* GState to draw to */
```

Draw a pixel.

Include: **graphics.h**

■ GrDrawPolygon()

```
void      GrDrawPolygon(
          GStateHandle        gstate,      /* GState to draw to */
          const  Point        * points,    /* array of points in polygon */
          word                numPoints);  /* number of points in array */
```

Draws a connected polygon.

Include: **graphics.h**

■ GrDrawPolyline()

```
void      GrDrawPolyline(
          GStateHandle        gstate,      /* GState to draw to */
          const  Point        * points,    /* array of points in polyline */
          word                numPoints);  /* number of points in array */
```

Draws a simple polyline.

Include: **graphics.h**

■ GrDrawRect()

```
void      GrDrawRect(
          GStateHandle        gstate,   /* GState to draw to */
          sword               left,     /* bounds of rectangle to draw */
          sword               top,
          sword               right,
          sword               bottom);
```

Draws the outline of a rectangle.

Include: **graphics.h**

■ GrDrawRectTo()

```
void      GrDrawRectTo(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* opposite corner of rectangle */
          sword               y);
```

Draws the outline of a rectangle, with one corner defined by the current position.

Include: **graphics.h**

■ GrDrawRegion()

```
void      GrDrawRegion(
          GStateHandle        gstate,   /* GState to draw to */
          sword               xPos,     /* Position at which to draw */
          sword               yPos,
          const  Region       * reg,    /* Region definition */
          sword               param0,   /* value to use with
                                     * parameterized coordinates */
          sword               param)1;  /* value to use with
                                     * parameterized coordinates */
```

Draw a region. The area will be rendered filled with the GState's area attributes.

Include: **graphics.h**

■ GrDrawRegionAtCP()

```
void      GrDrawRegionAtCP(
          GStateHandle        gstate,   /* GState to draw to */
          const  Region       * reg,    /* region definition */
          sword  param0,      /* Value to use with parameterized coordinates */
          sword  param1,      /* Value to use with parameterized coordinates */
          sword  param2,      /* Value to use with parameterized coordinates */
          sword  param)3;     /* Value to use with parameterized coordinates */
```

Draw a region at the current pen position. The area will be rendered filled with the GState's area attributes.

Include: **graphics.h**

■ GrDrawRelArc3PointTo()

```
void      GrDrawRelArc3PointTo(
          const ThreePointRelArcToParams *params);
```

Draw a circular arc relative to the current point given two additional points: the other endpoint and any other point on the arc, both described in relative coordinates.

Include: **graphics.h**

■ GrDrawRelLineTo()

```
void     GrDrawRelLineTo(
         GStateHandle        gstate,   /* GState to draw to */
         WWFixedAsDWord      x,        /* horizontal offset of second point */
         WWFixedAsDWord      y);       /* vertical offset of second point */
```

Draw a line from the current pen position, given a displacement from the current pen position to draw to.

Include: **graphics.h**

■ GrDrawRoundRect()

```
void     GrDrawRoundRect(
         GStateHandle        gstate,          /* GState to draw to */
         sword               left,            /* bounds of rectangle */
         sword               top,
         sword               right,
         sword               bottom,
         word                cornerRadius);   /* radius of corner rounding */
```

Draw the outline of a rounded rectangle.

Include: **graphics.h**

■ GrDrawRoundRectTo()

```
void     GrDrawRoundRectTo(
         GStateHandle        gstate,          /* GState to draw to */
         sword               x,               /* opposite corner of bounds */
         sword               y,
         word                cornerRadius);   /* radius of corner rounding */
```

Draw the outline of a rounded rectangle, where one corner of the bounding rectangle is the current position.

Include: **graphics.h**

■ GrDrawSpline()

```
void     GrDrawSpline(
         GStateHandle        gstate,      /* GState to draw to */
         const  Point        * points,    /* array of points */
         word                numPoints,); /* number of points in array */
```

Draw a Bezier spline.

Include: **graphics.h**

See Also: **GrDrawCurve()**

■ GrDrawSplineTo()

```
void      GrDrawSplineTo(
          GStateHandle        gstate,      /* GState to draw to */
          const  Point        *points,     /* array of points */
          word                numPoints);  /* number of points in array */
```

Draw a Bezier spline, using the current position as one endpoint.

Include: **graphics.h**

See Also: **GrDrawCurveTo()**

■ GrDrawText()

```
void      GrDrawText(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* point at which to draw */
          sword               y,
          const  Chars        * str,    /* pointer to character string */
          word                size);    /* length of string */
```

Draw a string of text. The string is represented as an array of characters. Note that the text will be drawn using the GState's font drawing attributes and that this routine does not accept any style run arguments.

If the passed *size* argument is zero, the string is assumed to be null-terminated.

Include: **graphics.h**

■ GrDrawTextAtCP()

```
void      GrDrawTextAtCP(
          GStateHandle        gstate,   /* GState to draw to */
          const  Chars        * str,    /* pointer to character string */
          word                size);    /* length of string */
```

As **GrDrawText()**, above, except that the text is drawn at the current position.

If the passed *size* argument is zero, the string is assumed to be null-terminated.

Include: **graphics.h**

■ GrDrawVLine()

```
void      GrDrawVLine(
          GStateHandle        gstate,   /* GState to draw to */
          sword               x,        /* horizontal position of line */
          sword               y1,       /* first vertical coordinate */
          sword               y2);      /* second vertical coordinate */
```

Draw a vertical line.

Include: **graphics.h**

■ GrDrawVLineTo()

```
void      GrDrawVLine(
          GStateHandle        gstate,   /* GState to draw to */
          sword               y);       /* second vertical position */
```

Draw a vertical line starting from the current position.

Include: **graphics.h**

■ GrEditBitmap()

```
GStateHandle GrEditBitmap(
          VMFileHandle        vmFile,      /* VM file of bitmap */
          VMBlockHandle       vmBlock,     /* VM block of bitmap */
          optr                exposureOD); /* optr to get MSG_META_EXPOSED */
```

This routine attaches a GState to the passed bitmap so that new drawings may be be sent to the bitmap.

Include: **graphics.h**

■ GrEditGString()

```
GStateHandle GrEditGString(
          Handle vmFile,      /* VM file containing the GString */
          word   vmBlock);    /* VM block containing the GString */
```

This routine takes the location of a GString data block stored in a VM file. It will associate a GState with this GString data and returns the handle of this GState. Any graphics commands issued using this GStateHandle will be appended to the GString.

Include: **graphics.h**

■ GrEndGString()

```
GStringErrorType GrEndGString(
          GStateHandle gstate);  /* GState to draw to */
```

Finish the definition of a graphics string.

Structures:

```
typedef enum {
GSET_NO_ERROR,
GSET_DISK_FULL
} GStringErrorType;
```

Include: **graphics.h**

■ GrEndPath()

```
void      GrEndPath(
          GStateHandle gstate);  /* GState to draw to */
```

Finish definition of a path. Further graphics commands will draw to the display, as normal.

Include: **graphics.h**

■ GrEndUpdate()

```
void      GrEndUpdate(
          GStateHandle gstate);  /* GState to draw to */
```

Unlocks window from an update.

Include: **win.h**

■ GrEnumFonts()

```
word      GrEnumFonts( /* Return value = number of fonts found */
          FontEnumStruct      * buffer, /* buffer for returned values */
          word                size,     /* number of structures to return */
          FontEnumFlags       flags,    /* FontEnumFlags */
          word                family);  /* FontFamily */
```

Generate a list of available fonts. The font information includes both the font's ID and a string name.

Structures:

```
typedef struct {
FontID FES_ID;
char FES_name[FID_NAME_LEN];
} FontEnumStruct;
```

Include: **font.h**

■ GrEscape()

```
void     GrEscape(
         GStateHandle        gstate,   /* GState to draw to */
         word                code,     /* escape code */
         const  void         * data,   /* pointer to the data */
         word                size);    /* Size of data, in bytes */
```

Write an escape code to a graphics string.

Include: **graphics.h**

■ GrFillArc()

```
void     GrFillArc(
         GStateHandle        gstate,      /* GState to draw to */
         sword               left,        /* bounding rectangle */
         sword               top,
         sword               right,
         sword               bottom,
         word                startAngle,  /* angles in degrees
         word                endAngle      * counter-clockwise */
         ArcCloseType        closeType);  /* OPEN, CHORD, or PIE */
```

Fill an elliptical arc. The arc is defined by the bounding rectangle of the base ellipse and two angles. Depending on how the arc is closed, this will result in either a wedge or a chord fill.

Include: **graphics.h**

■ GrFillArc3Point()

```
void     GrFillArc3Point(
         GStateHandle        gstate,      /* GState to draw to */
         const ThreePointParams *params);
```

Fill an arc. Depending on how the arc is closed, this will result in either a wedge or a chord fill. The arc is defined in terms of its endpoints and one other point, all of which must lie on the arc.

Include: **graphics.h**

■ GrFillArc3PointTo()

```
void        GrFillArc3PointTo(
            GStateHandle        gstate,      /* GState to draw to */
            const ThreePointArcParams *params);
```

As **GrFillArc3Point()**, above, except that one endpoint of the arc is defined by the current position.

Include: **graphics.h**

■ GrFillBitmap()

```
void        GrFillBitmap (
            GStateHandle        gstate,      /* GState to draw to */
            sword               x,           /* point at which to draw */
            sword               y,
            const Bitmap        * bm,        /* pointer to bitmap */
            Bitmap * (*callback) (Bitmap *bm));
```

Fill a monochrome bitmap with the current area attributes. The arguments to this routine are the same as those for **GrDrawBitmap()**.

Include: **graphics.h**

■ GrFillBitmapAtCP()

```
void        GrFillBitmapAtCP (
            GStateHandle        gstate,          /* GState to draw to */
            const Bitmap        * bm,            /* pointer to bitmap */
            Bitmap * (*callback) (Bitmap *bm));
```

Fill a monochrome bitmap with the current area attributes. The bitmap will be drawn at the current position. The arguments to this routine are the same as those for **GrDrawBitmapAtCP()**.

Include: **graphics.h**

■ GrFillEllipse()

```
void        GrFillEllipse(
            GStateHandle        gstate,      /* GState to draw to */
            sword               left,        /* Bounds of bounding rectangle */
            sword               top,
            sword               right,
            sword               bottom);
```

Draw a filled ellipse. The ellipse’s dimensions are defined by its bounding box.

Include: **graphics.h**

■ GrFillPath()

```
void      GrFillPath(
          GStateHandle        gstate,      /* GState to draw to */
          RegionFillRule      rule);       /* ODD_EVEN or WINDING */
```

Fill an area whose outline is defined by the GState's path.

Include: **graphics.h**

■ GrFillPolygon()

```
void      GrFillPolygon(
          GStateHandle        gstate,          /* GState to draw to */
          RegionFillRule      windingRule,     /* ODD_EVEN or WINDING */
          const  Point        * points,        /* array of points in polygon */
          word                numPoints);      /* number of points in array */
```

fill polygon. The polygon is defined by the passed array of points.

Include: **graphics.h**

■ GrFillRect()

```
void      GrFillRect(
          GStateHandle        gstate,      /* GState to draw to */
          sword               left,        /* bounds of rectangle */
          sword               top,
          sword               right,
          sword               bottom);
```

Draw a filled rectangle.

Include: **graphics.h**

■ GrFillRectTo()

```
void      GrFillRectTo(
          GStateHandle        gstate,      /* GState to draw to */
          sword               x,           /* opposite corner of rectangle */
          sword               y);
```

Draw a filled rectangle. The current position will define one of the corners.

Include: **graphics.h**

■ GrFillRoundRect()

```
void      GrFillRoundRect(
          GStateHandle        gstate,          /* GState to draw to */
          sword               left,            /* bounds of rectangle */
          sword               top,
          sword               right,
          sword               bottom
          word                cornerRadius);   /* radius of corner rounding */
```

Draw a filled rounded rectangle.

Include: **graphics.h**

■ GrFillRoundRectTo()

```
void      GrFillRoundRectTo(
          GStateHandle        gstate,      /* GState to draw to */
          sword               x,           /* opposite corner of rectangle */
          sword               y
          word                cornerRadius);   /* radius of corner roundings */
```

Draw a filled rounded rectangle, using the current position to define one corner of the bounding rectangle.

Include: **graphics.h**

■ GrFindNearestPointsize()

```
Boolean   GrFindNearestPointsize( /* If false, then FontID invalid */
          FontID              id,              /* fond ID */
          dword               sizeSHL16,       /* point size */
          TextStyle           styles,          /* style */
          TextStyle           * styleFound,       /* buffer for style */
          dword               * sizeFoundSHL16);  /* buffer for size */
```

Find the nearest available point size for a font. If the font passed in *id* exists, then *styleFound* will point to the styles available and *sizeFoundSHL16* will point to the nearest point size to that passed. If the font is not found, the return valued will be *true*.

Include: **font.h**

■ GrFontMetrics()

```
dword     GrFontMetrics(
          GStateHandle        gstate,      /* subject GState */
          GFM_info            info);       /* Type of information to return */
```

Get metrics information about a font. It returns the requested information based on the *info* parameter.

Structures:

```
typedef enum /* word */ {
      GFMI_HEIGHT,               /* return = val << 16 */
      GFMI_MEAN,                 /* return = val << 16 */
      GFMI_DESCENT,              /* return = val << 16 */
      GFMI_BASELINE,             /* return = val << 16 */
      GFMI_LEADING,              /* return = val << 16 */
      GFMI_AVERAGE_WIDTH,        /* return = val << 16 */
      GFMI_ASCENT,               /* return = val << 16 */
      GFMI_MAX_WIDTH,            /* return = val << 16 */
      GFMI_MAX_ADJUSTED_HEIGHT,  /* return = val << 16 */
      GFMI_UNDER_POS,            /* return = val << 16 */
      GFMI_UNDER_THICKNESS,      /* return = val << 16 */
      GFMI_ABOVE_BOX,            /* return = val << 16 */
      GFMI_ACCENT,               /* return = val << 16 */
      GFMI_MANUFACTURER,         /* return = val */
      GFMI_KERN_COUNT,           /* return = Char */
      GFMI_FIRST_CHAR,           /* return = Char */
      GFMI_LAST_CHAR,            /* return = FontMaker */
      GFMI_DEFAULT_CHAR,         /* return = Char */
      GFMI_STRIKE_POS,           /* return = Char */
      GFMI_BELOW_BOX,            /* return = Char */
      GFMI_HEIGHT_ROUNDED        /* return = Char */
      GFMI_DESCENT_ROUNDED,      /* return = Char */
      GFMI_BASELINE_ROUNDED,     /* return = Char */
      GFMI_LEADING_ROUNDED,      /* return = Char */
      GFMI_AVERAGE_WIDTH_ROUNDED,/* return = Char */
      GFMI_ASCENT_ROUNDED,       /* return = Char */
      GFMI_MAX_WIDTH_ROUNDED,    /* return = Char */
      GFMI_MAX_ADJUSTED_HEIGHT_ROUNDED, /* ret = Char */
      GFMI_UNDER_POS_ROUNDED,    /* return = Char */
      GFMI_UNDER_THICKNESS_ROUNDED, /* return = Char */
      GFMI_ABOVE_BOX_ROUNDED,    /* return = Char */
      GFMI_ACCENT_ROUNDED=,      /* return = Char */
      GFMI_STRIKE_POS_ROUNDED,   /* return = Char */
      GFMI_BELOW_BOX_ROUNDED     /* return = Char */
} GFM_info;
```

Include: **font.h**

■ GrGetAreaColor()

```
RGBColorAsDWord GrGetAreaColor(
          GStateHandle gstate);  /* GState of which to get color */
```

Get the color which is being used to fill areas.

Include: **graphics.h**

■ GrGetAreaColorMap()

```
ColorMapMode GrGetAreaColorMap(
          GStateHandle gstate);     /* GState of which to get area color map */
```

Get the mapping mode used for filling areas with unavailable colors.

Include: **graphics.h**

■ GrGetAreaMask()

```
SysDrawMask GrGetAreaMask(
          GStateHandle        gstate,   /* GState of which to get mask */
          DrawMask            * dm);    /* buffer for returned mask */
```

Get the draw mask used when filling areas. The *dm* argument should point to a buffer capable of holding at least eight bytes to get the bit-pattern of the mask; otherwise *dm* should be NULL. The returned buffer is the 8x8 bit pattern: each byte represents a row of the pattern, and the bytes are ordered from top row to bottom.

Include: **graphics.h**

■ GrGetAreaPattern()

```
GraphicPattern GrGetAreaPattern(
          GStateHandle        gstate,          /* GState of area pattern */
          const MemHandle     * customPattern, /* pointer to handle of block for
                                              * returned custom pattern */
          word                * customSize);   /* pointer to size of returned
                                              * buffer */
```

Get the area pattern used when filling areas.

Include: **graphics.h**

■ GrGetBitmap()

```
MemHandle GrGetBitmap(
          GStateHandle        gstate,          /* GState containing bitmap */
          sword               x,               /* bitmap origin */
          sword               y,
          word                width,           /* bitmap width and height */
          word                height,
          XYSize              * sizeCopied);   /* buffer for returned size */
```

Dump an area of the display to a bitmap. The handle of a block containing the bitmap is returned; the *sizeCopied* pointer points to the actual size of the bitmap successfully copied.

Include: **graphics.h**

■ GrGetBitmapMode()

```
BitmapMode GrGetBitmapMode(
         GStateHandle gstate);     /* GState containing bitmap */
```

Get mode bits for an editable bitmap.

Include: **graphics.h**

■ GrGetBitmapRes()

```
XYValueAsDWord GrGetBitmapRes(
         const Bitmap       * bm);       /* pointer to the bitmap */
```

Get the resolution of a bitmap.

Include: **graphics.h**

■ GrGetBitmapSize()

```
XYValueAsDWord GrGetBitmapSize(
         const  Bitmap      * bm);       /* pointer to the bitmap */
```

Get the dimensions, in points, of a bitmap.

Include: **graphics.h**

■ GrGetClipRegion()

```
MemHandle GrGetClipRegion(
         GStateHandle       gstate,   /* subject GState */
         RegionFillRule     rule);    /* ODD_EVEN or WINDING */
```

Get the current clip region. A null handle (zero) will be returned if no clip paths are se for the GState.

Include: **graphics.h**

■ GrGetCurPos()

```
XYValueAsDWord GrGetCurPos(
         GStateHandle gstate);     /* subject GState */
```

Get the current pen position.

Include: **graphics.h**

■ GrGetCurPosWWFixed()

```
void GrGetCurPosWWFixed(
        GStateHandle gstate,      /* subject GState */
        PointWWFixed *cp);        /* buffer in which to return cur. pos. */
```

Get the current pen position.

Include: **graphics.h**

■ GrGetDefFontID()

```
FontID  GrGetDefFontID(
        dword  * sizeSHL16);    /* pointer to buffer for returned size */
```

Get the system default font (including size).

Include: **font.h**

■ GrGetFont()

```
FontID  GrGetFont(
        GStateHandle        gstate,      /* subject GState */
        WWFixedAsDWord      * pointSize);/* pointer to buffer for
                                       * returned point size */
```

Get the passed GState's current font, including point size.

Include: **graphics.h**

■ GrGetFontName()

```
FontID  GrGetFontName(
        FontID              id,       /* ID of font */
        const char          * name);  /* buffer for returned name string */
```

Get the string name of a font. Note that if the returned **FontID** is zero, then the font was not found. The name string buffer should be a least FID_NAME_LEN in size.

Include: **font.h**

■ GrGetFontWeight()

```
FontWeight GrGetFontWeight(
        GStateHandle gstate);     /* GState containing the font */
```

Get the current font weight set for the passed GState.

Include: **font.h**

■ GrGetFontWidth()

```
FontWidth GrGetFontWidth(
          GStateHandle gstate);      /* GState containing the font */
```

Get the current font width set for the passed GState.

Include: **font.h**

■ GrGetGStringBounds()

```
void      GrGetGStringBounds(
          GStringHandle       source,       /* GString to be checked */
          GStateHandle        dest,         /* handle of GState to use */
          GSControl           flags,        /* GSControl flags */
          Rectangle           * bounds);    /* returned bounds of GState */
```

This routine returns the coordinate bounds of the *source* GString drawn at the current position in the GString. The *dest* GState will be used if passed; to have no GState restrictions, pass a null handle. The bounds of the smallest containing rectangle will be returned in the structure pointed to by *bounds*.

Include: **gstring.h**

■ GrGetGStringBoundsDWord

```
void      GrGetGStringBoundsDWord(
          Handle              gstring,      /* GString to be checked */
          GStateHandle        gstate,       /* handle of GState to use */
          GSControl           flags,        /* GSControl flags */
          RectDWord           * bounds);    /* returned bounds of GState */
```

This routine behaves as **GrGetGStringBounds()**, but has been alterred to work with 32-bit graphics spaces.

This routine returns the coordinate bounds of aGString drawn at the current position in the GString. The *gstate* GState will be used if passed; to have no GState restrictions, pass a null handle. The bounds of the smallest containing rectangle will be returned in the structure pointed to by *bounds*.

Include: **gstring.h**

■ GrGetGStringElement()

```
GStringElement GrGetGStringElement(
          GStateHandle        gstate,         /* handle of GString's GState */
          void                * buffer,       /* pointer to return buffer */
          word                bufSize,        /* size of return buffer */
          word                * elementSize,  /* size of GString element */
          void                ** pointerAfterData);  /* pointer to pointer to
                                                  * next element in GString */
```

Extract the next element from a graphics string. The opcode is returned explicitly. The routine's data can be returned in a buffer.

Include: **gstring.h**

■ GrGetInfo()

```
void      GrGetInfo(
          GStateHandle        gstate,   /* GState to get information about */
          GrInfoTypes         type,     /* type of information to get */
          void                * data);  /* buffer for returned information */
```

Get the private data, window handle, or pen position associated with the GState.

Structures:

```
typedef enum {
    GIT_PRIVATE_DATA,
    GIT_WINDOW,
    GIT_PEN_POS
} GrInfoType
```

Include: **graphics.h**

■ GrGetLineColor()

```
RGBColorAsDWord GrGetLineColor(
          GStateHandle gstate);     /* subject GState */
```

Get the color used when drawing lines.

Include: **graphics.h**

■ GrGetLineColorMap()

```
ColorMapMode GrGetLineColorMap(
          GStateHandle gstate);     /* subject GState */
```

Get the mode used when drawing lines in an unavailable color.

Include: **graphics.h**

■ GrGetLineEnd()

```
LineEnd  GrGetLineEnd(
         GStateHandle gstate);     /* subject GState */
```

Get the end used when drawing lines.

Include: **graphics.h**

■ GrGetLineJoin()

```
LineJoin GrGetLineJoin(
         GStateHandle gstate);     /* subject GState */
```

Get the join used when drawing corners.

Include: **graphics.h**

■ GrGetLineMask()

```
SysDrawMask GrGetLineMask(
         GStateHandle        gstate,   /* subject GState */
         DrawMask            * dm);    /* buffer for returned custom mask */
```

Get the drawing mask used when drawing lines. The *dm* argument should point to a buffer capable of holding at least eight bytes to get the bit-pattern of the mask; otherwise *dm* should be NULL. The returned buffer is the 8x8 bit pattern: each byte represents a row of the pattern, and the bytes are ordered from top row to bottom.

Include: **graphics.h**

■ GrGetLineStyle()

```
LineStyle GrGetLineStyle(
         GStateHandle gstate);     /* subject GState */
```

Get the style, or “dottedness,” used when drawing lines.

Include: **graphics.h**

■ GrGetLineWidth()

```
WWFixedAsDWord GrGetLineWidth(
         GStateHandle gstate);     /* subject GState */
```

Get the current line width.

Include: **graphics.h**

■ GrGetMaskBounds()

```
void      GrGetMaskBounds(
          GStateHandle       gstate,      /* subject GState */
          Rectangle          * bounds);   /* buffer for returned bounds */
```

Get the 16-bit bounds of the current clip rectangle.

Include: **graphics.h**

■ GrGetMaskBoundsDWord()

```
void      GrGetMaskBoundsDWord(
          GStateHandle       gstate,      /* subject GState */
          RectDWord          * bounds);   /* buffer for returned bounds */
```

Get the 16-bit bounds of the current clip rectangle, accurate to a fraction of a point.

Include: **graphics.h**

■ GrGetMiterLimit()

```
WWFixedAsDWord GrGetMiterLimit(
          GStateHandle gstate);           /* subject GState */
```

Get the miter limit to use when drawing mitered corners.

Include: **graphics.h**

■ GrGetMixMode()

```
MixMode GrGetMixMode(
          GStateHandle gstate);         /* subject GState */
```

Get the current mixing mode.

Include: **graphics.h**

■ GrGetPalette()

```
MemHandle GrGetPalette(
          GStateHandle       gstate,          /* subject GState */
          GetPalType         flag,            /* GPT_ACTIVE, GPT_CUSTOM, or
                                             * GPT_DEFAULT */
          word               * numEntries);   /* number of entries in block */
```

Return all or part of the window's color lookup table. This routine returns the handle of a block containing all the returned palette entries.

Include: **graphics.h**

■ GrGetPath()

```
MemHandle GrGetPath(
        GStateHandle        gstate,       /* subject GState */
        GetPathType         ptype);       /* Which path to retrieve */
```

Returns handle to block containing path data. This handle may be passed to **GrSetPath()**. Either the current path, the clipping path, or the window clipping path may be retrieved.

Include: **graphics.h**

■ GrGetPathBounds()

```
Boolean  GrGetPathBounds(
        GStateHandle        gstate,       /* subject GState */
        GetPathType         ptype,
        Rectangle           * bounds);    /* buffer for returned bounds */
```

Returns the rectangular bounds that encompass the current path as it would be filled. A *true* return value indicates an error occurred or there was no path for the GState.

Include: **graphics.h**

■ GrGetPathBoundsDWord()

```
Boolean  GrGetPathBoundsDWord(
        GStateHandle        gstate,       /* subject GState */
        GetPathType         ptype,
        RectDWord           * bounds);    /* buffer for returned bounds */
```

Returns the rectangular bounds that encompass the current path as it would be filled. A *true* return value indicates an error occurred or there was no path for the GState.

Include: **graphics.h**

■ GrGetPathPoints()

```
MemHandle GrGetPathPoints(
        GStateHandle        gstate,          /* subject GState */
        word                resolution);     /* dots per inch */
```

Returns a series of points that fall along the current path. The returned points are in document coordinates.

Include: **graphics.h**

■ GrGetPathRegion()

```
MemHandle GrGetPathRegion(
        GStateHandle        gstate,        /* subject GState */
        RegionFillRule      rule);         /* ODD_EVEN or WINDING */
```

Get the region enclosed by a path.

Include: **graphics.h**

■ GrGetPoint()

```
RGBColorAsDWord GrGetPoint(
        GStateHandle        gstate,    /* subject GState */
        sword               x,         /* coordinates of pixel */
        sword               y);
```

Get the color of the pixel corresponding to the specified coordinates.

Include: **graphics.h**

■ GrGetPtrRegBounds()

```
word    GrGetPtrRegBounds( /* Returns size of Region data struct. */
        const  Region       * reg,         /* pointer to region */
        Rectangle           * bounds);     /* returned bounds of region */
```

Get the bounds of the passed region.

Include: **graphics.h**

■ GrGetSubscriptAttr()

```
ScriptAttrAsWord GrGetSubscriptAttr(
        GStateHandle gstate);         /* subject GState */
```

Get the GState's subscript drawing attributes. The high byte of the return value is the percentage of the font size for the subscript; the low byte is the percentage of the font size from the top at which the character gets drawn.

Include: **font.h**

■ GrGetSuperscriptAttr()

```
ScriptAttrAsWord GrGetSuperscriptAttr(
        GStateHandle gstate);         /* subject GState */
```

Get the GState's superscript drawing attributes. The high byte of the return value is the percentage of the font size for the superscript; the low byte is the percentage of the font size from the bottom at which the character gets drawn.

Include: **font.h**

■ GrGetTextBounds()

```
Boolean  GrGetTextBounds(
         GStateHandle        gstate,   /* subject GState */
         word                xpos,     /* position where text would be drawn */
         word                ypos,
         const char          * str,    /* text string */
         word                count,    /* max number of characters to check */
         Rectangle           * bounds);   /* returned bounding rectangle */
```

Get the bounds required to draw the passed text. If the passed *size* argument is zero, the string is assumed to be null-terminated.

Include: **graphics.h**

■ GrGetTextColor()

```
RGBColorAsDWord GrGetTextColor(
         GStateHandle gstate);     /* subject GState */
```

Get the color used when drawing text.

Include: **graphics.h**

■ GrGetTextColorMap()

```
ColorMapMode GrGetTextColorMap(
         GStateHandle gstate);     /* subject GState */
```

Get the mode used when drawing text in an unavailable color.

Include: **graphics.h**

■ GrGetTextMask()

```
SystemDrawMask GrGetTextMask(
         GStateHandle        gstate,   /* subject GState */
         DrawMask            * dm);    /* returned custom mask, if any */
```

Get the draw mask used when drawing text.The *dm* argument should point to a buffer capable of holding at least eight bytes to get the bit-pattern of the mask; otherwise *dm* should be NULL. The returned buffer is the 8x8 bit pattern: each byte represents a row of the pattern, and the bytes are ordered from top row to bottom.

Include: **graphics.h**

■ GrGetTextMode()

```
TextMode GrGetTextMode(
         GStateHandle gstate);     /* subject GState */
```

Get the text mode, including information about the vertical offset used when drawing text.

Include: **graphics.h**

■ GrGetTextPattern()

```
GraphicPattern GrGetTextPattern(
         GStateHandle        gstate,            /* subject GState */
         const MemHandle     * customPattern,    /* pointer to returned handle
                                               * of block containing the
                                               * returned pattern */
         word                * customSize);      /* size of returned block */
```

Get the graphics pattern used when drawing text.

Include: **graphics.h**

■ GrGetTextSpacePad()

```
WWFixedAsDWord GrGetTextSpacePad(
         GStateHandle gstate);     /* subject GState */
```

Get the space pad used when drawing strings of text.

Include: **graphics.h**

■ GrGetTextStyle()

```
TextStyle GrGetTextStyle(
         GStateHandle gstate);         /* subject GState */
```

Get the style used when drawing text.

Include: **graphics.h**

■ GrGetTrackKern()

```
word     GrGetTrackKern(
         GStateHandle gstate);         /* subject GState */
```

Get the track kerning used when drawing strings of text.

Include: **graphics.h**

■ GrGetTransform()

```
void      GrGetTransform(
          GStateHandle        gstate,   /* subject GState */
          TransMatrix         * tm);    /* pointer to returned TransMatrix */
```

Get the current coordinate transformation, expressed as a matrix.

Include: **graphics.h**

■ GrGetWinBounds()

```
void      GrGetWinBounds(
          GStateHandle        gstate,      /* subject GState */
          Rectangle           * bounds);   /* returned window bounds */
```

Get the bounds of the GState's associated window.

Include: **graphics.h**

■ GrGetWinBoundsDWord()

```
void      GrGetWinBoundsDWord(
          GStateHandle        gstate,      /* subject GState */
          RectDWord           * bounds);   /* returned window bounds */
```

Get the bounds of the GState's associated window, accurate to a fraction of a point.

Include: **graphics.h**

■ GrGetWinHandle()

```
WindowHandle GrGetWinHandle(
          GStateHandle gstate);         /* subject GState */
```

Get the handle of the GState's associated window.

Include: **graphics.h**

■ GrGrabExclusive()

```
GStateHandle GrGrabExclusive(
          GeodeHandle         videoDriver, /* NULL for default */
          GStateHandle        gstate);     /* subject GState */
```

Start drawing exclusively to a video driver.

Include: **graphics.h**

■ GrInitDefaultTransform()

```
void      GrInitDefaultTransform(
          GStateHandle gstate);        /* subject GState */
```

Initialize the GState's default transformation to hold hte value of the current transformation.

Include: **graphics.h**

■ GrInvalRect()

```
void      GrInvalRect(
          GStateHandle        gstate,      /* subject GState */
          sword               left,        /* bounds to be invalidated */
          sword               top,
          sword               right,
          sword               bottom);
```

Invalidate the passed rectangular area. This area will be redrawn.

Include: **graphics.h**

■ GrInvalRectDWord()

```
void      GrInvalRectDWord(
          GStateHandle        gstate,      /* subject GState */
          const  RectDWord    * bounds);   /* bounds to be invalidated */
```

Invalidate the passed rectangular area. This area will be redrawn.

Include: **graphics.h**

■ GrLabel()

```
void      GrLabel(
          GStringHandle       gstate,   /* subject GState */
          word                label);   /* label to write to GString */
```

Write the passed label into the passed GString.

Include: **gstring.h**

■ GrLoadGString()

```
GStringHandle GrLoadGString(
          Handle              han,         /* handle of GString source */
          GStringType         hanType,     /* handle type */
          word                vmBlock);    /* if VM file, handle of VM block */
```

Load a graphics string from a file. Used with stream, VM, and pointer addressed GStrings.

Structures:

```
typedef ByteEnum GStringType;
/*     GST_MEMORY,
       GST_STREAM,
       GST_VMEM,
       GST_PTR,
       GST_PATH      */
```

Include: **gstring.h**

■ GrMapColorIndex()

```
RGBColorAsDWord GrMapColorIndex(
          GStateHandle        gstate,   /* GState to use for mapping */
          Color               c);       /* source color to be mapped */
```

Map a color index to its RGB equivalent using the color mapping scheme of the passed GState.

Include: **graphics.h**

■ GrMapColorRGB()

```
RGBColorAsDWord GrMapColorRGB(
          GStateHandle        gstate,   /* GState to use for mapping */
          word                red,      /* RGB values to map */
          word                green,
          word                blue);
```

Map an RGB color to an index.

Include: **graphics.h**

■ GrMoveReg()

```
void      GrMoveReg(
          Region * reg,       /* pointer to region */
          sword  xOffset,     /* amount to shift horizontally */
          sword  yOffset);    /* amount to shift vertically */
```

Moves a region a given amount. Note that this operation affects only the region's data structure. The region must be redrawn or used in some other way for the changes to have any visible effect.

Include: **graphics.h**

■ GrMoveTo()

```
void      GrMoveTo(
          GStateHandle        gstate,   /* subject GState */
          sword               x,        /* new absolute pen position */
          sword               y);
```

Change the pen position.

Include: **graphics.h**

■ GrMulDWFixed()

```
void      GrMulDWFixed(
          const  DWFixed      * i,          /* first number */
          const  DWFixed      * j,          /* second number */
          DWFixed             * result);    /* pointer to returned result */
```

Multiply two fixed point numbers.

Include: **graphics.h**

■ GrMulWWFixed()

```
WWFixedAsDWord GrMulWWFixed(
          WWFixedAsDWord i,      /* first number */
          WWFixedAsDWord j);     /* second number */
```

Multiply two fixed point numbers.

Include: **graphics.h**

■ GrNewPage()

```
void      GrNewPage(
          GStateHandle        gstate,
          PageEndCommand      pageEndCommand);
```

Begin drawing a new page. Normally used when printing documents.

Include: **graphics.h**

■ GrNullOp()

```
void      GrNullOp(
          GStateHandle gstate);     /* subject GState */
```

Write a null operation element to a GString.

Include: **graphics.h**

■ GrQuickArcSine()

```
WWFixedAsDWord GrQuickArcSine(
          WWFixedAsDWord      deltaYDivDistance,  /* delta y / distance */
          word                origDeltaX);        /* original delta x */
```

Compute a fixed point arcsine. Angles are given in degrees counterclockwise of the positive x axis.

Include: **graphics.h**

■ GrQuickCosine()

```
WWFixedAsDWord GrQuickCosine(
          WWFixedAsDWord angle);     /* angle to cosine */
```

Compute a fixed point cosine. Angles are given in degrees counterclockwise of the positive x axis.

Include: **graphics.h**

■ GrQuickSine()

```
WWFixedAsDWord GrQuickSine(
          WWFixedAsDWord angle);     /* angle to sine */
```

Compute a fixed point sine. Angles are given in degrees counterclockwise of the positive x axis.

Include: **graphics.h**

■ GrQuickTangent()

```
WWFixedAsDWord GrQuickTangent(
          WWFixedAsDWord angle);     /* angle to tangent */
```

Compute a fixed point tangent. Angles are given in degrees counterclockwise of the positive x axis.

Include: **graphics.h**

■ GrReleaseExclusive()

```
void      GrReleaseExclusive( /* TRUE if system had to force a redraw */
          GeodeHandle         videoDriver, /* handle of video driver */
          GStateHandle        gstate,      /* GState that was drawing */
          Rectangle           *bounds);    /* Bounds of aborted drawings */
```

Stop drawing exclusively to a video driver.

Include: **graphics.h**

■ GrRelMoveTo()

```
void      GrRelMoveTo(
          GStateHandle        gstate,   /* subject GState */
          WWFixedAsDWord      x,        /* offsets to new pen position */
          WWFixedAsDWord      y);
```

Change the pen position to coordinate expressed relative to the current position.

Include: **graphics.h**

■ GrRestoreState()

```
void      GrRestoreState(
          GStateHandle gstate);     /* subject GState */
```

Restore the values of a saved GState.

Include: **graphics.h**

■ GrSaveState()

```
void      GrSaveState(
          GStateHandle gstate);     /* subject GState */
```

Save the values of a GState, so that they may be restored by **GrRestoreState()**.

Include: **graphics.h**

■ GrSDivDWFByWWF()

```
void GrSDivDWFByWWF(
          const DWFixed       * dividend,
          const WWFixed       * divisor,
          DWFixed             * quotient)  /* returned value */
```

Divide two fixed point numbers.

Include: **graphics.h**

■ GrSDivWWFixed()

```
WWFixedAsDWord GrSDivWWFixed(
          WWFixedAsDWord dividend,
          WWFixedAsDWord divisor)
```

Divide two fixed point numbers.

Include: **graphics.h**

■ GrSetAreaAttr()

```
void      GrSetAreaAttr(
          GStateHandle        gstate,   /* subject GState */
          const AreaAttr      * aa);    /* AreaAttr structure */
```

Set all of the attributes used when filling areas.

Structures:

```
typedef struct {
      byte                AA_colorFlag;
      RGBValue            AA_color;
      SystemDrawMask      AA_mask;
      ColorMapMode        AA_mapMode;
} AreaAttr;
```

Include: **graphics.h**

■ GrSetAreaColor()

```
void      GrSetAreaColor(
          GStateHandle        gstate,       /* GState to set color for */
          ColorFlag           flag,         /* flag of how to set color */
          word                redOrIndex,   /* color index or red RGB value */
          word                green,        /* green RGB value or zero */
          word                blue);        /* blue RGB value or zero */
```

Set the color to use when filling areas. The flag parameter may be CF_RGB (to set RGB values), CF_INDEX (to set a palette index), CF_GRAY, or CF_SAME.

Include: **graphics.h**

■ GrSetAreaColorMap()

```
void      GrSetAreaColorMap(
          GStateHandle        gstate,       /* subject GState */
          ColorMapMode        colorMap);    /* color mapping mode */
```

Set mode to use when trying to fill an area with an unavailable color.

Include: **graphics.h**

■ GrSetAreaMaskCustom()

```
void      GrSetAreaMaskCustom(
          GStateHandle        gstate,       /* subject GState */
          const  DrawMask     * dm);        /* pointer to new custom mask */
```

Set the drawing mask to use when filling areas.

Include: **graphics.h**

■ GrSetAreaMaskSys()

```
void     GrSetAreaMaskSys(
         GStateHandle        gstate,      /* subject GState */
         SystemDrawMask      sysDM);      /* new system area mask */
```

Set the drawing mask to use when filling areas.

Include: **graphics.h**

■ GrSetAreaPattern()

```
void     GrSetAreaPattern(
         GStateHandle        gstate,      /* subject GState */
         GraphicPattern      pattern);    /* new pattern */
```

Set the graphics pattern to use when filling areas.

Include: **graphics.h**

■ GrSetBitmapMode()

```
void     GrSetBitmapMode(
         GStateHandle        gstate,      /* subject GState */
         word                flags,  /* BM_EDIT_MASK or BM_CLUSTERED_DITHER */
         MemHandle           colorCorr);  /* handle of ColorTransfer */
```

Set the bitmap editing mode. This allows the editing of a bitmap's mask, or turning on clustered dithering.

Include: **graphics.h**

■ GrSetBitmapRes()

```
Boolean  GrSetBitmapRes(
         GStateHandle        gstate,      /* subject GState */
         word                xRes,        /* new resolutions */
         word                yRes);
```

Set a complex bitmap's resolution.

Include: **graphics.h**

■ GrSetClipPath()

```
void     GrSetClipPath(
         GStateHandle        gstate,      /* subject GState */
         PathCombineType     params,      /* how paths should be combined */
         RegionFillRule      rule);       /* ODD_EVEN or WINDING */
```

Restrict the clipping region by intersecting it with the passed path.

Include: **graphics.h**

■ GrSetClipRect()

```
void    GrSetClipRect(
        GStateHandle        gstate,      /* subject GState */
        PathCombineType     flags,       /* how paths should be combined */
        sword               left,        /* bounds of clipping rectangle */
        sword               top,
        sword               right,
        sword               bottom);
```

Restrict the clipping region by intersecting it with the passed rectangle.

Include: **graphics.h**

■ GrSetCustomAreaPattern()

```
void    GrSetCustomAreaPattern(
        GStateHandle        gstate,      /* subject GState */
        GraphicPattern      pattern,     /* new area pattern */
        const void *        patternData, /* pointer to pattern data */
        word                patternSize); /* size of pattern data buffer */
```

Set the graphics pattern to use when filling areas.

Include: **graphics.h**

■ GrSetCustomTextPattern()

```
void    GrSetCustomTextPattern(
        GStateHandle        gstate,          /* subject GState */
        GraphicPattern      pattern,         /* new pattern */
        const void          * patternData);  /* pointer to pattern data */
```

Set the graphic pattern used when drawing text.

Include: **graphics.h**

■ GrSetDefaultTransform()

```
void    GrSetDefaultTransform(
        GStateHandle gstate);      /* subject GState */
```

Replace the current coordinate transformation with the default transformation.

Include: **graphics.h**

■ GrSetFont()

```
void      GrSetFont(
          GStateHandle        gstate,      /* subject GState */
          FontID              id,          /* new font ID */
          WWFixedAsDWord      pointSize);  /* new point size */
```

Set the font to use when drawing text.

Include: **graphics.h**

■ GrSetFontWeight()

```
void      GrSetFontWeight(
          GStateHandle        gstate,      /* subject GState */
          FontWeight          weight);     /* new font weight */
```

Set the font weight to use when drawing text.

Include: **font.h**

■ GrSetFontWidth()

```
void      GrSetFontWidth(
          GStateHandle        gstate,      /* subject GState */
          FontWidth           width);      /* new font width */
```

Set the font width to use when drawing text.

Include: **font.h**

■ GrSetGStringBounds()

```
void      GrSetGStringBounds(
          Handle              gstate,      /* GState or GString handle */
          sword               left,        /* new bounds of GString */
          sword               top,
          sword               right,
          sword               bottom);
```

Optimization routine which allows you to set bounds values for a GString. This bounds information will be returned by **GrGetGStringBounds()** whenever that routine is called upon the affected GString.

Include: **graphics.h**

■ GrSetGStringPos()

```
void      GrSetGStringPos(
          GStateHandle        gstate,      /* subject GState */
          GStringSetPosType   type,        /* how to set position */
          word                skip);       /* number of elements to skip */
```

Set a graphics strings' "playing position." Using this routine, it is possible to draw only selected elements of a GString.

Structures:

```
typedef ByteEnum GStringSetPosType;
/*     GSSPT_SKIP,
       GSSPT_RELATIVE,
       GSSPT_BEGINNING,
       GSSPT_END    */
```

Include: **gstring.h**

■ GrSetLineAttr()

```
void      GrSetLineAttr(
          GStateHandle        gstate,   /* subject GState */
          const LineAttr      * la);    /* new line attributes */
```

Set all attributes to use when drawing lines and corners.

Include: **graphics.h**

■ GrSetLineColor()

```
void      GrSetLineColor(
          GStateHandle        gstate,      /* subject GState */
          ColorFlag           flag,        /* color flag */
          word                redOrIndex,  /* new index or red RGB value */
          word                green,       /* new green RGB value or zero */
          word                blue);       /* new blue RGB value or zero */
```

Set the color to use when drawing lines.

Include: **graphics.h**

■ GrSetLineColorMap()

```
void      GrSetLineColorMap(
          GStateHandle gstate,      /* subject GState */
          ColorMapMode colorMap);   /* new color map mode for lines */
```

Set the mode to use when trying to draw lines in an unavailable color.

Include: **graphics.h**

■ GrSetLineEnd()

```
void        GrSetLineEnd(
            GStateHandle        gstate,       /* subject GState */
            LineEnd             end);         /* new line end specification */
```

Set the end to use when drawing lines.

Include: **graphics.h**

■ GrSetLineJoin()

```
void        GrSetLineJoin(
            GStateHandle        gstate,       /* subject GState */
            LineJoin            join);        /* new line join specification */
```

Set the line join to use when drawing corners.

Include: **graphics.h**

■ GrSetLineMaskCustom()

```
void        GrSetLineMaskCustom(
            GStateHandle        gstate,       /* subject GState */
            const  DrawMask     * dm);        /* new line draw mask */
```

Set the drawing mask used when drawing lines.

Include: **graphics.h**

■ GrSetLineMaskSys()

```
void        GrSetLineMaskSys(
            GStateHandle        gstate,       /* subject GState */
            SystemDrawMask      sysDM);       /* the new system line mask */
```

Set the drawing mask used when drawing lines.

Include: **graphics.h**

■ GrSetLineStyle()

```
void        GrSetLineStyle(
            GStateHandle        gstate,           /* subject GState */
            LineStyle           style,            /* new line style */
            word                skipDistance,     /* skip distance to first pair */
            const DashPairArray * dpa,            /* dash definition */
            word                numPairs);        /* number of pairs */
```

Set the style, or “dottedness,” to use when drawing lines.

Include: **graphics.h**

■ GrSetLineWidth()

```
void      GrSetLineWidth(
          GStateHandle        gstate,       /* subject GState */
          WWFixedAsDWord      width);       /* new line width */
```

Set the line width to use when drawing lines.

Include: **graphics.h**

■ GrSetMiterLimit()

```
void      GrSetMiterLimit(
          GStateHandle        gstate,       /* subject GState */
          WWFixedAsDWord      limit);       /* new miter limit */
```

Set the miter limit to use when drawing mitered corners.

Include: **graphics.h**

■ GrSetMixMode()

```
void      GrSetMixMode(
          GStateHandle        gstate,       /* subject GState */
          MixMode             mode);        /* new mix mode */
```

Set the GState's mix mode, used to determine what happens when something is drawn on top of an existing drawing.

Include: **graphics.h**

■ GrSetNullTransform()

```
void      GrSetNullTransform(
          GStateHandle gstate);     /* subject GState */
```

Clear the coordinate transformation. Most applications will actually want to replace the coordinate transformation with the default transformation using **GrSetDefaultTransform()**.

Include: **graphics.h**

■ GrSetPalette()

```
void      GrSetPalette(
          GStateHandle        gstate,       /* subject GState */
          SetPalType          type,         /* SPT_DEFAULT or SPT_CUSTOM */
          const RGBValue      *buffer,      /* array of palette entries */
          word                index,        /* First element to change */
          word                numEntries);  /* number of entries in array */
```

Set one or more entries in a palette, a window's color lookup table.

Include: **graphics.h**

■ GrSetPaletteEntry()

```
void      GrSetPaletteEntry(
          GStateHandle        gstate,       /* subject GState */
          word                index,        /* index in palette to set */
          word                red,          /* new RGB color values for entry */
          word                green,
          word                blue);
```

Set one entry in a palette, a GState's color lookup table.

Include: **graphics.h**

■ GrSetPath()

```
void      GrSetPath(
          GStateHandle        gstate,          /* subject GState */
          MemHandle           pathGString);    /* handle of path's block */
```

Takes the passed GState's path with the path encoded in the block with the passed handle. To get such a handle, call **GrGetPath()**

Include: **graphics.h**

■ GrSetPrivateData()

```
void      GrSetPrivateData(
          GStateHandle        gstate,       /* subject GState */
          word                dataAX,       /* data to set */
          word                dataBX,
          word                dataCX,
          word                dataDX);
```

Set the private data for a GState.

Include: **graphics.h**

■ GrSetStrokePath()

```
void      GrSetStrokePath(
          GStateHandle gstate);      /* subject GState */
```

Replace a GState's path with the path resulting from stroking the original path. Note that this stroked path may be drawn, but may not be used for clipping.

Include: **graphics.h**

■ GrSetSubscriptAttr()

```
void      GrSetSubscriptAttr(
          GStateHandle        gstate,      /* subject GState */
          ScriptAttrAsWord    attrs);      /* new subscript percentages */
```

Get the attributes used when drawing subscript characters.

Include: **font.h**

■ GrSetSuperscriptAttr()

```
void      GrSetSuperscriptAttr(
          GStateHandle        gstate,      /* subject GState */
          ScriptAttrAsWord    attrs);      /* new superscript percentages */
```

Get the attributes used when drawing superscript characters.

Include: **font.h**

■ GrSetTextAttr()

```
void      GrSetTextAttr(
          GStateHandle        gstate,      /* subject GState */
          const  TextAttr     * ta);       /* pointer to text attributes */
```

Set all attributes used when drawing characters and text strings.

Include: **graphics.h**

■ GrSetTextColor()

```
void      GrSetTextColor(
          GStateHandle        gstate,      /* subject GState */
          ColorFlag           flag,        /* color flag */
          word                redOrIndex,  /* palette index or red RGB value */
          word                green,       /* green RGB value or zero */
          word                blue);       /* blue RGB value or zero */
```

Set the color used when drawing text.

Include: **graphics.h**

■ GrSetTextColorMap()

```
void    GrSetTextColorMap(
        GStateHandle        gstate,       /* subject GState */
        ColorMapMode        colorMap);    /* new color mapping mode */
```

Set the mode used when trying to draw text in an unavailable color.

Include: **graphics.h**

■ GrSetTextMaskCustom()

```
void    GrSetTextMaskCustom(
        GStateHandle        gstate,       /* subject GState */
        const  DrawMask     * dm);        /* pointer to custom mask */
```

Set the drawing mask used when drawing text.

Include: **graphics.h**

■ GrSetTextMaskSys()

```
void    GrSetTextMaskSys(
        GStateHandle        gstate,       /* subject GState */
        SystemDrawMask      sysDM);       /* new system draw mask */
```

Set the drawing mask used when drawing text.

Include: **graphics.h**

■ GrSetTextMode()

```
void    GrSetTextMode(
        GStateHandle        gstate,           /* subject GState */
        TextMode            bitsToSet,        /* TextMode flags to set */
        TextMode            bitsToClear);     /* TextMode flags to clear */
```

Set the text mode associated with a GState. Using this routine, it is possible to change the vertical offset used when drawing text.

Include: **graphics.h**

■ GrSetTextPattern()

```
void    GrSetTextPattern(
        GStateHandle        gstate,       /* subject GState */
        GraphicPattern      pattern);     /* new graphic pattern for text */
```

Set the graphic pattern used when drawing text.

Include: **graphics.h**

■ GrSetTextSpacePad()

```
void    GrSetTextSpacePad(
        GStateHandle        gstate,       /* subject GState */
        WWFixedAsDWord      padding);     /* new space padding */
```

Set the space pad used when drawing text strings.

Include: **graphics.h**

■ GrSetTextStyle()

```
void    GrSetTextStyle(
        GStateHandle        gstate,       /* subject GState */
        TextStyle           bitsToSet,    /* TextStyle flags to set */
        TextStyle           bitsToClear);/* TextStyle flags to clear */
```

Set the style to use when drawing text.

Include: **graphics.h**

■ GrSetTrackKern()

```
void    GrSetTrackKern(
        GStateHandle        gstate,       /* subject GState */
        word                tk);          /* degree of track kerning */
```

Set the track kerning to use when drawing text strings.

Include: **graphics.h**

■ GrSetTransform()

```
void    GrSetTransform(
        GStateHandle        gstate,       /* subject GState */
        const TransMatrix   * tm);        /* new transformation matrix */
```

Set the GState's coordinate transformation.

Include: **graphics.h**

■ GrSetVMFile()

```
void    GrSetVMFile(
        GStateHandle        gstate,       /* subject GState */
        VMFileHandle        vmFile);      /* new transformation matrix */
```

Update the VM file associated with a GState (this may apply when working with certain kinds of bitmaps and GStrings).

Include: **graphics.h**

■ GrSetWinClipPath()

```
void      GrSetWinClipPath(
          GStateHandle        gstate,      /* subject GState */
          PathCombineType     params,      /* how paths are combined */
          RegionFillRule      rule);       /* ODD_EVEN or WINDING */
```

Restrict the window's clipping region by intersecting it with the passed path.

Include: **graphics.h**

■ GrSetWinClipRect()

```
void      GrSetWinClipRect(
          GStateHandle        gstate,      /* subject GState */
          PathCombineType     flags,       /* how paths are combined */
          sword               left,        /* new clipping rectangle bounds */
          sword               top,
          sword               right,
          sword               bottom);
```

Restrict the window's clipping region by intersecting it with the passed rectangle.

Include: **graphics.h**

■ GrSqrRootWWFixed()

```
WWFixedAsDWord GrSqrRootWWFixed(
          WWFixedAsDWord i);     /* number to get the square root of */
```

Compute the square root of a fixed point number.

Include: **graphics.h**

■ GrTestPath()

```
Boolean   GrTestPath(
          GStateHandle        gstate,      /* subject GState */
          GetPathType         ptype);      /* Type of path to check for */
```

Determine whether the GState has a path of the specified type.

Include: **graphics.h**

■ GrTestPointInPath()

```
Boolean  GrTestPointInPath(
         GStateHandle       gstate,      /* subject GState */
         word               xPos,        /* point to test */
         word               yPos,
         RegionFillRule     rule);       /* ODD_EVEN or WINDING */
```

Determine whether the passed point falls in the interior of the GState's path.

Include: **graphics.h**

■ GrTestPointInPolygon()

```
Boolean  GrTestPointInPolygon(
         GStateHandle       gstate,      /* subject GState */
         RegionFillRule     rule,        /* ODD_EVEN or WINDING */
         Point              * list,      /* array of points in polygon */
         word               numPoints,   /* number of points in array */
         sword              xCoord,      /* coordinates of point to test */
         sword              yCoord);
```

Determine whether the passed point lies in the interior of the passed polygon.

Include: **graphics.h**

■ GrTestPointInReg()

```
Boolean  GrTestPointInReg(
         const  Region      * reg,       /* pointer to region */
         sword              x,           /* coordinates of point to test */
         sword              y,
         Rectangle          *boundingRect);  /* returned bounding rectangle,
                                              * if point in region */
```

Determine whether a point lies within the passed region. If the point is not in the region, the return value is *true*.

Include: **graphics.h**

■ GrTestRectInReg()

```
TestRectReturnType GrTestRectInReg(
         const Region       * reg     /* pointer to region */
         sword              left,     /* bounds of rectangle to be tested */
         sword              top,
         sword              right,
         sword              bottom);
```

Determine whether a rectangle lies within the clip region.

Routines

Structures:

```
typedef ByteEnum TestRectReturnType;
      TRRT_OUT,     /* rectangle completely out of region */
      TRRT_PARTIAL, /* rectangle partially in region */
      TRRT_IN       /* rectangle completely in region */
```

Include: **graphics.h**

■ GrTextWidth()

```
word      GrTextWidth(
          GStateHandle        gstate,       /* subject GState */
          const  Chars        * str,        /* text string to check */
          word                size);        /* maximum number of
                                          * characters to check */
```

Compute the space the passed text string would require in a line of text. Use **GrGetTextBounds()** to determine the area necessary to render the text.

Include: **graphics.h**

■ GrTextWidthWWFixed()

```
WWFixedAsDWord GrTextWidthWWFixed( /* returns width << 16 */
          GStateHandle        gstate,       /* subject GState */
          const  Chars        * str,        /* text string to check */
          word                size)         /* maximum number of
                                          * characters to check */
```

Compute the spacing the passed text string would require in a line of text, accurate to a fraction of a point. Use **GrGetTextBounds()** to determine the area necessary to render the text.

Include: **graphics.h**

■ GrTransform()

```
XYValueAsDWord GrTransform(
          GStateHandle        gstate,       /* subject GState */
          sword               xCoord,       /* coordinates to transform */
          sword               yCoord);
```

Apply the device’s transformation to the passed point.

Include: **graphics.h**

■ GrTransformDWFixed()

```
void      GrTransformDWFixed(
          GStateHandle        gstate,      /* subject GState */
          PointDWFixed        * coord);    /* coordinates to transform */
```

Apply the device's transformation to the passed point.

Include: **graphics.h**

■ GrTransformDWord()

```
void      GrTransformDWord(
          GStateHandle        gstate,      /* subject GState */
          sdword              xCoord,      /* coordinates to transform */
          sdword              yCoord,
          PointDWord          * deviceCoordinates);
                              /* pointer to returned devide coordinates */
```

Apply the device's transormation to the passed point.

Include: **graphics.h**

■ GrTransformWWFixed()

```
void      GrTransformWWFixed(
          GStateHandle        gstate,      /* subject GState */
          WWFixedAsDWord      xPos,        /* coordinates to transform */
          WWFixedAsDWord      yPos,
          PointWWFixed        * deviceCoordinates);
                                /* pointer to returned devide coordinates */
```

Apply the device's transormation to the passed point.

Include: **graphics.h**

■ GrUDivWWFixed()

```
WWFixedAsDWord GrUDivWWFixed(
          WWFixedAsDWord      dividend,
          WWFixedAsDWord      divisor);
```

Compute an unsigned division of two fixed point numbers.

Include: **graphics.h**

■ GrUntransform()

```
XYValueAsDWord GrUnTransformCoord(
          GStateHandle        gstate,      /* subject GState */
          sword               xCoord,      /* coordinates to untransform */
          sword               yCoord);
```

Apply the reverse of the device's transformation to the passed point.

Include: **graphics.h**

■ GrUntransformDWFixed()

```
void      GrUnTransCoordDWFixed(
          GStateHandle        gstate,      /* subject GState */
          PointDWFixed        * coord);    /* coordinates to untransform */
```

Apply the reverse of the device's transformation to the passed point.

Include: **graphics.h**

■ GrUntransformDWord()

```
void      GrUnTransformExtCoord(
          GStateHandle        gstate,      /* subject GState */
          sdword              xCoord,      /* coordinates to untransform */
          sdword              yCoord,
          PointDWord          * documentCoordinates);
                                 /* pointer to returned devide coordinates *
```

Apply the reverse of the device's transformation to the passed point.

Include: **graphics.h**

■ GrUntransformWWFixed()

```
void      GrUnTransCoordWWFixed(
          GStateHandle        gstate,      /* subject GState */
          WWFixedAsDWord      xPos,        /* coordinates to untransform */
          WWFixedAsDWord      yPos,
          PointWWFixed        * documentCoordinates);
                                 /* pointer to returned devide coordinates *
```

Apply the reverse of the device's transformation to the passed point.

Include: **graphics.h**

Routines

■ HAL_COUNT()

```
word      HAL_COUNT(
          dword  val);
```

This macro is provided for use with **HugeArrayLock()**. It extracts the lower word of the **HugeArrayLock()** return value. This is the number of elements in the Huge Array block after the locked one (counting that locked one).

■ HAL_PREV

```
word      HAL_PREV(
          dword  val);
```

This macro is provided for use with **HugeArrayLock()**. It extracts the upper word of the **HugeArrayLock()** return value. This is the number of elements in the Huge Array block before the locked one (counting that locked one).

■ HandleModifyOwner()

```
void      HandleModifyOwner(
          MemHandle           mh,       /* Handle of block to modify */
          GeodeHandle         owner);   /* Handle of block's new owner */
```

This routine changes the owner of the indicated global memory block. Note that this routine can be called only by a thread belonging to the block's original owner; that is, you can only use this routine to transfer ownership of a block *from* yourself *to* some other geode.

Include: heap.def

Never Use Situations:

Never use this unless the block already belongs to you and you are giving up ownership.

See Also: MemGetInfo(), MemModifyFlags(), MemModifyOtherInfo()

■ HandleP()

```
void      HandleP(
          MemHandle           mh);   /* Handle of block to grab */
```

If several different threads will be accessing the same global memory block, they need to make sure their activities will not conflict. The way they do that is to use synchronization routines to get control of a block. **HandleP()** is part of one set of synchronization routines.

If the threads are using this family of routines, then whenever a thread needs access to the block in question, it can call **HandleP()**. This routine checks whether any thread has grabbed the block with **HandleP()** (or **MemPLock()**). If no thread has the block, it grabs the block for the calling thread and returns (it does not lock the block on the global heap). If a thread has the block, **HandleP()** puts the thread on a priority queue and sleeps. When the block is free for it to take, it awakens, grabs the block, and returns.When the thread is done with the block, it should release it with **MemUnlockV()** or **HandleV()**.

Include: heap.h

Tips and Tricks: If you will be locking the block after you grab it, use the routine **MemPLock()** (which calls **HandleP()** and then locks the block with **MemLock()**). You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads accessing the block use **HandleP()** and/or **MemPLock()** to access the block. The routines use the *HM_otherInfo* field of the handle table entry; do not alter this field. Release the block with **HandleV()** or **MemUnlockV()** when you are done with it.

Warnings: If a thread calls **HandleP()** when it already has control of the block, it will deadlock; **HandleP()** will put the thread to sleep until the thread releases the block, but the thread will not be able to release the block because it's sleeping. **MemThreadGrab()** avoids this conflict. If you try to grab a non-sharable block owned by another thread, **HandleP()** will fatal-error.

See Also: HandleV(), MemPLock(), MemUnlockV()

■ HandleToOptr()

```
optr    HandleToOptr(
        Handle han;
```

This macro casts any handle to an optr, leaving the chunk handle portion of the resultant optr to be zero.

See Also: ConstructOptr(), OptrToHandle(), OptrToChunk()

■ HandleV()

```
void      HandleV(
          MemHandle          mh);   /* Handle of block to grab */
```

HandleV() is part of a set of synchronization routines. If several different threads will be accessing the same global memory block, they need to make sure their activities will not conflict. The way they do that is to use synchronization routines to get control of a block. **HandleV()** is part of one set of synchronization routines.

If a block is being accessed via these synchronization routines, then a thread will not access a block until it has "grabbed" it with **HandleP()** or **MemPLock()**. When a thread is done with the block, it can release it for use by the other threads by calling **HandleV()**. Note that **HandleV()** does not unlock the block; it just changes the block's semaphore so other threads can grab it.

Include: heap.h

Tips and Tricks: If you need to unlock the thread just before releasing it, use the routine **MemUnlockV()**, which first unlocks the thread, and then calls **HandleV()** to release it. You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads accessing the block use **HandleP()** or **MemPLock()** to access the thread. The routines use the *HM_otherInfo* field of the handle table entry; do not alter this field.

Warnings: Do not use this on a block unless you have grabbed it. The routine does not check to see that you have grabbed the thread; it just clears the semaphore and returns.

See Also: HandleP(), MemPLock(), MemUnlockV()

■ HugeArrayAppend()

```
void      HugeArrayAppend(
          VMFileHandle        file,
          VMBlockhandle       vmBlock,      /* Handle of directory block */
          word                numElem,      /* # of elements to add to end of
                                          * array */
          const void *        initData);    /* Copy into each new element */
```

This routine appends one or more elements to a Huge Array. The data pointed to by *initData* will be copied into each new element. If *initData* is a null pointer, the elements will be uninitialized.

If the Huge Array contains variable sized elements, this routine will append a single element; this element will be *numElem* bytes long.

Include: hugearr.h

■ HugeArrayCompressBlocks()

```
void      HugeArrayCompressBlocks(
          VMFileHandle        vmFile,   /* File containing Huge Array */
          VMBlockHandle       vmBlock); /* handle of directory block */
```

This routine compacts a Huge Array, resizing every block to be just as large as necessary to accommodate its elements. It does not change any of the data in the Huge Array.

Include: hugearr.h

■ HugeArrayContract()

```
word      HugeArrayContract(
          void **             elemPtr,      /* **elemPtr is first element to
                                          * delete */
          word                numElem);     /* # of elements to delete */
```

Delete a number of elements starting at an address in a Huge Array. The routine will fix up the pointer so it points to the first element after the deleted elements. The routine automatically locks and unlocks Huge Array blocks as necessary.

Include: hugearr.h

■ HugeArrayCreate()

```
VMBlockhandle HugeArrayCreate(
          VMFileHandle        vmFile,       /* Create in this VM file */
          word                elemSize,     /* Pass zero for variable-size
                                          * elements */
          word                headerSize);  /* Pass zero for default header */
```

This routine creates and initializes a Huge Array in the specified file. It returns the handle of the Huge Array's directory block.

Include: hugearr.h

■ HugeArrayDelete()

```
void      HugeArrayDelete(
          VMFileHandle        vmFile,
          VMBlockHandle       vmBlock,  /* handle of directory block */
          word                numElem,  /* # of elements to delete */
          dword               elemNum); /* Index of first element to delete */
```

This routine deletes one or more elements from a Huge Array. It contracts and frees blocks as necessary.

Include: hugearr.h

■ HugeArrayDirty()

```
void      HugeArrayDirty(
          const void *        elemPtr);     /* Element in dirty block */
```

This routine marks a block in a Huge Array as dirty. The routine is passed a pointer to anywhere in a dirty element; that element's block will be dirtied.

Include: hugearr.h

Warnings: Be sure to call this routine before you unlock the element; otherwise, the block may be discarded before you can dirty it.

■ HugeArrayDestroy()

```
void      HugeArrayDestroy(
          VMFileHandle        vmFile,
          VMBlockHandle       vmBlock); /* Handle of directory block */
```

This routine destroys a HugeArray by freeing all of its blocks.

Include: hugearr.h

■ HugeArrayEnum()

```
Boolean   HugeArrayEnum(
          VMFileHandle        vmFile,   /* subject to override */
          VMBlockHandle       vmBlock,  /* Handle of the Huge Array's directory
                                                * block */
          Boolean _pascal     (*callback) (       /* return true to stop */
                              void *    element,  /* element to examine */
                              void *    enumData),
          dword               startElement,/* first element to examine */
          dword               count,    /* examine this many elements */
          void *              enumData; /* this pointer is passed to callback
                                                * routine */
```

This routine lets you examine a sequence of elements in a Huge Array. **HugeArrayEnum()** is passed six arguments. The first two are a file handle and block handle; these specify the Huge Array to be examined. The third is a pointer to a Boolean callback routine. The fourth argument is the index of the first element to be examined (remember, the first element in the Huge Array has an index of zero). The fifth argument is the number of elements to examine, or -1 to examine through the last element. The sixth argument is a pointer which is passed unchanged to the callback routine; you can use this to pass data to the callback routine, or to keep track of a scratch space.

The callback routine, which must be declared _pascal, itself takes two arguments. The first is a pointer to an element in the huge array. The callback routine will be called once for each element in the specified range; each time, the first argument will point to the element being examined. The second argument is the pointer that was passed as the final argument to **HugeArrayEnum()**. The callback routine can make **HugeArrayEnum()** abort by returning *true*; this is useful if you need to search for a specific element. Otherwise, the callback routine should return *false*. If the callback routine aborts the enumeration, **HugeArrayEnum()** returns *true*; otherwise, it returns *false*.

HugeArrayEnum() is guaranteed to examine the elements in numerical order, beginning with *startElement*. The routine will automatically stop with the last element, even if *count* elements have not been enumerated. However, the starting element must be the index of an element in the array.

Include: hugearr.h

Warnings: The callback routine may not allocate, free, or resize any elements in the Huge Array. All it should do is examine or change (*without* resizing) a single element.

The starting element must be an element in the array. If you pass a starting index which is out-of-bounds, the results are undefined.

■ HugeArrayExpand()

```
word    HugeArrayExpand(
        void **             elemPtr,     /* **elemPtr is element at location
                                       * where new elements will be
                                       * created */
        word                numElem,     /* # of elements to insert */
        const void *        initData);   /* Copy this into each new
                                       * element */
```

This routine inserts a number of elements at a specified location in a HugeArray. The element pointed to will be shifted so it comes after the newly-created elements. The pointer will be fixed up to point to the first new element. The data pointed to by *initData* will be copied into each new element. If *initData* is null, the new elements will be uninitialized.

If the elements are of variable size, this routine will insert a single element; this element will be *numElem* bytes long.

Include: hugearr.h

■ HugeArrayGetCount()

```
dword   HugeArrayGetCount(
        VMFileHandle        vmFile,
        VMBlockHandle       vmBlock); /* Handle of directory block */
```

This routine returns the number of elements in a Huge Array.

Include: hugearr.h

■ HugeArrayInsert()

```
void    HugeArrayInsert(
        VMFileHandle        vmFile,
        VMBlockHandle       vmBlock,  /* Handle of directory block */
        word                numElem,  /* # of elements to insert */
        dword               elemNum,  /* Index of first new element */
        const void *        initData);/* Copy this into each new element */
```

This routine inserts one or more elements in the midst of a Huge Array. The first new element will have index *elemNum*; thus, the element which previously had that index will now come after the new elements. The data pointed to by *initData* will be copied into each new element. If *initData* is null, the new elements will be uninitialized.

If the elements are of variable size, this routine will insert a single element; this element will be *numElem* bytes long.

Include: heap.h

■ HugeArrayLock()

```
dword    HugeArrayLock(
         VMFileHandle        vmFile,
         VMBlockhandle       vmBlock,  /* Handle of directory block */
         dword               elemNum,  /* Element to lock */
         void **             elemPtr);    /* Pointer to element is written
                                        * here */
```

This routine locks an element in a Huge Array. It writes the element's address to **elemPtr*. The dword returned indicates how many elements come before and after the element in that block. The upper word indicates how many elements come before the locked one, counting the locked element. The lower word indicates how many elements come after the locked element, again counting the locked one. You may examine or change all the other elements in the block without making further calls to **HugeArrayLock()**.

Include: heap.h

See Also: HAL_COUNT(), HAL_PREV()

■ HugeArrayNext()

```
word     HugeArrayNext(
         void **             elemPtr);
```

This routine increments a pointer to an element in a HugeArray to point to the next element. If the element was the last element in its block, **HugeArrayNext()** will unlock its block and lock the next one. The routine writes the pointer to **elemPtr*; it returns the number of elements which come after the newly-locked one in its block, counting the newly-locked element. If this routine is passed a pointer to the last element in a HugeArray, it unlocks the element, writes a null pointer to **elemPtr,* and returns zero.

Include: heap.h

Warnings: This routine may unlock the block containing the passed element. Therefore, if you need to mark the block as dirty, do so before making this call.

■ HugeArrayPrev()

```
word      HugeArrayPrev(
          void **            elemPtr1,     /* indicates current element */
          void **            elemPtr2);
```

This routine decrements a pointer to an element in a HugeArray to point to the previous element. If the element was the first element in its block, **HugeArrayPrev()** will unlock its block and lock the previous one. The routine writes the pointer to **elemPtr*1, and writes a pointer to the first element in the block in **elemPtr2*. It returns the number of elements which come before the newly-locked one in its block, counting the newly-locked element. If this routine is passed a pointer to the first element in a HugeArray, it unlocks the element, writes a null pointer to **elemPtr*, and returns zero.

Include: hugearr.h

Warnings: This routine may unlock the block containing the passed element. Therefore, if you need to mark the block as dirty, do so before making this call.

■ HugeArrayReplace()

```
void      HugeArrayReplace(
          VMFileHandle       file,
          VMBlockHandle      vmblock,      /* Handle of directory block */
          word               numElem,      /* # of elements to replace */
          dword              elemNum,      /* First element to replace */
          const void *       initData);    /* Copy this into each element
```

This routine replaces one or more elements with copies of the passed data. If *initData* is null, the elements will be filled with null bytes.

If the elements are of variable size, a single element will be resized; its new size will be *enumData* bytes long.

Include: hugearr.h

See Also: HugeArrayResize()

■ HugeArrayResize()

```
void      HugeArrayResize(
          VMFileHandle        vmFile,
          VMBlockHandle       vmBlock,  /* Handle of directory block */
          dword               elemNum,  /* Resize this element */
          word                newSize); /* New size in bytes */
```

This routine resizes an element in a Huge Array. The array must contain variable-sized elements. If the new size is larger than the old, the extra space will be zero-initialized. If it is smaller, the element will be truncated.

Include: hugearr.h

■ HugeArrayUnlock()

```
void      HugeArrayUnlock(
          void *              elemPtr);
```

This routine unlocks the block of a HugeArray which contains the passed element.

Include: hugearr.h

Warnings: If you have changed any of the elements in the block, be sure to call **HugeArrayDirty()** *before* you unlock the block; otherwise the block might be discarded.

■ IACPConnect()

```
IACPConnection IACPConnect(
          GeodeToken          *list,
          IACPConnectFlags    flags,
          MemHandle           appLaunchBlock,
          optr                client,
          word                *numServers);
```

This routine establishes a connection between a client object (by default the calling thread's application object) and one or more servers registered with the indicated list.

The *client* argument should be **NullOptr** unless the IACPCF_CLIENT_OD_SPECIFIED flag is set in the flags parameter.

Include: iacp.goh

■ IACPCreateDefaultLaunchBlock()

```
MemHandle IACPCreateDefaultLaunchBlock(
          word               appMode);
```

This routine creates a memory block holding an **AppLaunchBlock** structure suitable for passing to **IACPConnect()**. The two valid values to pass in *appMode* are MSG_GEN_PROCESS_OPEN_APPLICATION and MSG_GEN_PROCESS_OPEN_ENGINE.

Include: iacp.goh

■ IACPFinishConnect()

```
void      IACPFinishConnect(
          IACPConnection     connection,
          optr               server);
```

Finishes a connection made to a server which had to change from non-interactible to interactible.

Include: iacp.goh

■ IACPLostConnection()

```
void IACPLostConnection(
          optr               oself,
          IACPConnection     connection);
```

This routine is called by IACP server objects to handle when a client closes a connection.

Include: iacp.goh

■ IACPProcessMessage()

```
void IACPProcessMessage(
          optr               oself,
          EventHandle        msgToSend,
          TravelOption       topt,
          EventHandle        completionMsg);
```

This is a utility routine to dispatch an encapsulated message handed to an object by an IACP connection.

Include: iacp.goh

■ IACPRegisterDocument()

```
void IACPRegisterDocument(
         optr   server,
         word   disk,
         dword  fileID);
```

This routine registers an open document and the server object for it.

This routine is to be used only by servers, not by clients, and should only be used by the creator of the document. There is no provision for using IACP to connect to a server that is not the creator of the document in question.

Include: iacp.goh

■ IACPRegisterServer()

```
void     IACPRegisterServer(
         GeodeToken          *list,
         optr                server,
         IACPServerMode      mode,
         IACPServerFlags     flags);
```

This routine registers an object as a server for the IACP server list specified by the passed token.

Include: iacp.goh

■ IACPSendMessage()

```
word IACPSendMessage(
         IACPConnection      connection,
         EventHandle         msgToSend,
         TravelOption        topt,
         EventHandle         completionMsg,
         IACPSide            side);
```

This routine sends a recorded message to all the objects on the other side of an IACP connection.

Include: iacp.goh

■ IACPSendMessageToServer()

```
word IACPSendMessageToServer(
          IACPConnection      connection,
          EventHandle         msgToSend,
          TravelOption        topt,
          EventHandle         completionMsg,
          word                serverNum);
```

This routine sends a message to a specific server on the other side of an IACP connection.

Include: iacp.goh

■ IACPShutdown()

```
void IACPShutdown(
          IACPConnection      connection,
          optr                serverOD);
```

This routine removes a server or client from an IACP connection.

Include: iacp.goh

■ IACPShutdownAll()

```
void IACPShutdownAll(
          optr obj);
```

This calls **IACPShutdown()** for all connections to which the passed object is a party. It's primarily used by **GenApplicationClass** when the application is exiting.

Include: iacp.goh

■ IACPUnregisterDocument()

```
void IACPUnregisterDocument(
          optr   server,
          word   disk,
          dword  fileID);
```

This routine unregisters an open document and the server object for it.

Include: iacp.goh

■ IACPUnregisterServer()

```
void IACPUnregisterServer(
          GeodeToken          *token,
          optr                object);
```

This removes the specified server object from the indicated IACP server list.

Include: iacp.goh

■ ImpexCreateTempFile()

```
TransError ImpexCreateTempFile(
          char *              buffer,
          word                fileType,
          FileHandle *        file,
          MemHandle *         errorString);
```

This routine creates and opens a unique temporary file to be used by translation libraries for file importing and exporting. The routine is called only by translation libraries.

The routine is passed the following arguments:

buffer	The file name will be written to the buffer pointed to by this argument. The buffer should be at least FILE_LONGNAME_BUFFER_SIZE bytes long.
fileType	This specifies what kind of temporary file should be created. If IMPEX_TEMP_VM_FILE is passed, a GEOS VM file will be created. If IMPEX_TEMP_NATIVE_FILE is passed, a temporary file in the native format will be created.
file	This is a pointer to a FileHandle variable. The temporary file's handle will be written to **file*.
errString	If **ImpexCreateTempFile** fails with error condition TE_CUSTOM it will allocate a block containing an error string. It will write the block's handle to **errString*. It is the caller's responsibility to free this block when it's done with it.

If **ImpexCreateTempFile** is successful, it returns TE_NO_ERROR (which equals zero). If it fails, it returns a member of the **TransferErrors** enumerated type (usually TE_METAFILE_CREATION_ERROR). When you're done with the temporary file, call **FileDeleteTempFile()**.

Include: impex.goh

Warnings: If you close this file, the system may delete it at any time. Ordinarily you should close it with **ImpexDeleteTempFile()**, which deletes the file immediately.

If the routine does not fail with condition TE_CUSTOM, **errString* may contain a random value. Do not use **errString* if the routine did not return TE_CUSTOM.

■ ImpexDeleteTempFile()

```
TransError ImpexDeleteTempFile(
          const char *        buffer,
          FileHandle          tempFile,
          word                fileType);
```

This routine closes, then deletes, a temporary file which was created by **ImpexCreateTempFile()**. It is passed the following arguments:

buffer	This is a pointer to a character buffer containing the name of the temporary file. You can just pass the address of the buffer which was filled by **ImpexCreateTempFile()**.
tempFile	This is the handle of the temporary file.
fileType	This specifies what type of file is being deleted. If the temporary file is a GEOS VM file, this will be IMPEX_TEMP_VM_FILE. If it is a native-format file, it will be IMPEX_TEMP_NATIVE_FILE.
errString	If **ImpexDeleteTempFile** fails with error condition TE_CUSTOM it will allocate a block containing an error string. It will write the block's handle to **errString*. It is the caller's responsibility to free this block when it's done with it.

ImpexDeleteTempFile() closes the specified file, then deletes it. If it is successful, it returns TE_NO_ERROR (i.e. zero); otherwise, it returns an appropriate member of the **TransError** enumerated type.

Include: impex.goh

Warnings: If the routine does not fail with condition TE_CUSTOM, **errString* may contain a random value. Do not use **errString* if the routine did not return TE_CUSTOM.

■ ImpexExportToMetafile()

```
TransError ImpexExportToMetafile(
          Handle              xlatLib,
          VMFileHandle        xferFile,
          FileHandle          metafile,
          dword               xferFormat,
          word                arg1,
          word                arg2,
          MemHandle *         errString);
```

This routine is used by translation libraries. The routine calls an intermediate translation library to finish translating a given file into the GEOS Metafile format.

Include: impex.goh

Warnings: If the routine does not fail with condition TE_CUSTOM, **errString* may contain a random value. Do not use **errString* if the routine did not return TE_CUSTOM.

■ ImpexImportExportCompleted()

```
void      ImpexImportExportCompleted(
          ImpexTranslationParams *itParams);
```

The application should send this message when it is finished importing or exporting data. The routine will send an appropriate acknowledgment message to the ImportControl or ExportControl object, depending on the settings of *ITP_impexOD* and *ITP_returnMsg*.

If the application has just finished an import, it should not have changed the **ImpexTranslationParams** structure. If it had just finished preparing data for export, it should have set the *ITP_transferVMChain* field to contain the handle of the head of the VM chain.

Warnings: This routine, in essence, informs the ImportControl or ExportControl object that the application is finished with the transfer file. The ImportControl will respond by destroying the transfer file; the ExportControl will call the appropriate translation library to produce an output file. Therefore, an application should not call this routine until it is absolutely finished with the transfer file.

■ ImpexImportFromMetafile()

```
TransError ImpexExportToMetafile(
          Handle              xlatLib,
          VMFileHandle        xferFile,
          FileHandle          metafile,
          dword *             xferFormat,
          word                arg1,
          word                arg2,
          MemHandle *         errString);
```

This routine is used by translation libraries. The routine calls an intermediate translation library to translate a given file from the GEOS Metafile format to an intermediate format.

Include: impex.goh

Warnings: If the routine does not fail with condition TE_CUSTOM, **errString* may contain a random value. Do not use **errString* if the routine did not return TE_CUSTOM.

■ InitFileCommit()

```
void      InitFileCommit(void);
```

This routine commits any changes to the GEOS.INI file, removing and replacing its stored backup. It ensures that no other threads are working on the file during the commit operation.

Include: initfile.h

■ InitFileDeleteCategory()

```
void      InitFileDeleteCategory(
          const char *category);
```

This routine deletes the specified category, along with all its entries, from the GEOS.INI file. Pass it the following:

category	A pointer to the null-terminated string representing the category to be deleted. This string ignores white space and is case-insensitive.

Include: initfile.h

■ InitFileDeleteEntry()

```
void     InitFileDeleteEntry(
         const char *category,
         const char *key);
```

This routine deletes an entry in the GEOS.INI file. Pass it the following:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be deleted.

Include: initfile.h

■ InitFileDeleteStringSection()

```
void     InitFileDeleteStringSection(
         const char *        category,
         const char *        key,
         word                stringNum);
```

This routine deletes the specified string section from the given blob in the GEOS.INI file. Pass it the following:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be edited.
stringNum	The zero-based string section number.

Include: initfile.h

■ InitFileEnumStringSection()

```
Boolean  InitFileEnumStringSection(
         const char *        category,
         const char *        key,
         InitFileReadFlags   flags,
         Boolean _pascal (*callback)(const char *stringSection,
                              word          sectionNum,
                              void *        enumData),
         void *              enumdata);
```

This routine enumerates a particular blob, allowing a callback routine to process each of the string sections in it. The routine will stop processing either after the last string section or when the callback routine returns *true*.

Pass this routine the following:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be enumerated.
flags	A record of **InitFileReadFlags** indicating the method of character conversion upon reading (upcase all, downcase all, do not change).
callback	A pointer to a Boolean callback routine. The callback routine is described below.
enumData	This pointer is passed unchanged to the callback routine. **InitFileEnumStringSection()** does not use it.

This routine returns a Boolean value. It returns *true* if the callback routine halted the enumeration by returning *true*; otherwise, it returns *false*.

Callback Routine:

The callback routine may do anything it wants with the string section it receives. It must be declared _pascal. It must return a Boolean value: If it returns *true*, **InitFileEnumStringSection()** will stop processing the blob. If it returns *false*, processing will continue to the next string section, if any. The callback will receive the following parameters:

stringSection	A pointer to the null-terminated string section to be processed.
sectionNum	The zero-based number of the string section currently being processed.

enumData	A pointer passed through from the caller of **InitFileEnumStringSection()**.

Include: initfile.h

■ InitFileGetTimeLastModified()

```
dword    InitFileGetTimeLastModified(void);
```

This routine returns the time when the GEOS.INI file was last modified. The returned time is the value of the system counter when the file was last written.

Include: initfile.h

■ InitFileReadBoolean()

```
Boolean  InitFileReadBoolean(
         const char *        category,
         const char *        key,
         Boolean *           bool);
```

This routine reads a Boolean entry in the GEOS.INI file, copying it into a passed buffer. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteBoolean()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
bool	A pointer to a Boolean variable in which the Boolean value will be returned.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Warnings: The return value of this function is *not* the Boolean stored in the GEOS.INI file. That value is returned in the Boolean pointed to by *bool*.

Include: initfile.h

■ InitFileReadDataBlock()

```
Boolean  InitFileReadDataBlock(
         const char *        category,
         const char *        key,
         MemHandle *         block,
         word *              dataSize);
```

This routine reads an entry in the GEOS.INI file, allocating a new block and copying the data into it. The routine returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteData()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
block	A pointer to a null memory handle. This pointer will point to the newly-allocated block handle upon return. The data read will be in the new block. It is your respojnsibility to free this block when you're done with it.
dataSize	The size of the read data. All the data will be read; the block will be as large as necessary.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadDataBuffer()

```
Boolean  InitFileReadDataBuffer(
         const char *        category,
         const char *        key,
         void *              buffer,
         word                bufSize,
         word *              dataSize);
```

This routine reads an entry in the GEOS.INI file, copying it into a passed buffer. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteData()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
buffer	A pointer to the buffer in which the data will be returned. This buffer must be in locked or fixed memory.
bufSize	The size of the passed buffer in bytes. If you are not sure what the data's size will be, you may want to use the (slightly less efficient) **InitFileReadDataBlock()**.
dataSize	A pointer to a word; on return, the word pointed to will contain the size (in bytes) of the data returned.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadInteger()

```
Boolean InitFileReadInteger(
        const char *        category,
        const char *        key,
        word *              i);
```

This routine reads an integer entry in the GEOS.INI file, copying it into the passed variable. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteInteger()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
i	A pointer to a word in which the integer will be returned.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadStringBlock()

```
Boolean  InitFileReadStringBlock(
         const char *        category,
         const char *        key,
         MemHandle *         block,
         InitFileReadFlags   flags,
         word *              dataSize);
```

This routine reads a string entry in the GEOS.INI file, allocates a new block on the global heap, and copies the read string into the new block. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteString()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
block	A pointer to a memory block handle variable. Upon return, this variable will contain the handle of the newly allocated block; the block will contain the string read from the file. It is your responsibility to free this block when you're done with it.
flags	A record of **InitFileReadFlags** indicating the method of character conversion upon reading (upcase all, downcase all, do not change).
dataSize	A pointer to a word which, upon return, will contain the size of the string (in bytes) actually read from the file.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadStringBuffer()

```
Boolean  InitFileReadStringBuffer(
         const char *        category,
         const char *        key,
         char *              buffer,
         InitFileReadFlags   flags,
         word *              dataSize);
```

This routine reads a string entry in the GEOS.INI file, copying it into a passed, locked buffer. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteString()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
buffer	A pointer to a buffer into which the returned string will be written. This buffer must be in locked or fixed memory. If you don't know the approximate size of the data, you may want to use the (slightly less efficient) **InitFileReadStringBlock()**.
flags	A record of **InitFileReadFlags** indicating the size of the passed buffer as well as the method of character conversion upon reading (upcase all, downcase all, do not change).
dataSize	A pointer to a word which, upon return, will contain the size of the string (in bytes) actually read from the file.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadStringSectionBlock()

```
Boolean  InitFileReadStringSectionBlock(
         const char *        category,
         const char *        key,
         word                section,
         MemHandle *         block,
         InitFileReadFlags   flags,
         word *              dataSize);
```

This routine reads a string section from the specified entry in the GEOS.INI file, allocates a new block on the global heap, and copies the read string section into the new block. It returns the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteString()** or **InitFileWriteStringSection()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
section	The zero-based number of the string section to retrieved.
block	A pointer to a memory block handle. Upon return, this pointer will point to the handle of the newly allocated block; the block will contain the string section read from the file.
flags	A record of **InitFileReadFlags** indicating the method of character conversion upon reading (upcase all, downcase all, do not change).
dataSize	A pointer to a word which, upon return, will contain the size of the string section (in bytes) actually read from the file.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileReadStringSectionBuffer()

```
Boolean  InitFileReadStringSectionBuffer(
         const char *        category,
         const char *        key,
         word                section,
         char *              buffer,
         InitFileReadFlags   flags,
         word *              dataSize);
```

This routine reads a string section from the specified entry in the GEOS.INI file, copying it into a passed, locked buffer. It returns the indicated section in the first instance of the category/key combination it encounters, searching the local INI file first. Thus, local settings will always override system or network settings.

This routine is used for reading data written with **InitFileWriteStringSection()**. Pass it the following parameters:

category	A pointer to the null-terminated string representing the category in which the entry resides. This string ignores white space and is case-insensitive.
key	A pointer to the null-terminated string representing the key to be retrieved.
section	The zero-based number of the string section to be retrieved.
buffer	A pointer to a buffer into which the returned string section will be written. This buffer must be in locked or fixed memory. If you don't know the approximate size of the string section, you may want to use the (slightly less efficient) **InitFileReadStringSectionBlock()**.
flags	A record of **InitFileReadFlags** indicating the size of the passed buffer as well as the method of character conversion upon reading (upcase all, downcase all, do not change).
dataSize	A pointer to a word which, upon return, will contain the size of the string section (in bytes) actually read from the file.

The function's return value will be *true* if an error occurs or if the entry could not be found; it will be *false* otherwise.

Include: initfile.h

■ InitFileRevert()

```
Boolean  InitFileRevert(void);
```

This routine restores the GEOS.INI file from its saved backup version. It ensures that no other thread is operating on the file while it is being restored. This function returns an error flag: *true* represents an error in restoring the file; *false* indicates success.

Include: initfile.h

■ InitFileSave()

```
Boolean  InitFileSave(void);
```

This routine saves the GEOS.INI file synchronously by updating the backup file to be the current version. (**InitFileCommit()** actually overwrites the GEOS.INI file itself.) It ensures that no other thread is operating on the file while it is being written out. This function returns an error flag: *true* represents an error in trying to save the file; *false* indicates success.

Include: initfile.h

■ InitFileWriteBoolean()

```
void     InitFileWriteBoolean(
         const char *         category,
         const char *         key,
         Boolean              bool);
```

This integer writes a Boolean value into the specified category and key of the local GEOS.INI file. The Boolean will appear as “true” or “false” if the user looks at GEOS.INI with a text editor, but it will be an actual Boolean value to GEOS. Pass this routine the following:

category	A pointer to the null-terminated character string representing the INI category into which the data should be written.
key	A pointer to the null-terminated character string representing the INI key within *category* into which the data should be written.
bool	The Boolean value to be written.

Once written, the Boolean value can be read with **InitFileReadBoolean()**.

Include: initfile.h

■ InitFileWriteData()

```
void      InitFileWriteData(
          const char          *category,
          const char          *key,
          const void          *buffer,
          word                bufSize);
```

This routine writes a given piece of data to the local GEOS.INI file. Pass it the following:

category	A pointer to the null-terminated character string representing the INI category into which the data should be written.
key	A pointer to the null-terminated character string representing the INI key within *category* into which the data should be written.
buffer	A pointer to a locked or fixed buffer containing the data to be written.
bufSize	The size of the buffer in bytes.

Once data has been written to the INI file, it can be read with **InitFileReadDataBlock()** or **InitFileReadDataBuffer()**.

Include: initfile.h

■ InitFileWriteInteger()

```
void      InitFileWriteInteger(
          const char          *category,
          const char          *key,
          word                value);
```

This routine writes an integer into the category and key specified for the local GEOS.INI file. Pass the following:

category	A pointer to the null-terminated character string representing the INI category into which the data should be written.
key	A pointer to the null-terminated character string representing the INI key within *category* into which the data should be written.
value	The integer to be written.

The integer, once written, can be read with **InitFileReadInteger()**.

Include: initfile.h

■ InitFileWriteString()

```
void      InitFileWriteString(
          const char *category,
          const char *key,
          const char *str);
```

This routine writes an entire string into the category and key specified for the local GEOS.INI file. Pass it the following:

category	A pointer to the null-terminated character string representing the INI category into which the data should be written.
key	A pointer to the null-terminated character string representing the INI key within *category* into which the data should be written.
str	A pointer to the null-terminated string to be written. If the string contains line feeds or carriage returns, it will automatically be parsed into string segments and be put within curly braces; if it contains curly braces, all closing braces will automatically have a backslash inserted before them.

To read a string written with this routine, use **InitFileReadStringBlock()** or **InitFileReadStringBuffer()**.

Include: initfile.h

■ InitFileWriteStringSection()

```
void      InitFileWriteStringSection(
          const char *category,
          const char *key,
          const char *string);
```

This routine appends a string section onto the blob specified by the *category* and *key* parameters. The string section will become part of the blob and will be its last section. The section may not contain any carriage returns or line feeds. Pass this routine the following:

category	A pointer to the null-terminated character string representing the INI category into which the data should be written.
key	A pointer to the null-terminated character string representing the INI key within *category* into which the data should be written.
string	A pointer to the string section to be written.

Once written, the segment may be read with **InitFileReadStringSectionBlock()** or **InitfileReadStringSectionBuffer()**.

Include: initfile.h

■ InkDBGetDisplayInfo()

```
void     InkDBGetDisplayInfo(
         InkDBDisplayInfo *  retVal,
         VMFileHandle        fh);
```

This routine returns the dword ID of the note or folder which is presently being displayed by the Ink Database. It also returns the ID of the parent folder, and the page number, if applicable.

Structures: It returns this information by filling in an **InkDBDisplayInfo** structure:

```
typedef struct {
      dword  IDBDI_currentDisplay;
      dword  IDBDI_parentFolder;
      word   IDBDI_pageNumber;
} InkDBDisplayInfo;
```

Include: pen.goh

■ InkDBGetHeadFolder()

```
dword    InkDBGetHeadFolder(
         VMFileHandle        fh);
```

This routine returns the dword ID of the head folder of an Ink Database file.

Include: pen.goh

■ InkDBInit()

```
void     InkDBInit(
         VMFileHandle        fh);
```

This routine takes a new Ink Database file. It initializes the file for use, creating all needed maps and a top-level folder.

Include: pen.goh

■ InkDBSetDisplayInfo()

```
void      InkDBSetDisplayInfo(
          VMFileHandle        fh,
          dword               ofh,   /* Parent Folder dword ID# */
          dword               note,  /* ID# of note or folder to display */
          word                page); /* If displaying note, page # to display*/
```

This routine sets the display information for an Ink Database file. This routine sets the user's location in the database. The caller must supply the dword ID number of the note or folder to display, the parent folder (0 if displaying the top level folder), and the page number to display if displaying a note.

Include: pen.goh

■ InkFolderCreateSubFolder()

```
dword     InkFolderCreateSubFolder(
          dword               tag, /* ID# of parent folder (0 for top-level) */
          VMFileHandle        fh); /* Handle of Ink DB file */
```

This routine creates a subfolder within the passed folder. The new folder is automatically added to it's parent's chunk array. The return value is new folder's dword ID number.

Include: pen.goh

■ InkFolderDelete()

```
void      InkFolderDelete(
          dword               tag,   /* ID# of folder */
          VMFileHandle        fh);   /* Handle of Ink DB file */
```

This routine removes an Ink Database folder.

Include: pen.goh

■ InkFolderDepthFirstTraverse()

```
word      InkFolderDepthFirstTraverse(
          dword               rfldr, /* ID# of folder at root of search tree */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          Boolean _pascal     (*callback)( /* far ptr to callback routine */
                              dword        fldr,
                              VMFileHandle fh,
                              word *       info),
          word *              info); /* Extra data to pass to callback */
```

This routine does a depth-first traversal of a folder tree. The callback routine, which must be declared _pascal, can halt the search by returning *true*, in which case the search routine will immediately return *true*; otherwise the search will return *false*.

Include: pen.goh

■ InkFolderDisplayChildInList()

```
void      InkFolderDisplayChildInList(
          dword               fldr,  /* ID# of folder */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          optr                list,  /* GenDynamicList */
          word                entry, /* entry number of child to display */
          Boolean             displayFolders); /* Include monikers in count,
                                          * return their monikers */
```

This routine requests that a dynamic list display the name of one of a folder's children. It is normally called in an applications *GDLI_queryMsg* handler.

Include: pen.goh

■ InkFolderGetChildInfo()

```
Boolean   InkFolderDisplayChildInfo( /* true if folder; else note */
          dword               fldr,  /* ID# of folder */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          word                entry, /* entry number of child */
          dword *             childID);/* Pointer to returned child ID # */
```

This routine returns information about one of a folder's children. The explicit return value will be true if the child is a folder, false if the child is a note. In addition, the passed dword pointer will point to the child's dword ID number.

Include: pen.goh

■ InkFolderGetChildNumber()

```
word      InkFolderDisplayChildInList(
          dword               fldr,  /* ID# of folder */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          dword               note); /* ID# of child note or folder */
```

This routine returns the passed note or folder's entry number within its passed parent folder.

Include: pen.goh

■ InkFolderGetContents()

```
DBGroupAndItem InkFolderGetContents(
          dword               tag,           /* ID# of folder */
          VMFileHandle        fh,            /* Handle of Ink DB file */
          DBGroupAndItem *    subFolders); /* pointer to return value */);
```

This routine returns the contents of a folder. It returns two chunk arrays, each of which is filled with dword ID numbers of the folder's children. The explicitly returned array holds the numbers of the folder's child notes. The routine also fills in a pointer with a DB item holding a chunk array with the ID numbers of the subfolders.

Include: pen.goh

■ InkFolderGetNumChildren()

```
dword     InkFolderGetNumChildren( /* Subfolders:Notes */
          dword               fldr,  /* ID# of folder */
          VMFileHandle        fh);   /* Handle of Ink DB file */
```

This message returns the number of children the Ink Database folder has. The high word of the return value holds the number of sub folders; the low word holds the number of notes.

Include: pen.goh

■ InkFolderMove()

```
void      InkFolderMove(
          dword               fldr,  /* ID# of folder to move */
          dword               pfldr);/* ID# of new parent folder */
```

This routine moves an Ink Database folder to a new location in the folder tree.

Include: pen.goh

■ InkFolderSetTitle()

```
void      InkFolderSetTitle(
          dword               tag,   /* ID# of folder */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          const char *        name); /* Text object */);
```

This routine renames an Ink Database folder. The passed name should be null-terminated.

Include: pen.goh

■ InkFolderSetTitleFromTextObject()

```
void      InkFolderSetTitleFromTextObject(
          dword               tag,   /* ID# of folder */
          FileHandle          fh,    /* Handle of Ink DB file */
          optr                text); /* Text object */);
```

This routine sets the name of the passed Ink Database folder from the contents of the passed VisText object.

Include: pen.goh

■ InkGetDocPageInfo()

```
void      InkGetDocPageInfo(
          PageSizeReport *    psr,   /* Structure to fill with return value */
          VMFileHandle        fh);
```

This routine returns the dword ID of the head folder of an Ink Database file.

Include: pen.goh

■ InkGetDocCustomGString()

```
GStateHandle InkGetDocCustomGString(
          VMFileHandle        dbfh);
```

This routine returns the custom GString associated with the passed Ink Database file. Note that this custom background will only be used if the document's basic **InkBackgroundType** is IBT_CUSTOM. (This may be determined using the **InkDBSetDocGString()** routine.

Include: pen.goh

■ InkGetDocGString()

```
InkBackgroundType InkGetDocGString(
          VMFileHandle       dbfh);
```

This routine returns the standard GString to use as a background picture with the passed Ink Database file. If the returned background type is custom, be sure to also call **InkGetDocCustomGString()**.

Include: pen.goh

■ InkGetParentFolder()

```
dword     InkGetParentFolder(
          dword              tag,   /* ID# of folder or note */
          VMFileHandle       fh);   /* Handle of Ink DB file */
```

This message returns the dword ID of the passed Ink Database note or folder.

Include: pen.goh

■ InkGetTitle()

```
word      InkGetTitle(
          dword              tag,   /* ID# of folder or note */
          VMFileHandle       fh,    /* Handle of Ink DB file */
          char *             dest); /* should be INK_DB_MAX_TITLE_SIZE +1 */);
```

This message fills the passed text buffer with the folder’s or note’s title, a null-terminated string. The routine’s explicit return value is the length of the string (including the terminator).

Include: pen.goh

■ InkNoteCopyMoniker()

```
dword     InkNoteCopyMoniker(
          dword  title,      /* ID# of parent folder */
          optr   list,       /* Output list */
          word   type,       /* 1: text note
                              * 0: ink note
                              * -1: folder */
          word   entry);     /* Handle of Ink DB file */
```

This routine copies the icon nd title into the VisMoniker.

Include: pen.goh

■ InkNoteCreate()

```
dword     InkNoteCreate(
          dword               tag,   /* ID# of parent folder */
          VMFileHandle        fh);   /* Handle of Ink DB file */
```

This routine creates a note and adds it to the passed folder's child list. The new note's dword ID is returned.

Include: pen.goh

■ InkNoteCreatePage()

```
word      InkNoteCreatePage(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          word                page); /* Page number to insert before,
                                         * CA_NULL_ELEMENT to append */
```

This routine creates a new page within a note. It returns the new page number.

Include: pen.goh

■ InkNoteDelete()

```
void      InkNoteDelete(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh);   /* Handle of Ink DB file */
```

This message deletes the passed note. All references to the note are deleted.

Include: pen.goh

■ InkNoteFindByKeywords()

```
ChunkHandle InkNoteFindByKeywords(
                /* Return value is chunk array with elements:
                 *  FindNoteHeader
                 *  -dword tag-
                 *  -dword tag-
                 *   etc… */
          VMFileHandle       fh,
          char *             strings,/* strings to match (separated by
                                    * whitespace or commas), can contain
                                    * C_WILDCARD or C_SINGLE_WILDCARD */
          word               opt,   /* true to match all keywords;
                                  * false to match at least one keyword */
```

This routine returns a chunk array containing the dword ID numbers of all notes whose keywords match the passed search string, preceded by the number of matching notes. If no such notes are found, then the returned handle will be NULL.

Note that this routine will only return about 20K notes; if there are more that match, only the first 20K will be returned.

Include: pen.goh

■ InkNoteFindByTitle()

```
ChunkHandle InkNoteFindByTitle(
                /* Return value is chunk array with elements:
                 *  FindNoteHeader
                 *  -dword tag-
                 *  -dword tag-
                 *   etc… */
          const char *       string,/* string to match (can contain C_WILDCARD
                                   * or C_SINGLE_WILDCARD */
          SearchOptions      opt,   /* Search options */
          Boolean            Body,  /* true if you want to look in the body
                                  * of text notes */
          VMFileHandle       fh);   /* Handle of Ink DB file */
```

This routine returns a chunk array containing the dword ID numbers of all notes whose titles match the passed search string, preceded by the number of matching notes. If no such notes are found, then the returned handle will be NULL.

Note that this routine will only return about 20K notes; if there are more that match, only the first 20K will be returned.

Include: pen.goh

■ InkNoteGetCreationDate()

```
dword    InkNoteGetCreationDate(
         dword              tag,   /* ID# of note */
         VMFileHandle       fh);   /* Handle of Ink DB file */
```

This routine gets a note's creation date.

Include: pen.goh

■ InkNoteGetKeywords()

```
void     InkNoteGetKeywords(
         dword              tag,   /* ID# of note */
         VMFileHandle       fh,    /* Handle of Ink DB file */
         char *             text); /* String to hold return value */);
```

This routine fills the passed buffer with the note's keywords. The target buffer should be of atleast length INK_DB_MAX_NOTE_KEYWORDS_SIZE +1. The string will be null-terminated.

Include: pen.goh

■ InkNoteGetModificationDate()

```
dword    InkNoteGetModificationDate(
         dword              tag,   /* ID# of note */
         VMFileHandle       fh);   /* Handle of Ink DB file */
```

This routine gets a note's modification date.

Include: pen.goh

■ InkNoteGetNoteType()

```
NoteType InkNoteGetNoteType( /* 0: Ink, 1: Text */
         dword              tag,   /* ID# of note */
         VMFileHandle       fh);   /* Handle of Ink DB file */
```

This routine gets a note's **NoteType**: NT_INK or NT_TEXT.

Include: pen.goh

■ InkNoteGetNumPages()

```
word     InkNoteGetNumPages(
         dword              tag);  /* ID# of note */
```

This routine returns the number of pages within the passed note.

Include: pen.goh

■ InkNoteGetPages()

```
DBGroupAndItem InkNoteGetPages(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh);   /* Handle of Ink DB file */
```

This routine returns a DB group and item containing a chunk array. The chunk array contains the page information of the note, either compressed pen data or text. Each array element holds one page of data.

Include: pen.goh

■ InkNoteLoadPage()

```
void      InkNoteLoadPage(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          word                page,  /* Page number */
          optr                obj,   /* an Ink or VisText object */
          word                type); /* note type 0: ink, 1: text */
```

This routine loads a visual object (Ink or Text) with the contents of the passed Ink Database page. Be sure to load only the correct type of data into an object.

Include: pen.goh

■ InkNoteMove()

```
void      InkNoteMove(
          dword               tag,      /* ID# of note */
          dword               pfolder,  /* ID# of new parent folder */
          VMFileHandle        fh);      /* Handle of Ink DB file */
```

This message moves the passed note to a new location. All references to the note are suitably altered.

Include: pen.goh

■ InkNoteSavePage()

```
void      InkNoteSavePage(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          word                page,  /* Page number */
          optr                obj,   /* an Ink or VisText object */
          word                type); /* note type 0: ink, 1: text */
```

This routine saves the contents of a visual object (Ink or Text) to the passed Ink Database page.

Include: pen.goh

■ InkNoteSendKeywordsdToTextObject()

```
void      InkNoteSendKeywordsToTextObject(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          optr                text); /* Text object to set */);
```

This message replaces the passed VisText object's text with the keywords from the passed folder or note of an Ink Database file.

Include: pen.goh

■ InkNoteSetKeywords()

```
void      InkNoteSetKeywords(
          dword               tag,      /* ID# of note */
          VMFileHandle        fh,       /* Handle of Ink DB file */
          const char *        text);    /* Keyword string */);
```

This message sets an Ink Database note's keywords. The passed string should be null-terminated.

Include: pen.goh

■ InkNoteSetKeywordsFromTextObject()

```
void      InkNoteSetKeywordsFromTextObject(
          dword               tag,      /* ID# of note */
          VMFileHandle        fh,       /* Handle of Ink DB file */
          optr *              text);    /* Text object */);
```

This message sets an Ink Database note's keywords by copying them from the passed text object.

Include: pen.goh

■ InkNoteSetModificationDate()

```
void      InkNoteSetModificationDate(
          word                tdft1, /* First two words of */
          word                tdft2, /*   TimerDateAndTime structure */
          dword               note,  /* ID# of note */
          FileHandle          fh);   /* Handle of Ink DB file */
```

This routine sets a note's modification date.

Include: pen.goh

■ InkNoteSetNoteType()

```
void      InkNoteSetNoteType(
          dword               tag,   /* ID# of note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          NoteType            nt);   /* NT_INK or NT_TEXT */
```

This routine sets a note's type: text or ink.

Include: pen.goh

■ InkNoteSetTitle()

```
void      InkNoteSetTitle(
          dword               tag,      /* ID# of note */
          VMFileHandle        fh,       /* Handle of Ink DB file */
          const char *        name);    /* Text object */);
```

This message renames an Ink Database note. The passed name should be null-terminated. The string may be up to INK_DB_MAX_NOTE_KEYWORDS_SIZE +1 in length.

Include: pen.goh

■ InkNoteSetTitleFromTextObject()

```
void      InkNoteSetTitleFromTextObject(
          dword               tag,   /* ID# of note */
          FileHandle          fh,    /* Handle of Ink DB file */
          optr                text); /* Text object */);
```

This message sets the name of the passed Ink Database note from the contents of the passed VisText object.

Include: pen.goh

■ InkSendTitleToTextObject()

```
void      InkSendTitleToTextObject(
          dword               tag,   /* ID# of folder or note */
          VMFileHandle        fh,    /* Handle of Ink DB file */
          optr                to);   /* Text object to set */);
```

This message replaces the passed VisText object's text with the name from the passed folder or note of an Ink Database file.

Include: pen.goh

■ InkSetDocCustomGString()

```
void      InkSetDocCustomGString(
          VMFileHandle        dbfh,
          Handle              gstring);
```

This routine sets the custom GString to use as a background for the passed Ink Database file. Note that this custom background will only be used if the document's basic **InkBackgroundType** is IBT_CUSTOM. (Set this using the **InkDBSetDocGString()** routine.)

Include: pen.goh

■ InkSetDocGString()

```
void      InkSetDocGString(
          VMFileHandle        dbfh,
          InkBackgroundType   type);
```

This routine sets the standard GString to use as a background picture with the passed Ink Database file. If the passed background type is custom, be sure to also call **InkSetDocCustomGString()**.

Include: pen.goh

■ InkSetDocPageInfo()

```
void      InkSetDocPageInfo(
          PageSizeReport *    psr,
          VMFileHandle        fh);
```

Set the page information for an Ink Database file.

Include: pen.goh

■ IntegerOf()

```
word      IntegerOf(
          WWFixedAsDWord      wwf)
```

This macro returns the integral portion of a **WWFixedAsDWord** value.

Include: geos.h

■ LMemAlloc()

```
ChunkHandle LMemAlloc(
        MemHandle           mh,        /* Handle of block containing heap */
        word                chunkSize);   /* Size of new chunk in bytes */
```

This routine allocates a new chunk in the LMem heap. The heap must be locked or fixed. It allocates a chunk, expanding the chunk table if enccessary, and returns the chunk's handle. The chunk is not zero-initialized. If the chunk could not be allocated, it returns a null handle. Chunks are dword-aligned, so the chunk's actual size may be slightly larger than you request.

Include: lmem.h

Be Sure To: Lock the block on the global heap (unless the block is fixed).

Warnings: The heap may be compacted; thus, all pointers to chunks are invalidated. If LMF_NO_EXPAND is not set, the heap may be resized (and thus moved), thus invalidating all pointers to that block. Even fixed blocks can be resized and moved.

See Also: LMemDeref(), LMemReAlloc()

■ LMemContract()

```
void    LMemContract(
        MemHandle           mh);   /* Handle of LMem heap */
```

This routine contracts an LMem heap; that is, it deletes all the free chunks, moves all the used chunks to the beginning of the heap (right after the chunk handle table), and resizes the block to free the unused space at the end. It's a good idea to call this routine if you have just freed a lot of chunks, since that will free up some of the global heap. The LMem heap is guaranteed not to move; however, all pointers to chunks will be invalidated.

Be Sure To: Lock the block on the global heap (if it isn't fixed).

Include: lmem.h

■ LMemDeleteAt()

```
void      LMemDeleteAt(
          optr   chunk,                 /* Chunk to resize */
          word   deleteOffset,          /* Offset within chunk of first
                                      * byte to be deleted */
          word   deleteCount);          /* # of bytes to delete */
```

This routine deletes a specified number of bytes from inside a chunk. It is guaranteed not to cause the heap to be resized or compacted; thus, pointers to other chunks remain valid.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: The bytes you delete must all be in the chunk. If *deleteOffset* and *deleteCount* indicate bytes that are not in the chunk, results are undefined.

Include: lmem.h

See Also: LMemReAlloc(), LMemInsertAt(), LMemDeleteAtHandles()

■ LMemDeleteAtHandles()

```
void      LMemDeleteAtHandles(
          MemHandle           mh,           /* Handle of LMem heap */
          ChunkHandle         ch,           /* Handle of chunk to resize */
          word                deleteOffset, /* Offset within chunk of first
                                          * byte to be deleted */
          word                deleteCount); /* # of bytes to delete */
```

This routine is exactly like **LMemDeleteAt()** above, except that the chunk is specified by its global and chunk handles.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: The bytes you delete must all be in the chunk. If *deleteOffset* and *deleteCount* indicate bytes that are not in the chunk, results are undefined.

Include: lmem.h

■ LMemDeref()

```
void *    LMemDeref(
          optr   chunk);      /* optr to chunk to dereference */
```

This routine translates an optr into the address of the chunk. The LMem heap must be locked or fixed on the global heap. Chunk addresses can be invalidated by many LMem routines, forcing you to dereference the optr again.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

See Also: LMemDerefHandles()

■ LMemDerefHandles()

```
void *    LMemDerefHandles(
          MemHandle          mh,           /* Handle of LMem heap's block */
          ChunkHandle        chunk);       /* Handle of chunk to dereference */
```

This routine is exactly like **LMemDeref()** above, except that the chunk is specified by its global and chunk handles.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

See Also: LMemDeref()

■ LMemFree()

```
void      LMemFree(
          optr   chunk);                /*optr of chunk to free */
```

This routine frees a chunk from an LMem heap. The chunk is added to the heap's free list. The routine is guaranteed not to compact or resize the heap; thus, all pointers within the block remain valid (except for pointers to data in the freed chunk, of course).

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

See Also: LMemFreeHandles()

■ LMemFreeHandles()

```
void      LMemFreeHandles(
          MemHandle          mh,           /* Handle of LMem heap */
          ChunkHandle        chunk);       /* Handle of chunk to free */
```

This routine is just like **LMemFree()** above, except that the chunk is specified by its global and chunk handles (instead of by an optr).

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

■ LMemGetChunkSize()

```
word      LMemGetChunkSize(
          optr   chunk);                   /* optr of subject chunk */
```

This routine returns the size (in bytes) of a chunk in an LMem heap. Since LMem chunks are dword-aligned, the chunk's size may be slightly larger than the size specified when it was allocated. The routine is guaranteed not to compact or resize the heap; thus, all pointers within the block remain valid.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

See Also: LMemGetChunkSizeHandles()

■ LMemGetChunkSizeHandles()

```
word      Routine(
          MemHandle          mh,          /* Handle of LMem heap */
          ChunkHandle        chunk);      /* Handle of chunk in question */
```

This routine is just like **LMemGetChunkSize()** above, except that the chunk is specified by its global and chunk handles (instead of by an optr).

Be Sure To: Lock the block on the global heap (unless it is fixed).

Include: lmem.h

See Also: LMemGetChunkSize()

■ LMemInitHeap()

```
void      LMemInitHeap(
          MemHandle          mh,          /* Handle of (locked or fixed)
                                         * block which will contain heap */
          LMemType           type,        /* Type of heap to create */
          LocalMemoryFlags   flags,       /* Record of LocalMemoryFlags */
          word               lmemOffset,  /* Offset of first chunk in heap (or
                                         * zero for default offset) */
          word               numHandles,  /* Size of starter handle table */
          word               freeSpace);  /* Size of first free chunk
                                         * created */
```

This routine creates an LMem heap in a global memory block. The block must be locked or fixed in memory. The routine initializes the **LMemBlockHeader**, creates a handle table, allocates a single free chunk, and turns on the HF_LMEM flag for the block. The block will be reallocated if necessary to make room for the heap. The routine takes six arguments:

mh	The memory block's handle
type	A member of the **LMemType** enumerated type, specifying the kind of block to create. For most applications, this will be LMEM_TYPE_GENERAL.
flags	A record of **LocalMemoryFlags**, specifying certain properties of the heap. Most applications will pass a null record.
lmemOffset	The offset within the block at which to start the heap. This must be larger than the size of the **LMemBlockHeader** structure which begins every heap block, or it must be zero, indicating that the heap should begin immediately after the header. Any space between the **LMemBlockHeader** and the heap is left untouched by all LMem routines.
numHandles	The number of entries to create in the block's chunk handle table. The chunk handle table will grow automatically when all entries have been used up. Applications should generally pass the constant STD_LMEM_INIT_HANDLES; they should definitely pass a positive number.
freeSpace	The amount of space to allocate to the first free chunk. Applications should generally pass the constant STD_LMEM_INIT_HEAP they should definitely pass a positive number.

To destroy an LMem heap, call **MemFree()** to free the block containing the heap.

Structures: There are two special data types used by this routine: **LMemTypes** and **LocalMemoryFlags**.

LMem heaps are created for many different purposes. Some of these purposes require the heap to have special functionality. For this reason, you must pass a member of the **LMemTypes** enumerated type to specify the kind of heap to create. The following types can be used; other types exist but should not be used with **LMemInitHeap()**.

LMEM_TYPE_GENERAL
: Ordinary heap. Most application LMem heaps will be of this type.

LMEM_TYPE_OBJ_BLOCK
: The heap will contain object instance chunks.

When an LMem heap is created, you must pass a record of flags to **LMemInitHeap()** to indicate how the heap should be treated. Most of the **LocalMemoryFlags** are only passed by system routines; all the flags available are listed below. The flags can be read by examining the **LMemBlockHeader** structure at the beginning of the block. Ordinarily, general LMem heaps will have all flags cleared.

LMF_HAS_FLAGS
: Set if the block has a chunk containing only flags. This flag is set for object blocks; it is usually cleared for general LMem heaps.

LMF_DETACHABLE
: Set if the block is an object block which can be saved to a state file.

LMF_NO_ENLARGE
: Indicates that the local-memory routines should not enlarge this block to fulfill chunk requests. This guarantees that the block will not be moved by a chunk allocation request; however, it makes these requests more likely to fail.

LMF_RETURN_ERRORS
: Set if local memory routines should return errors when allocation requests cannot be fulfilled. If the flag is not set, allocation routines will fatal-error if they cannot comply with requests.

STD_LMEM_OBJECT_FLAGS
: Not actually a flag; rather, it is the combination of LMF_HAS_FLAGS and LMF_RELOCATED. These flags are set for object blocks.

Tips and Tricks: If you want a fixed data space after the header, declare a structure whose first element is an **LMemBlockHeader** and whose other fields are for the data you will store in the fixed data space. Pass the size of this structure as the *LMemOffset* argument. You can now access the fixed data area by using the fields of the structure.

Be Sure To: Pass an offset of either zero or at least as large as **sizeof(LMemBlockHeader)**. If you pass a positive offset that is too small, the results are undefined. Lock the block on the global heap before calling this routine (unless the block is fixed).

Warnings: The block may be relocated, if its initial size is too small to accommodate the heap. This is true even for fixed blocks. If the flag LMF_NO_ENLARGE is set, the block will never be relocated; however, you must make sure it starts out large enough to accommodate the entire heap.

Routines

Include: lmem.h

See Also: LMemBlockHeader, LMemType, LocalMemoryFlags, MemAlloc(), MemFree(), VMAllocLMem()

■ LMemInsertAt()

```
void    LMemInsertAt(
        optr   chunk,               /* optr of chunk to resize */
        word   insertOffset,        /* Offset within chunk of first byte
                                   * to be added */
        word   insertCount);        /* # of bytes to add */
```

This routine inserts space in the middle of a chunk and zero-initializes the new space. The first new byte will be at the specified offset within the chunk.

Be Sure To: Lock the block on the global heap (unless it is fixed). Make sure the offset is within the specified chunk.

Warnings: This routine may resize or compact the heap; thus, all pointers to data within the block are invalidated.

You must pass an *insertOffset* that is actually within the chunk; if the offset is out-of-bounds, results are undefined.

Include: lmem.h

See Also: LMemReAlloc(), LMemDeleteAt(), LMemInsertAtHandles()

■ LMemInsertAtHandles()

```
void    LMemInsertAtHandles(
        MemHandle          mh,          /* Handle of LMem heap */
        ChunkHandle        chunk,       /* Chunk to resize */
        word               insertOffset,/* Offset within chunk of first byte
                                      * to be added */
        word               insertCount);/* # of bytes to add */
```

This routine is just like **LMemInsertAt()** above, except that the chunk is specified by its global and chunk handles (instead of by an optr).

Be Sure To: Lock the block on the global heap (unless it is fixed). Make sure the offset is within the specified chunk.

Warnings: This routine may resize or compact the heap; thus, all pointers to data within the block are invalidated.

You must pass an *insertOffset* that is actually within the chunk; if the offset is out-of-bounds, results are undefined.

Include: lmem.h

■ LMemReAlloc()

```
Boolean LMemReAlloc(
        optr   chunk,                /* optr of chunk to resize */
        word   chunkSize);           /* New size of chunk in bytes */
```

This routine resizes a chunk in an LMem heap. The heap must be in a locked or fixed block. If the routine succeeds, it returns zero. If it fails (because the heap ran out of space and could not be expanded), it returns non-zero.

If the new size is larger than the original size, extra bytes will be added to the end of the chunk. These bytes will not be zero-initialized. The heap may have to be compacted or resized to accommodate the request; thus, all pointers to data within the block are invalidated.

If the new size is smaller than the old, the chunk will be truncated. The request is guaranteed to succeed, and the chunk will not be moved; neither will the heap be compacted or resized. Thus, all pointers to other chunks remain valid. Reallocating a chunk to zero bytes is the same as freeing it.

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: As noted, if the new size is larger than the old, the heap may be compacted or resized, invalidating pointers.

Include: lmem.h

See Also: LMemReAllocHandles(), LMemInsertAt(), LMemDeleteAt()

■ LMemReAllocHandles()

```
void    LMemReAllocHandles(
        MemHandle          mh,          /* Handle of LMem heap */
        ChunkHandle        chunk,       /* Handle of chunk to resize */
        word               chunkSize);  /* New size of chunk in bytes */
```

This routine is just like **LMemReAlloc()** above, except that the chunk is specified by its global and chunk handles (instead of by an optr).

Be Sure To: Lock the block on the global heap (unless it is fixed).

Warnings: As noted, if the new size is larger than the old, the heap may be compacted or resized, invalidating pointers.

Include: lmem.h

■ LocalAsciiToFixed()

```
WWFixedAsDWord LocalAsciiToFixed(
          const char *        buffer,
          char **             parseEnd);
```

This routines converts a string like “12.345” to a fixed point number.

Include: localize.h

■ LocalCmpStrings()

```
sword     LocalCmpStrings(
          const char *        str1,
          const char *        str2,
          word                strSize);
```

This routine compares two strings to determine which comes first in a lexical (i.e. alphabetic) ordering. If the return value is negative, then *str1* is earlier than *str2*. If the return value is positive, then *str1* is later than *str2*. If the return value is zero, then the strings appear at the same place in alphabetical order.

Include: localize.h

■ LocalCmpStringsDosToGeos()

```
sword     LocalCmpStringsDosToGeos(
          const char *                  str1,
          const char *                  str2,
          word                          strSize,
          word                          defaultChar,
          LocalCmpStringsDosToGeosFlags flags);
```

This routine compares two strings to determine which comes first in lexical ordering. Either or both of these strings may be a DOS string. If the return value is negative, then *str1* is earlier than *str2*. If the return value is positive, then *str1* is later than *str2*. If the return value is zero, then the strings appear at the same place in alphabetical order.

Structures:

```
          typedef ByteFlags LocalCmpStringsDosToGeosFlags;
                 /* The following flags may be combined using | and &:
                   *LCSDTG_NO_CONVERT_STRING_2,
                   * LCSDTG_NO_CONVERT_STRING_1 */
```

Include: localize.h

■ LocalCmpStringsNoCase()

```
sword    LocalCmpStringsNoCase(
         const char *        str1,
         const char *        str2,
         word                strSize);
```

This routine compares two strings to determine which comes first in a lexical (i.e. alphabetic) ordering. The comparison used is not case-sensitive. If the return value is negative, then *str1* is earlier than *str2*. If the return value is positive, then *str1* is later than *str2*. If the return value is zero, then the strings appear at the same place in alphabetical order.

Include: localize.h

■ LocalCodePageToGeos()

```
Boolean  LocalCodePageToGeos(
         char *              str,
         word                strSize,  /* Size of the string, in bytes */
         DosCodePage         codePage,
         word                defaultChar);
```

This routine converts a DOS string to standard GEOS text using a specified code page. Any characters for which there is no GEOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalCodePageToGeosChar()

```
word     LocalCodePageToGeosChar(
         word   ch,
         DosCodePage codePage,
         word   defaultChar);
```

This routine converts a DOS character to standard GEOS text using a specified code page. Any character for which there is no GEOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalCustomFormatDateTime()

```
word      LocalCustomFormatDateTime(
          char * str,               /* Buffer to save formatted text in */
          const char *format,       /* Format string */
          const  TimerDateAndTime *dateTime);
```

This routine takes a date or time and constructs a string using a custom format.

Include: localize.h

■ LocalCustomParseDateTime()

```
word      LocalCustomParseDateTime(
          const char *        str,
          DateTimeFormat      format,
          TimerDateAndTime *  dateTime);
```

This routine parses a date and time string by comparing it with the passed **DateTimeFormat**. It fills in the fields of the **TimerDateAndTime** structure. Any fields which are not specified in the format string will be filled with -1.

If the string parses correctly, **LocalCustomParseDateTime()** returns -1. Otherwise it reutrns the offset to the start of the text which did not parse correctly.

Include: localize.h

■ LocalDistanceFromAscii()

```
WWFixedAsDword LocalDistanceFromAscii(
          const char *        buffer,
          DistanceUnit        distanceUnits,
          MeasurementTypes    measurementType);
```

This routine takes a function like “72 pt” and returns a number representing the distance. The returned answer represents the measure in points, inches, centimeters, or some other measure as specified by the passed unit.

Include: localize.h

■ LocalDistanceToAscii()

```
word      LocalDistanceToAscii( /* Length of string, including NULL */
          char *              buffer,   /*Buffer to save formatted text in */
          word                value,
          DistanceUnit        distanceUnits,
          MeasurementType     measurementType);
```

This routine takes a distance and a set of units and returns a string containing a properly formatted distance.

Include: localize.h

■ LocalDosToGeos()

```
Boolean   LocalDosToGeos(
          char * str,
          word   strSize,
          word   defaultChar);
```

Convert a DOS string to GEOS text. Any characters for which there is no GEOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalDosToGeosChar()

```
word      LocalDosToGeosChar(
          word   ch,
          word   defaultChar);
```

Convert a DOS character to GEOS text. Any characters for which there is no GEOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalDowncaseChar()

```
word      LocalDowncaseChar(
          word   ch);
```

Return the lower case equivalent, if any, of the passed character.

Include: localize.h

■ LocalDowncaseString()

```
void      LocalDowncaseString(
          char * str,
          word   size);            /* Size of string, in bytes */
```

Convert the passed string to its all lower case equivalent.

Include: localize.h

■ LocalFixedToAscii()

```
void      LocalFixedToAscii(
          char * buffer,
          WWFixedAsDWord value,
          word   fracDigits);
```

This routine returns the ASCII expression of a fixed point number.

Include: localize.h

■ LocalFormatDateTime()

```
word      LocalFormatDateTime( /* Length of returned string */
          char *                str,
          DateTimeFormat        format,
          const TimerDateAndTime *dateTime);
```

This routine returns the string (e.g. “9:37”) corresponding to the passed DateAndTime.

Include: localize.h

■ LocalGeosToCodePage()

```
Boolean   LocalGeosToCodePage(
          char *             str,
          word               strSize,
          DosCodePage        codePage,
          word               defaultChar);
```

Convert a GEOS string to DOS text, using the specified code page. Any characters for which there is no DOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalGeosToCodePageChar()

```
word      LocalGeosToCodePageChar(
          word                ch,
          DosCodePage         codePage,
          word                defaultChar);
```

Convert a GEOS character to DOS text, using the specified code page. Any character for which there is no DOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalGeosToDos()

```
Boolean   LocalGeosToDos(
          char * str,
          word   strSize,
          word   defaultChar);
```

Convert a GEOS string to DOS text. Any characters for which there is no DOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalGeosToDosChar()

```
word      LocalGeosToDosChar(
          word   ch,
          word   defaultChar);
```

Convert a GEOS character to DOS text. Any character for which there is no DOS equivalent will be replaced by the passed default character.

Include: localize.h

■ LocalGetCodePage()

```
DosCodePage LocalGetCodePage(void);
```

This routine returns the current code page, used by DOS to handle international character sets.

Include: localize.h

■ LocalGetCurrencyFormat()

```
void      LocalGetCurrencyFormat(
          LocalCurrencyFormat *   buf,
          char *                  symbol);
```

This routine returns the current currency format and symbol.

Include: localize.h

■ LocalGetDateTimeFormat()

```
void      LocalGetDateTimeFormat(
          char *              str,
          DateTimeFormat      format);
```

This routine returns the user’s preferred time and date formats.

Include: localize.h

■ LocalGetDefaultPrintSizes()

```
void      LocalGetDefaultPrintSizes(
          DefaultPrintSizes * sizes);
```

This routine returns the system’s default page and document size.

Include: localize.h

■ LocalGetMeasurementType()

```
MeasurementTypes LocalGetMeasurementType(void);
```

This routine returns the user preference between US and metric measurement systems.

Include: localize.h

■ LocalGetNumericFormat()

```
void      LocalGetNumericFormat(
          LocalNumericFormat *buf);
```

This routine returns the user’s preferred format for numbers.

Include: localize.h

■ LocalGetQuotes()

```
void      LocalGetQuotes(
          LocalQuotes *      quotes);
```

This routine returns the user’s preferred quote marks.

Include: localize.h

■ LocalIsAlpha()

```
Boolean   LocalIsAlpha(
          word   ch);
```

This routine returns *true* if the passed character is alphabetic.

Include: localize.h

■ LocalIsAlphaNumeric()

```
Boolean   LocalIsAlphaNumeric(
          word   ch);
```

This routine returns *true* if the passed character is alphanumeric.

Include: localize.h

■ LocalIsControl()

```
Boolean   LocalIsControl(
          word   ch);
```

This routine returns *true* if the passed character is a control character.

Include: localize.h

■ LocalIsDateChar()

```
Boolean   LocalIsDateChar(
          word   ch);
```

This routine returns *true* if the passed character could be part of a date or time.

Include: localize.h

■ LocalIsDigit()

```
Boolean   LocalIsDigit(
          word   ch);
```

This routine returns *true* if the passed character is a decimal digit.

Include: localize.h

■ LocalIsDosChar()

```
Boolean  LocalIsDosChar(
         word   ch);
```

This routine returns *true* if the passed character is part of the DOS character set.

Include: localize.h

■ LocalIsGraphic()

```
Boolean  LocalIsGraphic(
         word   ch);
```

This routine returns *true* if the passed character is displayable.

Include: localize.h

■ LocalIsHexDigit()

```
Boolean  LocalIsHexDigit(
         word   ch);
```

This routine returns *true* if the passed character is a hexadecimal digit.

Include: localize.h

■ LocalIsLower()

```
Boolean  LocalIsLower(
         word   ch);
```

This routine returns *true* if the passed character is a lower case alphabetic character.

Include: localize.h

■ LocalIsNumChar()

```
Boolean  LocalIsNumChar(
         word   ch);
```

This routine returns *true* if the passed character is a number or part of the number format.

Include: localize.h

■ LocalIsPrintable()

```
Boolean  LocalIsPrintable(
         word   ch);
```

This routine returns *true* if the passed character is printable (i.e. takes up a space when printing).

Include: localize.h

■ LocalIsPunctuation()

```
Boolean  LocalIsPunctuation(
         word   ch);
```

This routine returns *true* if the passed character is a punctuation mark.

Include: localize.h

■ LocalIsSpace()

```
Boolean  LocalIsSpace(
         word   ch);
```

This routine returns *true* if the passed character is whitespace.

Include: localize.h

■ LocalIsSymbol()

```
Boolean  LocalIsSymbol(
         word   ch);
```

This routine returns *true* if the passed character is a symbol.

Include: localize.h

■ LocalIsTimeChar()

```
Boolean  LocalIsTimeChar(
         word   ch);
```

This routine returns *true* if the passed character is a number or part of the user's time format.

Include: localize.h

■ LocalIsUpper()

```
Boolean   LocalIsUpper(
          word   ch);
```

This routine returns *true* if the passed character is an upper case alphabetic character.

Include: localize.h

■ LocalLexicalValue()

```
word      LocalLexicalValue(
          word   ch);
```

This routine returns the passed character's lexical value, useful when trying to sort strings alphabetically.

Include: localize.h

■ LocalLexicalValueNoCase()

```
word      LocalLexicalValueNoCase(
          word   ch);
```

This routine returns the passed character's case-insensitive lexical value, useful when trying to sort strings alphabetically.

Include: localize.h

■ LocalParseDateTime()

```
Boolean   LocalParseDateTime(
          const char *        str,
          DateTimeFormat      format,
          TimerDateAndTime *  dateTime);
```

This routine takes a string describing a date or time (e.g. “9:37”) and parses it using the passed format.

Include: localize.h

■ LocalSetCurrencyFormat()

```
void      LocalSetCurrencyFormat(
          const LocalCurrencyFormat *buf,
          const char *              symbol);
```

This routine changes the stored preferred currency format.

Include: localize.h

■ LocalSetDateTimeFormat()

```
void      LocalSetDateTimeFormat(
          const char *        str,
          DateTimeFormat      format);
```

This routine changes the stored preferred time and date format.

Include: localize.h

■ LocalSetDefaultPrintSizes()

```
void      LocalSetDefaultPrintSizes(
          const DefaultPrintSizes *sizes);
```

This routine changes the stored preferred default page and document sizes.

Include: localize.h

■ LocalSetMeasurementType()

```
void      LocalSetMeasurementType(
          MeasurementTypes meas);
```

This routine changes the stored preferred measurement type.

Include: localize.h

■ LocalSetNumericFormat()

```
void      LocalSetNumericFormat(
          const LocalNumericFormat * buf);
```

This routine changes the stored preferred number format.

Include: localize.h

■ LocalSetQuotes()

```
void      LocalSetQuotes(
          const LocalQuotes * quotes);
```

This routine changes the stored preferred quote marks.

Include: localize.h

■ LocalStringLength()

```
word      LocalStringLength(
          const char *        str);
```

This routine returns the length (in characters) of a null-terminated string (not counting the null), even for multibyte character sets.

Include: localize.h

■ LocalStringSize()

```
word      LocalStringSize(
          const char *        str);
```

This routine returns the size (in bytes) of a null-terminated string.

Include: localize.h

■ LocalUpcaseChar()

```
word      LocalUpcaseChar(
          word   ch);
```

This routine returns the upper case equivalent, if any, of the passed character.

Include: localize.h

■ LocalUpcaseString()

```
void      LocalUpcaseString(
          char * str,
          word   size);
```

This routine converts the passed string to its all upper case equivalent.

Include: localize.h

Routines

■ MakeWWFixed()

```
WWFixed MakeWWFixed(number);
```

This macro casts a floating-point or integer number to a **WWFixed** value.

Include: geos.h

■ malloc()

```
void     * malloc(
         size_t             blockSize);   /* # of bytes to allocate*/
```

The **malloc()** family of routines is provided for Standard C compatibility. If a geode needs a small amount of fixed memory, it can call one of the routines. The kernel will allocate a fixed block to satisfy the geode's **malloc()** requests; it will allocate memory from this block. When the block is filled, it will allocate another fixed malloc-block. When all the memory in the block is freed, the memory manager will automatically free the block.

When a geode calls **malloc()**, a section of memory of the size specified will be allocated out of its malloc-block, and the address of the start of the memory will be returned. The memory will *not* be zero-initialized. If the request cannot be satisfied, **malloc** will return a null pointer. The memory is guaranteed not to be moved until it is freed (with **free()**) or resized (with **realloc()**). When GEOS shuts down, all fixed blocks are freed, and any memory allocated with **malloc()** is lost.

Using too many fixed blocks degrades the memory manager's performance, slowing the whole system. For this reason, applications should not use **malloc**-family routines if they can possibly be avoided. They are provided only to simplify porting of existing programs; however, applications should make every effort to use the GEOS memory management and LMem routines instead. If you must use the **malloc**-family routines, use them sparingly, and free the memory as quickly as possible.

Tips and Tricks: You can allocate memory in another geode's malloc-block by calling **GeoMalloc()**. However, that block will be freed when the other geode exits.

Warnings: All memory allocated with **malloc()** is freed when GEOS shuts down.

Include: stdlib.h

See Also: calloc(), free(), GeoMalloc(), realloc()

■ ManufacturerFromFormatID

```
word      ManufacturerFromFormatID(id);
          ClipboardItemFormatID id;
```

This macro extracts the word-sized manufacturer ID (of type **ManufacturerIDs**) from a **ClipboardInfoFormatID** argument.

■ MemAlloc()

```
MemHandle MemAlloc(
          word              byteSize,    /* Size of block in bytes */
          HeapFlags         hfFlags,     /* Type of block */
          HeapAllocFlags    haFlags);    /* How to allocate block */
```

This routine allocates a global memory block and creates an entry for it in the global handle table. The properties of the block are determined by the **HeapFlags** record passed; the way the block should be allocated is determined by the **HeapAllocFlags** record. Both sets of flags are described below. The routine returns the block's handle. If it could not allocate the block, it returns a null handle. The block allocated may be larger than the size requested, as the block size is rounded up to the next even paragraph (one paragraph equals sixteen bytes).

HeapFlags are stored in the block's handle table entry. They can be retrieved with the routine **MemGetInfo()**; some of them can be changed with the routine **MemModifyFlags()**. The following flags are available:

HF_FIXED
: The block will not move from its place in the global heap until it is freed. If this flag is off, the memory manager may move the block while it is unlocked. If the flag is on, the block may not be locked, and HF_DISCARDABLE and HF_SWAPABLE must be off.

HF_SHARABLE
: The block may be locked by threads belonging to geodes other than the block's owner.

HF_DISCARDABLE
: The block may be discarded when unlocked.

HF_SWAPABLE
: The block may be swapped to extended/expanded memory or to the disk swap space when it is unlocked.

HF_LMEM
: The block contains a local memory heap. This flag is set automatically by **LMemInitHeap()** and **VMAllocLMem()**; applications should not need to set this flag.

HF_DISCARDED
The memory manager turns this bit on when it discards a block. The bit is turned off when the block is reallocated.

HF_SWAPPED
The memory manager turns this bit on when it swaps a block to extended/expanded memory or to the disk swap space. It turns the bit off when it swaps the block back into the global heap.

HeapAllocFlags indicate how the block should be allocated and initialized. They are not stored and can not be retrieved. Some of the flags can be passed with **MemReAlloc()**. The following flags are available:

HAF_ZERO_INIT
The memory manager should initialize the block to null bytes. This flag may be passed to **MemReAlloc()** to cause new memory to be zero-initialized.

HAF_LOCK
The memory manager should lock the block after allocating it. To get the block's address, call **MemDeref()**. This flag may be passed to **MemReAlloc()**.

HAF_NO_ERR
The memory manager should not return errors. If it cannot allocate block, GEOS will crash. Use of this flag is strongly discouraged. This flag may be passed to **MemReAlloc()**.

HAF_UI
If both HAF_OBJECT_RESOURCE and HAF_UI are set, this block will be run by the application's UI thread.

HAF_READ_ONLY
The block's data will not be modified. Useful for the debugger.

HAF_OBJECT_RESOURCE
This block will be an object block.

HAF_CODE
This block contains executable code.

HAF_CONFORMING
If the block contains code, the code may be run by a less privileged entity. If the block contains data, the data may be accessed or altered by a less privileged entity.

Include: heap.h

See Also: MemAllocSetOwner(), MemReAlloc(), MemDeref()

■ MemAllocLMem()

```
MemHandle MemAllocLMem(
         LMemType          type,        /* type of LMem block */
         word              headerSize); /* size of header structure */
```

This routine allocates and initializes a local memory block; it can be used to simplify this procedure from the two-step process of **MemAlloc()** followed by **LMemInitHeap()**. Pass an LMem type indicating what will be stored in the block, along with the size of the header structure to use. If the block is to have the standard header, pass zero in *headerSize*.

This routine returns the handle of the unlocked, newly allocated block. The block will contain two LMem handles and 64 bytes allocated for the LMem heap.

Include: lmem.h

See Also: LMemInitHeap()

■ MemAllocSetOwner()

```
MemHandle MemAllocSetOwner(
         GeodeHandle       owner,       /* Handle of block's owner */
         word              byteSize,    /* Size of block in bytes */
         HeapFlags         hfFlags,     /* Type of block */
         HeapAllocFlags    haFlags);    /* How to allocate block */
```

This routine is the same as **MemAlloc()** except that you can specify the owner of the global memory block created.

Include: heap.h

See Also: MemAlloc()

■ MemDecRefCount()

```
void     MemDecRefCount(
         MemHandle         mh);         /* handle of affected block */
```

This routine decrements the reference count of a global memory block (the reference count is stored in *HM_otherInfo*). If the reference count reaches zero, **MemDecRefCount()** will free the block.

Warnings: This routine assumes that a reference count is stored in *HM_otherInfo*. You may only use this routine if the block has had a reference count set up with **MemInitRefCount()**.

Include: heap.h

■ MemDeref()

```
void        * MemDeref(
            MemHandle           mh);   /* Handle of locked block to dereference */
```

This routine takes one argument, the handle of a global memory block; it returns the address of the block on the global heap. If the block has been discarded, or if the handle is not a memory handle, it returns a null pointer. It gets this information by reading the block's handle table entry; it does not need to actually access the block.

Note that if the handle is of an unlocked, moveable block, **MemDeref()** will return the block's address with out any warning; however, the address will be unreliable, since the memory manager can move the block at any time.

Include: heap.h

Tips and Tricks: This is very useful when you allocate a fixed or locked block, and need to get the block's address without calling **MemLock()**.

Warnings: This routine, if given an unlocked, moveable block, will return the pointer without a warning, even though that block may move at any time.

See Also: MemGetInfo(), MemModifyFlags()

■ MemDowngradeExclLock()

```
void        MemDowngradeExclLock(
            MemHandle           mh);      /* handle of affected block */
```

An application that has an exclusive lock on a block may downgrade it to a shared lock with this routine. It does not otherwise affect the block.

Include: heap.h

■ MemFree()

```
void        MemFree(
            MemHandle           mh);      /* handle of block to be freed */
```

This routine frees a global memory block. The block can be locked or unlocked.

Include: heap.h

Warnings: The routine does not care whether other threads have locked the block. If you try to free a bad handle, routine may fatal-error.

■ MemGetInfo()

```
word      MemGetInfo( /* return value depends on flag passed */
          MemHandle           mh,       /* Handle of block to get info about */
          MemGetInfoType      info);    /* Type of information to get */
```

MemGetInfo() is a general-purpose routine for getting information about a global memory block. It gets the information by looking in the block's handle table entry; it does not need to access the actual block. It returns a single word of data; the meaning of that word depends on the value of the **MemGetInfoType** variable passed. The following types are available:

MGIT_SIZE Return value is size of the block in bytes. This may be larger than the size originally requested, as blocks are allocated along paragraph boundaries.

MGIT_FLAGS_AND_LOCK_COUNT
The upper byte of the return value is the current **HeapFlags** record for the block. The lower byte is the number of locks currently on the block.

MGIT_OWNER_OR_VM_FILE_HANDLE
If the block is part of a VM file, return value is the VM file handle. Otherwise, return value is the GeodeHandle of the owning thread.

MGIT_ADDRESS
Return value is block's segment address on the global heap, or zero if block has been discarded. If block is unlocked and moveable, address may change without warning. Ordinarily, **MemDeref()** is preferable.

MGIT_OTHER_INFO
Returns the value of the *HM_otherInfo* word. This word is used in different ways for different types of handles.

MGIT_EXEC_THREAD
Returns the ThreadHandle of the thread executing this block, if any.

Include: heap.h

Warnings: If the handle is not a global memory block handle, results are unpredictable (the routine will read inappropriate data from the handle table entry).

See Also: MemDeref(), MemModifyFlags(), MemModifyOtherInfo(), HandleModifyOwner()

■ MemIncRefCount()

```
void        MemIncRefCount(
            MemHandle           mh);        /* handle of affected block */
```

This routine increments the reference count of a global memory block (the reference count is stored in *HM_otherInfo*).

Warnings: This routine assumes that a reference count is stored in *HM_otherInfo*. You may only use this routine if the block has had a reference count set up with **MemInitRefCount()**.

Include: heap.h

■ MemInitRefCount()

```
void        MemInitRefCount(
            MemHandle           mh,           /* handle of affected block */
            word                count);       /* initial reference count */
```

This routine sets up a reference count for the specified global memory block. The passed count is stored in the *HM_otherInfo* field of the block's handle-table entry.

Warnings: This routine overwrites the *HM_otherInfo* field. Since the semaphore routines (**HandleP()** and **HandleV()** and the routines which use them) use this field, you cannot use both the semaphore routines and the reference count routines on the same block.

Include: heap.h

■ MemLock()

```
void        * MemLock(
            MemHandle           mh);        /* Handle of block to lock */
```

This routine locks a global memory block on the global heap. If the block is swapped, the memory manager swaps it back into the global heap; it then increments the lock count (up to a maximum of 255). The block will not be moved, swapped, or discarded until the lock count reaches zero. This routine returns a pointer to the start of the block, or a null pointer if block has been discarded. To get the address of a block without locking it, use **MemDeref()**.

Include: heap.h

Warnings: If you try to lock a bad handle, the routine may fatal-error. This routine does not check for synchronization problems; if the block is used by several threads, you should use the synchronization routines.

Never Use Situations:

Never lock a fixed block.

See Also: MemDeref()

MemLockExcl()

```
void      * MemLockExcl(
          MemHandle         mh);      /* Handle of block to grab */
```

If several different threads will be accessing the same global memory block, they should use data-access synchronization routines. **MemLockExcl()** belongs to one such set of routines. Often, several threads will need access to the same block; however, most of the time, they will not need to change the block. There is no synchronization problem if several threads read the same block at once, as long as none of them alters the block (by resizing it, writing to it, etc.) However, if a thread needs to change a block, no other thread should have access at that time.

The routines **MemLockExcl()**, **MemLockShared()**, **MemUnlockShared()**, and **MemUnlockExcl()** take advantage of this situation. They maintain a queue of threads requesting access to a block. When the block is not being used, they awaken the highest priority thread on the queue. If that thread requested exclusive access, the other threads sleep until it relinquishes access (via **MemUnlockExcl()**). If it requested shared access, the routines awaken every other thread which requested shared access; the other threads on the queue will sleep until every active thread relinquishes access (via **MemUnlockShared()**).

MemLockExcl() requests exclusive access to the block. If the block is not being accessed, the routine will grab exclusive access for the block, lock the block, and return the block's address. If the block is being accessed, the routine will sleep on the queue until it can get access; it will then awaken, lock the block, and return its address.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all routines accessing the block get access with **MemLockExcl()** or **MemLockShared()**. The routines use the block's *HM_otherInfo* word; you must not alter it. When you are done accessing the block, make sure to relinquish access by calling **MemUnlockExcl()**.

Warnings: If a thread calls **MemLockExcl()** when it already has shared or exclusive access, it will deadlock; it will sleep until access is relinquished, but it cannot relinquish access while it is sleeping. If you try to grab a block which is owned by a different geode and is non-sharable, the routine will fatal-error.

Never Use Situations:
Never use **MemLockExcl()** or **MemLockShared()** on a fixed block. It will attempt to lock the block, and fixed blocks cannot be locked. Instead, use the **HandleP()** and **HandleV()** routines.

See Also: MemLockShared(), MemUnlockExcl(), MemUnlockShared()

■ MemLockFixedOrMovable()

```
void      * MemLockFixedOrMovable(
          void   * ptr);         /* virtual segment */
```

Given a virtual segment, this routine locks it (if it was movable). A virtual segment is an opaque pointer to a block that an application views as locked or fixed—the memory manager can actually swap locked or fixed blocks and will designate them as virtual segments.

Include: heap.h

■ MemLockShared()

```
void      * MemLockShared(
          MemHandle          mh);      /* Handle of block to grab */
```

MemLockShared() requests shared access to the passed block. If the block is not being accessed, or if it is held for shared access and the queue is empty, the routine gets access, locks the block, and returns the block's address. Otherwise it sleeps on the queue until the shared requests are awakened; it then locks the block and returns the block's address.

Include: heap.h

Be Sure To: Make sure that all routines accessing the block get access with **MemLockExcl()** or **MemLockShared()**. The routines use the block's *HM_otherInfo* word; you must not alter it. When you are done accessing the block, make sure to relinquish access by calling **MemUnlockExcl()**.

Warnings: If a thread calls **MemLockShared()** when it already has exclusive access, it will deadlock; it will sleep until access is relinquished, but it cannot relinquish access while it is sleeping. The thread must be careful not to take actions which could change the block, such as resizing it or writing to it. The routine will not enforce this. If you try to grab a block which is owned by a different geode and is non-sharable, the routine will fatal-error.

Never Use Situations:

Never use **MemLockExcl()** or **MemLockShared()** on a fixed block. It will attempt to lock the block, and fixed blocks cannot be locked. Instead, use the **HandleP()** and **HandleV()** routines.

See Also: MemLockExcl(), MemUnlockExcl(), MemUnlockShared()

■ MemModifyFlags()

```
void      MemModifyFlags(
          MemHandle          mh,               /* Handle of block to modify */
          HeapFlags          bitsToSet,        /* HeapFlags to turn on */
          HeapFlags          bitsToClear);     /* HeapFlags to turn off */
```

MemModifyFlags() changes the **HeapFlags** record of the global memory block specified. Not all flags can be changed after the block is created; only the flags HF_SHARABLE, HF_DISCARDABLE, HF_SWAPABLE, and HF_LMEM can be changed.

The routine uses the handle table entry of the block specified; it does not need to look at the actual block. The routine performs normally whether or not the block is locked, fixed, or discarded.

Include: heap.h

Warnings: If the handle is not a global memory handle, results are unpredictable; the routine will change inappropriate bits of the handle table entry.

See Also: MemGetInfo(), HandleModifyOwner(), MemModifyOtherInfo()

■ MemModifyOtherInfo()

```
void      MemModifyOtherInfo(
          MemHandle          mh,            /* Handle of block to modify */
          word               otherInfo);    /* New value of HM_otherInfo word */
```

Use this routine to change the value of the global memory block's *HM_otherInfo* word. Some blocks need this word left alone; for example, data-access synchronization routines use this word.

Include: heap.h

See Also: MemGetInfo(), MemModifyFlags(), HandleModifyOwner()

Routines

■ MemOwner()

```
GeodeHandle MemOwner(
          MemHandle           mh);      /* handle of block queried */
```

This routine returns the owning geode of the passed block. If the block belongs to a VM file, the owner of the VM file will be returned (unlike **MemInfo()**, which returns the VM file handle).

Include: heap.h

■ MemPLock()

```
void      * MemPLock(
          MemHandle           mh);   /* Handle of block to lock */
```

If several different threads will be accessing the same global memory block, they need to make sure their activities will not conflict. The way they do that is to use synchronization routines to get control of a block. **MemPLock()** is part of one set of synchronization routines.

If the threads are using the **MemPLock()** family, then whenever a thread needs access to the block in question, it can call **MemPLock()**. This routine calls **HandleP()** to get control of the block; it then locks the block and returns its address. If the block has been discarded, it grabs the block and returns a null pointer; you can then reallocate the block. When the thread is done with the block, it should release it with **MemUnlockV()**.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads accessing the block use **MemPLock()** and/or **HandleP()** to grab the block. These routines use the *HM_otherInfo* field of the block's handle table entry; do not alter this field. Release the block with **HandleV()** or **MemUnlockV()** when you are done with it.

Warnings: If a thread calls **MemPLock()** when it already has control of the block, it will deadlock. **MemThreadGrab()** avoids this conflict. If you try to grab a non-sharable block owned by another thread, **MemPLock()** will fatal-error.

Never Use Situations:

Never use **MemPLock()** with a fixed block. It will try to lock the block, and fixed blocks cannot be locked. Instead, use **HandleP()**.

See Also: HandleP(), MemUnlockV(), HandleV()

■ MemPtrToHandle()

```
MemHandle MemPtrToHandle(
         void   * ptr);         /* pointer to locked block */
```

This routine returns the global handle of the locked block.

Include: heap.h

■ MemReAlloc()

```
MemHandle MemReAlloc(
         MemHandle          mh,             /* Handle of block */
         word               byteSize,       /* New size of the block */
         HeapAllocFlags     heapAllocFlags); /* How to reallocate block */
```

This routine reallocates a global memory block. It can be used to resize a block; it can also be used to reallocate memory for a discarded block. Locked and fixed blocks can be reallocated; however, they may move on the global heap, so all pointers within the block must be adjusted. Note, however, that if the new size is smaller than the old size, the block is guaranteed not to move. The reallocated block may be larger than the size requested, as the block size is rounded up to the next even paragraph (one paragraph equals sixteen bytes).

The routine is passed a record of **HeapAllocFlags**. Only the flags HAF_ZERO_INIT, HAF_LOCK, and HAF_NO_ERR may be passed.

Include: heap.h

Warnings: If HAF_LOCK is passed, the lock count will be incremented even if the block is already locked by this thread. The routine does not care whether the block has been locked by another thread (possibly belonging to another geode); thus, if the block is being used by more than one thread, it is important to use the synchronization routines.

See Also: MemAlloc(), MemDeref()

■ MemThreadGrab()

```
void       * MemThreadGrab(
           MemHandle          mh);      /* Handle of block to grab */
```

MemThreadGrab() is used in conjunction with **MemThreadGrabNB()** and **MemThreadRelease()** to maintain data-access synchronization. If several threads will all have access to the same global memory block, they should use data-acess synchronization routines to make sure that their activities do not conflict. If a thread uses **MemThreadGrab()** and no other thread has grabbed the block in question, the routine will increment the "grab count," lock the block, and return its address. It can do this even if the calling thread has already grabbed the block. If another thread has grabbed the block, **MemThreadGrab()** will put the calling thread in a queue to get the block; the thread will sleep until it gets the block, then **MemThreadGrab()** will grab the block, lock it, and return its address.

If the block has been discarded, **MemThreadGrab()** grabs the block and returns a null pointer; you can then reallocate memory for the block.

Include: heap.h

Be Sure To: Make sure that all threads using the block use the **MemThread...()** routines to access it (not other data-acess synchronization routines). Do not change the *HM_otherInfo* word of the block's handle table entry (the routines use that word as a semaphore).

Warnings: If you try to grab a block which is owned by a different geode and is non-sharable, the routine will fatal-error.

Never Use Situations:
Never use **MemThreadGrab()** with a fixed block. It will try to lock the block, and fixed blocks cannot be locked. If you need data-access synchronization for a fixed block, use the **HandleP()** and **HandleV()** routines.

See Also: MemThreadGrabNB(), MemThreadRelease()

■ MemThreadGrabNB()

```
void       * MemThreadGrabNB(
           MemHandle          mh); /* handle of block to grab */
```

This is a data-synchronization routine to be used in conjunction with **MemThreadGrab()** and **MemThreadRelease()**. It is exactly the same as **MemThreadGrab()**, except that if it cannot grab the global memory block because another thread has it, the routine returns an error instead of blocking.

If successful, **MemThreadGrabNB()** returns a pointer to the block. If the block has been discarded, it grabs the block and returns a null pointer; you can then reallocate memory for the block. If the block has been grabbed by another thread, **MemThreadGrab()** returns the constant BLOCK_GRABBED.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads using the block use the **MemThread...()** routines to access the block (not other data-access synchronization routines). Do not change the *HM_otherInfo* word of the block's handle table entry (the routines use that word as a semaphore).

Warnings: If you try to grab a block that is owned by a different geode and is non-sharable, the routine will fatal-error.

Never Use Situations:
Never use **MemThreadGrabNB()** with a fixed block. It will try to lock the block, and fixed blocks cannot be locked. If you need synchronization for a fixed block, use the **HandleP()** and **HandleV()** routines.

See Also: MemThreadGrab(), MemThreadRelease()

■ MemThreadRelease()

```
void        MemThreadRelease(
            MemHandle           mh); /* handle of locked block to release */
```

Use this routine to release a global memory block which you have grabbed with **MemThreadGrab()** or **MemThreadGrabNB()**. The routine decrements the grab count; if the grab count reaches zero, the routine unlocks the block.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads using the block use the **MemThread...()** routines to access the block (not other data-access synchronization routines). Do not change the *HM_otherInfo* word of the block's handle table entry (the routines use that word as a semaphore). Make sure to release the block once for every time you grab it; the block is not unlocked until each of your grabs is released.

Warnings: If you try to release a block that you have not successfully grabbed, the routine will fatal-error.

See Also: MemThreadGrab(), MemThreadGrabNB()

■ MemUnlock()

```
void      MemUnlock(
          MemHandle          mh);   /* Handle of block to unlock */
```

This routine decrements the lock count of the indicated block. If the lock count reaches zero, the block becomes unlocked (it can be moved, swapped, or discarded). Do not try to unlock a block that has not been locked.

Include: heap.h

■ MemUnlockExcl()

```
void      MemUnlockExcl(
          memHandle          mh);      /* Handle of block to release */
```

If a thread has gained access to a block with **MemLockExcl()**, it should release the block as soon as it can. Until it does, no other thread can access the block for either shared or exclusive access. It can release the block by calling **MemUnlockExcl()**. This routine unlocks the block and releases the thread's access to it. If there is a queue for this block, the highest-priority thread waiting will be awakened, as described in **MemLockExcl()**.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all routines accessing the block get access with **MemLockExcl()** or **MemLockShared()**. The routines use the block's

HM_otherInfo word; you must not alter it. Call this routine while the block is still locked; it will call **MemUnlock()** to unlock the block.

Warnings: If you call this routine on a block which you have not gained access to, it may fatal-error.

See Also: MemLockExcl(), MemLockShared(), MemUnlockShared()

■ MemUnlockFixedOrMovable()

```
void    MemUnlockFixedOrMovable(
        void   * ptr);          /* virtual segment */
```

This routine unlocks a previously locked, movable virtual segment. Do not call this routine with normal locked or fixed blocks; only call it for those blocks locked with **MemLockFixedOrMovable()**.

Include: heap.h

■ MemUnlockShared()

```
void    MemUnlockShared(
        MemHandle          mh);   /* Handle of block to release */
```

If a thread has gained access to a block with **MemLockShared()**, it should release the block as soon as it can. Until it does, no thread can be awakened from the queue. It can release the block by calling **MemUnlockShared()**. This routine calls **MemUnlock()**, decrementing the block's lock count; it then releases the thread's access to it. If no other thread is accessing the block and there is a queue for this block, the highest-priority thread waiting will be awakened, as described in **MemLockExcl()**.

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all routines accessing the block get access with **MemLockExcl()** or **MemLockShared()**. These routines use the block's *HM_otherInfo* word; you must not alter it. Call this routine while the block is still locked; it will call **MemUnlock()** to unlock the block.

Warnings: If you call this routine on a block which you have not gained access to, it may fatal-error.

Routines

See Also: MemLockExcl(), MemLockShared(), MemUnlockExcl()

■ MemUnlockV()

```
void        MemUnlockV(
            MemHandle           mh);       /* Handle of block to release */
```

This routine unlocks a block with **MemUnlock()**, then releases its semaphore with **HandleV()**. Do not use this routine unless the block's semaphore was grabbed and the block locked (typically with the **MemPLock()** routine).

Include: heap.h

Tips and Tricks: You can find out if the block is being accessed by looking at the *HM_otherInfo* word (with **MemGetInfo()**). If *HM_otherInfo* equals one, the block is not grabbed; if it equals zero, it is grabbed, but no threads are queued; otherwise, it equals the handle of the first thread queued.

Be Sure To: Make sure that all threads accessing the block use **HandleP()** or **MemPLock()** to access the thread. These routines use the *HM_otherInfo* field of the handle table entry; do not alter this field.

Warnings: Do not use this on a block unless you have grabbed it. The routine does not check to see that you have grabbed the thread; it just clears the semaphore and returns.

Never Use Situations:

Never use this routine to release a fixed block. It will try to unlock the block; fixed blocks cannot be locked or unlocked. Instead, call **HandleV()** directly.

See Also: MemPLock(), HandleP(), HandleV()

■ MemUpgradeSharedLock()

```
void        * MemUpgradeSharedLock(
            MemHandle           mh);       /* handle of locked block */
```

This routine upgrades a shared lock on the block to an exclusive lock, as if the caller had used MemLockExcl(). If other threads have access to the block, the caller will sleep in the access queue until it can gain exclusive access.

This routine returns the pointer of the locked block because, if the caller sleeps in the queue, the memory block could move between the call and the granting of access.

Include: heap.h

See Also: MemLockExcl(), MemLockShared(), MemDowngradeExclLock()

■ MessageSetDestination()

```
void      MessageSetDestination(
          EventHandle        event,    /* handle of the event to be modified */
          optr               dest);    /* new destination for the event */
```

This routine sets the destination of an event to the optr passed.

Include: object.h

■ NameArrayAdd()

```
word      NameArrayAdd(
          optr                arr,          /* optr of name array */
          const char          * nameToAdd,  /* Name of new element */
          word                nameLength,   /* Length of name; pass zero if
                                           * name string is null-terminated */
          NameArrayAddFlags   flags,        /* Described below */
          const  void         * data);      /* Copy this data to new element */
```

This routine creates a new element in a name array, copying the passed name and data into the new element. If no element with the passed name exists, **NameArrayAdd()** will create the element and return its token. If an element with the same name already exists, the existing element's reference count will be incremented and its token will be returned. The routine takes the following arguments:

array The optr of the name array.

nameToAdd
The name of the new element. This string may contain nulls.

nameLength
The length of the name string, in bytes. If you pass zero, **NameArrayAdd()** will assume the string is null-terminated.

flags A record of **NameArrayAddFlags**, described below.

data The data to copy into the new element.

Structures: The argument is passed a set of **NameArrayAddFlags**. Only one flag is currently defined:

NAAF_SET_DATA_ON_REPLACE
If an element with the specified name exists and this flag is set, the data passed will be copied into the data area of the existing element. If this flag is not set, the existing element will not be changed.

Warnings: This routine may resize the name array; therefore, all pointers to the LMem heap are invalidated.

Include: chunkarr.h

■ NameArrayAddHandles()

```
dword     NameArrayAddHandles(
          MemHandle          mh,           /* Handle of LMem heap */
          ChunkHandle        arr,          /* Chunk handle of name array */
          const char *       nameToAdd,    /* Name of new element */
          word               nameLength,   /* Length of name; pass zero if
                                         * name string is null-terminated */
          NameArrayAddFlags  flags,        /* Described below */
          const  void *      data);        /* Copy this data to new element */
```

This routine is exactly like **NameArrayAdd()** above, except that the name array is specified by its global and chunk handles (instead of with an optr).

Warnings: This routine may resize the name array; therefore, all pointers to within the LMem heap are invalidated.

Include: chunkarr.h

■ NameArrayChangeName()

```
void      NameArrayChangeName(
          optr               array,    /* optr of name array */
          word               token,    /* Token of element to be changed */
          const char *       newName,  /* New name for element */
          word               nameLength); /* Length of name in bytes; pass
                                         * zero if name string is
                                         * null-terminated */
```

This routine changes the name of an element in a name array.

Warnings: If the new name is longer than the old, the chunk will be resized, invalidating all pointers to within the LMem heap.

Include: chunkarr.h

■ NameArrayChangeNameHandles()

```
dword     NameArrayChangeNameHandles(
          MemHandle          mh,           /* Handle of LMem heap */
          ChunkHandle        array,        /* Chunk handle of name array */
          word               token,        /* Token of element to be changed */
          const char *       newName,      /* New name for element */
          word               nameLength);  /* Length of name in bytes; pass
                                          * zero if name string is
                                          * null-terminated */
```

This routine is exactly like **NameArrayChangeName()** above, except that the name array is specified by its global and chunk handles (instead of with an optr).

Warnings: If the new name is longer than the old, the chunk will be resized, invalidating all pointers to within the LMem heap.

Include: chunkarr.h

■ NameArrayCreate()

```
ChunkHandle NameArrayCreate(
          MemHandle          mh,           /* Global handle of LMem heap */
          word               dataSize,     /* Size of data section for
                                          * each element */
          word               headerSize);  /* Size of header; pass
                                          * zero for default header */
```

This routine creates a name array in the indicated LMem heap. It creates a **NameArrayHeader** structure at the head of a new chunk. If you want to leave extra space before the start of the array, you can pass a larger header size; if you want to use the standard header, pass a header size of zero.

You must specify the size of the data portion of each element when you create the array. The name portion will be variable sized.

Include: chunkarr.h

Tips and Tricks: If you want extra space after the **NameArrayHeader**, you may want to create your own header structure, the first element of which is a **NameArrayHeader**. You can pass the size of this header to **NameArrayCreate()** and access the data in your header via the structure fields.

Be Sure To: Lock the block on the global heap before calling this routine (unless it is fixed). If you pass a header size, make sure it is larger than **sizeof(NameArrayHeader)**.

Include: chunkarr.h

■ NameArrayCreateAt()

```
ChunkHandle NameArrayCreateAt(
          optr   array,       /* Turn this chunk into a name array */
          word   dataSize,    /* Size of data section of each element */
          word   headerSize); /* Size of header; pass zero for default header */
```

This routine is just like **NameArrayCreate()** above, except that the element array is created in a pre-existing chunk. The contents of that chunk will be destroyed.

Include: chunkarr.h

Warnings: If the chunk isn't large enough, it will be resized. This will invalidate all pointers to chunks in that block.

Include: chunkarr.h

■ NameArrayCreateAtHandles()

```
ChunkHandle NameArrayCreateAtHandles(
          MemHandle          mh,           /* Global handle of LMem heap */
          ChunkHandle        chunk,        /* the chunk for the array */
          word               dataSize,     /* Size of data for each element */
          word               headerSize);  /* Size of header; pass
                                          * zero for default header */
```

This routine is exactly like **NameArrayCreateAt()** above, except that the name array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

Warnings: If the chunk isn't large enough, it will be resized. This will invalidate all pointers to chunks in that block.

Include: chunkarr.h

■ NameArrayFind()

```
word      NameArrayFind(
          optr                array,        /* optr to name array */
          const char          * nameToFind, /* Find element with this name */
          word                nameLength,   /* Pass zero if name string is
                                          * null-terminated */
          void *              returnData);  /* Copy data section into this
                                          * buffer */
```

This routine locates the element with the specified name. It returns the token of the element and copies its data section into the passed buffer. If there is no element with the specified name, the routine will return CA_NULL_ELEMENT. The routine takes the following arguments:

array The optr of the name array.

nameToAdd
The name of the element to find. This string may contain nulls.

nameLength
The length of the name string, in bytes. If you pass zero, **NameArrayFind()** will assume the string is null-terminated.

returnData The data section of the element is written to this buffer.

Include: chunkarr.h

Warnings: You must make sure the *returnData* buffer is large enough to hold an element's data portion; otherwise, data after the buffer will be overwritten.

■ NameArrayFindHandles()

```
word      NameArrayFindHandles(
          MemHandle           mh,           /* Handle of LMem heap */
          ChunkHandle         array,        /* Handle of name array */
          const char *        nameToFind,   /* Find element with this name */
          word                nameLength,   /* Pass zero if name string is
                                          * null-terminated */
          void *              returnData);  /* Copy data section into this
                                          * buffer */
```

This routine is exactly like **NameArrayFind()** above, except that the name array is specified by its global and chunk handles (instead of with an optr).

Include: chunkarr.h

■ NEC()

```
NEC( line)
```

This macro defines a line of code that will only be compiled into the *non*-error-checking version of the geode. The *line* parameter of the macro is the actual line of code. When the non-EC version of the program is compiled, the line will be treated as a normal line of code; when the EC version is compiled, the line will be ignored.

Include: ec.h

■ ObjBlockGetOutput()

```
optr      ObjBlockGetOutput(
          MemHandle mh);         /* handle of the subject object block */
```

This routine returns the optr of the output destination object set for the specified object block. The output object is stored in the object block's header in the *OLMBH_output* field. If the block has no output set, NullOptr will be returned.

Include: object.h

See Also: ObjLMemBlockHeader

■ ObjBlockSetOutput()

```
void      ObjBlockSetOutput(
          MemHandle           mh,       /* handle of the subject object block */
          optr                o);       /* optr of the new output object */
```

This routine sets the *OLMBH_output* field in the specified object block's header. The optr passed in *o* will be set as the block's output.

Include: object.h

See Also: ObjLMemBlockHeader

■ ObjCompAddChild()

```
void      ObjCompAddChild(
          optr   obj,             /* optr of parent composite */
          optr   objToAdd,        /* optr of new child */
          word   flags,           /* CompChildFlags */
          word   masterOffset,    /* offset to master part */
          word   compOffset,      /* offset to comp field in master part */
          word   linkOffset);     /* offset to link field in master part */
```

This routine adds the given object to an object tree as the child of another specified object. It returns nothing. You will not likely want to use this routine; instead, you will probably use the messages listed below under "See Also." The parameters of this routine are

obj	The optr of the parent composite object. The parent must be a composite; if it is not, an error will result.
objToAdd	The optr of the child object. The child must have a link instance field (defined with **@link**).
flags	A record of **CompChildFlags**. These flags indicate whether the object should initially be marked dirty as well as where in the parent's child list the new child should be placed (first, second, last, etc.).
masterOffset	The offset within the parent's instance chunk to the master group's offset. (The value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure.)
compOffset	The offset within the parent's instance chunk to the composite field.
linkOffset	The offset within the parent's instance chunk to the link field.

Warnings: This routine may resize and move LMem and Object blocks on the heap, thereby invalidating all segment addresses and pointers.

Include: metaC.goh

See Also: MSG_VIS_ADD_CHILD, MSG_GEN_ADD_CHILD

■ ObjCompFindChildByNumber()

```
optr    ObjCompFindChildByNumber(
        optr   obj,              /* parent’s optr */
        word   childToFind,      /* zero-based child number */
        word   masterOffset,     /* offset to master part */
        word   compOffset,       /* offset to comp field in master part */
        word   linkOffset);      /* offset to link field in master part */
```

This routine returns the optr of the passed object’s child; the child is specified based on its position in the object’s child list. You will not often use this routine, but you will probably use the messages listed under “See Also” instead. The routine’s parameters are listed below:

obj The optr of the parent object.

childToFind
The zero-based number of the child to be found. For example, to return the first child’s optr, pass zero or CCO_FIRST; to return the last child’s optr, pass CCO_LAST.

masterOffset
The offset within the parent’s instance chunk to the master group’s offset. (The value that would appear in the parent class’ *Class_masterOffset* field in its **ClassStruct** structure.)

compOffset The offset within the parent’s instance chunk to the composite field.

linkOffset The offset within the parent’s instance chunk to the link field.

Include: metaC.goh

See Also: MSG_GEN_FIND_CHILD, MSG_VIS_FIND_CHILD

■ ObjCompFindChildByOptr()

```
word    ObjCompFindChildByOptr(
        optr   obj,              /* parent’s optr */
        optr   childToFind,      /* optr of child to find */
        word   masterOffset,     /* offset to master part */
        word   compOffset,       /* offset to comp field in master part */
        word   linkOffset);      /* offset to link field in master part */
```

This routine returns the zero-based child number of an object given its optr and its parent’s optr. The returned number represents the child’s position in its parent’s child list. For example, a return value of zero indicates the object is the parent’s first child. You will not likely use this routine; instead, you will probably use the messages shown below under “See Also.”

The parameters for this routine are listed below:

obj	The optr of the parent object.
childToFind	The optr of the child whose number is to be returned. If the child is not found, the routine will return -1.
masterOffset	The offset within the parent's instance chunk to the master group's offset. (The value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure.)
compOffset	The offset within the parent's instance chunk to the composite field.
linkOffset	The offset within the parent's instance chunk to the link field.

Include: metaC.goh

See Also: MSG_GEN_FIND_CHILD, MSG_VIS_FIND_CHILD

■ ObjCompMoveChild()

```
void    ObjCompMoveChild(
        optr   obj,             /* parent's optr */
        optr   objToMove,       /* optr of child to move */
        word   flags,           /* CompChildFlags */
        word   masterOffset,    /* offset to master part */
        word   compOffset,      /* offset to comp field in master part */
        word   linkOffset);     /* offset to link field in master part */
```

This routine moves the specified child within its parent's child list. This routine will not move a child from one parent to another, but it can reorganize a parent's children. You will not likely use this routine, but you may often use the messages listed under "See Also" below.

The parameters of this routine are shown below:

obj	The optr of the parent object.
objToMove	The optr of the child to be moved. If the optr does not point to a valid child, behavior is undefined and an error is likely.
flags	A record of **CompChildFlags** indicating the new position of the child and whether the link should be marked dirty.
masterOffset	The offset within the parent's instance chunk to the master group's offset. (The value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure.)

compOffset The offset within the parent's instance chunk to the composite field.

linkOffset The offset within the parent's instance chunk to the link field.

Warnings: This routine may cause LMem and/or Object Blocks to move or to shuffle their chunks, thereby invalidating any segment addresses or pointers.

Include: metaC.goh

See Also: MSG_GEN_MOVE_CHILD, MSG_VIS_MOVE_CHILD

■ ObjCompProcessChildren()

```
Boolean  ObjCompProcessChildren(
         optr                obj,          /* parent's optr */
         optr                firstChild,   /* optr of first child to process */
         ObjCompCallType     stdCallback,  /* standard callback type */
         void                * cbData,     /* data passed to callback */
         word                masterOffset, /* offset to master part */
         word                compOffset,   /* offset to comp field */
         word                linkOffset);  /* offset to link field */
         Boolean _pascal (*callback) (optr parent, optr child, void *cbData));
```

This routine performs a specific set of actions on all or some of an object's children. It is very rare that you will use this routine; typically, you should send a message to all of the parent's children. If, however, you use this routine, you must also write a callback routine that will be executed once for each affected child.

ObjCompProcessChildren() returns *true* (nonzero) only if it was stopped before all children had been processed. The only two ways this could be returned is if an error occurs or if your callback returns *true* when called.

The parameters for this routine are

obj The optr of the composite whose children are to be processed.

firstChild The optr of the first child to be processed. This routine will begin with the passed child and continue with all subsequent children. Pass the optr of the composite's first child—retrieved with the routine **ObjCompFindChildByNumber()**—to process all children.

stdCallback
A value of **ObjCompCallType** indicating how the data in the buffer pointed to by *cbData* will be passed to your callback routine. These values are detailed below.

cbData A pointer to a buffer in which data can be passed to your callback routine. This buffer can be altered by your callback.

masterOffset
The offset within the parent's instance chunk to the master group's offset. (The value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure.)

compOffset The offset within the parent's instance chunk to the composite field.

linkOffset The offset within the parent's instance chunk to the link field.

callback
A pointer to the actual callback routine that will be executed once for each child. The callback should be in your geode's fixed memory. The parameters and return values for the callback routine are given below.

The callback routine takes three parameters and returns a boolean value. It must be declared _pascal. The three parameters to the callback are listed below:

parent The optr of the parent composite.

child The optr of the current child being processed.

cbData A pointer to the buffer passed by the original caller of **ObjCompProcessChildren()**. What is actually in this buffer may depend on the value in the original *sdtCallback* parameter; if the buffer is not saved and restored by **ObjCompProcessChildren()** between children, each child may receive data altered by the previous child.

The callback routine can access and alter the buffer pointed to by *cbData*, or it can query the child or do anything else with the exception of destroying the child. It should return a Boolean value: *true* if **ObjCompProcessChildren()** should be aborted, *false* if it should not.

The values you can pass to **ObjCompProcessChildren()** in *stdCallback* are of type **ObjCompCallType**. You can use one of the following values to specify how the buffer in *cbData* will be passed on to the next child's callback routine:

OCCT_SAVE_PARAMS_TEST_ABORT
Save the buffer passed in *cbData* before calling each child; abort the routine if the callback returns *true*.

OCCT_SAVE_PARAMS_DONT_TEST_ABORT
: Save the buffer passed in *cbData* before calling each child; do not check the return value of the callback before proceeding to the next child.

OCCT_DONT_SAVE_PARAMS_TEST_ABORT
: Do not save the buffer in *cbData*, and abort if the callback routine returns *true*.

OCCT_DONT_SAVE_PARAMS_DONT_TEST_ABORT
: Do not save the buffer in *cbData*, and do not check the callback routine's return value.

OCCT_ABORT_AFTER_FIRST
: Abort the processing after only one child (typically used to call the *nth* child).

OCCT_COUNT_CHILDREN
: Counts the number of children rather than calling each.

Include: metaC.goh

See Also: @send, @call, MSG_META_SEND_CLASSED_EVENT

■ ObjCompRemoveChild()

```
void    ObjCompRemoveChild(
        optr   obj,             /* parent's optr */
        optr   objToRemove      /* optr of child to be removed */
        word   flags,           /* CompChildFlags */
        word   masterOffset,    /* offset to master part */
        word   compOffset,      /* offset to comp field in master part */
        word   linkOffset);     /* offset to link field in master part */
```

This routine removes the given child from the specified parent composite. The child will be removed entirely from the object tree, but it will not be detached or freed. The parameters of this routine are listed below:

obj — The optr of the parent composite.

objToRemove
: The optr of the child to be removed.

flags — A record of **CompChildFlags** indicating whether the parent and child should be marked dirty after the operation.

masterOffset
: The offset within the parent's instance chunk to the master group's offset. (The value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure.)

compOffset The offset within the parent's instance chunk to the composite field.

linkOffset The offset within the parent's instance chunk to the link field.

Include: metaC.goh

■ ObjDecInteractibleCount()

```
void      ObjDecInteractibleCount(
          MemHandle mh);         /* subject object block */
```

This routine decrements the given object block's interactable count. Do not decrement the interactable count without first incrementing it with **ObjIncInteractibleCount()**. Visible objects automatically decrement the interactable count in their MSG_VIS_CLOSE handlers.

Include: object.h

See Also: ObjIncInteractibleCount(), MSG_VIS_CLOSE, ObjLMemBlockHeader

■ ObjDecInUseCount()

```
void      ObjDecInUseCount(
          MemHandle mh);      /* subject object block */
```

This routine decrements the given object block's in-use count. When the in-use count reaches zero, the block may safely be freed. You should not decrement the in-use count of a block without first incrementing it at some point with **ObjIncInUseCount()**.

Warnings: Do not decrement the in-use count without incrementing it first. An error will result.

Include: object.h

See Also: ObjIncInUseCount(), ObjDecInteractibleCount(), ObjLMemBlockHeader

■ ObjDeref()

```
void      * ObjDeref(
          optr   obj                 /* optr to dereference */
          word   masterLevel);       /* specific master level to dereference */
```

This routine dereferences the given optr and master level to reset the message parameter *pself*. Because many routines and messages may cause the calling object's instance chunk to move, the *pself* parameter may become invalid. The two parameters to **ObjDeref()** are *obj* — The optr of the object to be dereferenced; nearly always you will want to pass **oself**.

masterLevel — The master level of the part to be dereferenced. This is the offset into the instance chunk where the offset to the master part is stored. Since *pself* points to the first byte of a master part, you must specify which master part you are dereferencing.

For example, a visible object dereferencing its **VisClass** instance data would call this routine as follows:

```
pself = ObjDeref(oself, 4);
```

Note, however, the **ObjDeref1()** and **ObjDerefVis()** exist to dereference the Vis master part, and **ObjDeref2()** and **ObjDerefGen()** exist to dereference the Gen master part.

Include: object.h

See Also: ObjDeref1(), ObjDeref2()

■ ObjDerefHandles()

```
void      * ObjDerefHandles(
          MemHandle           mh,               /* handle portion of optr */
          ChunkHandle         ch,               /* chunk portion of optr */
          word                masterLevel);     /* master level to dereference */
```

This routine is exactly the same as **ObjDeref()**, above, except that the optr is specified as its separate handles.

Include: object.h

■ ObjDeref1()

```
void      * ObjDeref1(
          optr obj);              /* optr of object to be dereferenced */
```

This routine is a special version of **ObjDeref()** which dereferences the first master part of an object. Visible objects should use this routine or **ObjDerefVis()** instead of **ObjDeref()**.

Include: object.h

See Also: ObjDeref(), ObjDeref2()

■ ObjDeref1Handles()

```
void        *ObjDeref1Handles(
            MemHandle           mh,       /* handle portion of optr */
            ChunkHandle         ch,);     /* chunk handle portion of optr */
```

This routine is exactly like **ObjDeref1()**, above, except that the optr is specified as its separate handles.

Include: object.h

■ ObjDeref2()

```
void        * ObjDeref2(
            optr   obj);           /* optr of object to be dereferenced */
```

This routine is a specialized version of **ObjDeref()** which dereferences the second master part of an object. Generic objects should use this routine or **ObjDerefGen()** instead of **ObjDeref()**.

Include: object.h

See Also: ObjDeref(), ObjDeref1()

■ ObjDeref2Handles()

```
void        * ObjDeref2Handles(
            MemHandle           mh,/      /* handle portion of optr */
            ChunkHandle         ch);      /* chunk portion of optr */
```

This routine is exactly like **ObjDeref2()**, above, except that the optr is specified as its separate handles.

Include: object.h

■ ObjDerefGen()

```
void        * ObjDerefGen(
            optr   obj);               /* generic object to be dereferenced */
```

This routine is exactly the same as **ObjDeref2()** and dereferences the Gen master part of a generic object.

Include: object.h

See Also: ObjDeref(), ObjDeref2()

■ ObjDerefVis()

```
void     * ObjDerefVis(
         optr   obj);             /* visible object to be dereferenced */
```

This routine is exactly the same as **ObjDeref1()** and dereferences the Vis master part of a visible object or a visibly-realized generic object.

Include: object.h

See Also: ObjDeref(), ObjDeref1()

■ ObjDoRelocation()

```
Boolean  ObjDoRelocation( /* returns true if error */
         ObjRelocationType   type,        /* type of relocation */
         MemHandle           block,       /* handle of info block */
         void                * sourceData,/* source of relocation */
         void                * destData); /* relocated value */
```

This routine relocates a given word or dword argument and is used for resolving handles and pointers to movable objects. Most often, relocation and unrelocation occur when resources are loaded, swapped, or saved, and this is in most cases taken care of by the kernel.

ObjDoRelocation() takes four parameters:

type	The type of relocation to be performed (**RelocationType**). This can be one of the three values shown below.
block	The handle of the block containing the relocation.
sourceData	A pointer to the source of the relocation; this pointer should be cast to the proper type (word or dword) when calling the routine.
destData	A pointer to the value to be returned; this pointer should be cast appropriately when the routine is called. The exact type of return value depends on *sourceData* and *type*, above.

The type of relocation to be done affects the type of data passed in *sourceData* and *destData*. The relocation type is passed in the type parameter and must be one of the following enumerations of **RelocationType**:

RELOC_RELOC_HANDLE
The relocation will be from a resource ID to a handle. The *sourceData* pointer should be cast to type word, and the *destData* pointer should be cast to type Handle.

RELOC_RELOC_SEGMENT
: The relocation will be from a resource ID to a segment address. The *sourceData* pointer should be cast to type word, and the *destData* pointer should be cast to type Segment.

RELOC_RELOC_ENTRY_POINT
: The relocation will be from either a resource ID or an entry number to an entry point. Both the *sourceData* pointer and the *destData* pointer should be cast to type dword.

ObjDoRelocation() returns an error flag that will be *true* if an error occurs, *false* otherwise.

The relocation done by this routine can be undone with **ObjDoUnRelocation()**.

Include: object.h

■ ObjDoUnRelocation()

```
Boolean  ObjDoUnRelocation( /* returns true if error */
         ObjRelocationType   type,        /* type of relocation */
         MemHandle           block,       /* handle of info block */
         void                * sourceData,/* source of relocation */
         void                * destData); /* relocated value */
```

This routine unrelocates a given word or dword. It translates a handle, a segment address, or an entry point back into a resource ID. The translation done is the exact reverse of that done by **ObjDoRelocation()**. See that routine (above) for more information.

ObjDoUnRelocation() returns an error flag that will be *true* if an error occurs and *false* if the unrelocation is successful. The unrelocated resource ID will be returned pointed to by the *destData* pointer.

Include: object.h

See Also: ObjDoRelocation()

■ ObjDuplicateMessage()

```
EventHandle ObjDuplicateMessage(
         EventHandle msg);          /* event to duplicate */
```

This routine duplicates a prerecorded event, returning the event handle of the new event. Pass the handle of the event to be duplicated. You can then change information about the event with **ObjSetMessageDestination()**.

Include: object.h

See Also: ObjSetEventInfo()

■ ObjDuplicateResource()

```
MemHandle ObjDuplicateResource(
          MemHandle           blockToDup,      /* handle of resource; must
                                                * not be loaded */
          GeodeHandle         owner,           /* owner of duplicate */
          ThreadHandle        burdenThread);   /* burden thread of duplicate */
```

This routine duplicates an entire object resource block. The new block will be put on the “saved blocks” list so it gets saved to the geode’s state file. Usually this is used by the UI to make editable copies of an application’s UI resources to ensure the proper state information gets saved. This routine takes three parameters:

blockToDup The handle of the block to be duplicated. The block must not be resident in memory when **ObjDuplicateResource()** is called. Also, it can only be a “template” resource—a resource that does not get used by the UI or the application directly, but only gets copied via this routine.

owner The owner geode of the new block. This should be the geode handle of the owning geode or zero to have the calling geode own it. If you pass an *owner* of -1, the new block will be owned by the same geode that owns the original.

burdenThread
The thread that will run the block and handle its messages. This should be a thread handle or zero to have the calling thread run the block. Passing a *burdenThread* of -1 makes the new resource have the same burden thread as the original.

ObjDuplicateResource() returns the handle of the newly created block, which will be unlocked and may or may not be resident in memory.

Include: object.h

See Also: ObjFreeDuplicate(), MSG_META_BLOCK_FREE, ObjLockObjBlock()

■ ObjEnableDetach()

```
void      ObjEnableDetach(
          optr   obj);            /* object calling this routine */
```

This routine acts as an object’s handler for MSG_META_ACK. This handler decrements the acknowledgment count (incremented with **ObjIncDetach()**) and, if the count is zero, enables the detach mechanism so the object can be fully detached. Because the detach mechanism is implemented in **MetaClass**, it is highly unlikely you will ever call this routine.

The lone parameter of this routine is the optr of the calling object (or, in the case of MSG_META_ACK, the object sending acknowledgment).

Warnings: This routine may resize and/or move chunks and object blocks, thereby invalidating all pointers and segment addresses.

Include: metaC.goh

See Also: MSG_META_DETACH, ObjInitDetach(), ObjIncDetach(), MSG_META_ACK

■ ObjFreeChunk()

```
void      ObjFreeChunk(
          optr   o);             /* optr of chunk to be freed */
```

This routine frees the passed object's instance chunk. If the object came from a loaded resource, however, the object is resized to zero and marked dirty rather than actually freed.

Warnings: The object must be fully detached, and its message queues must be empty before it can safely be freed. All this is handled by MSG_META_DETACH and MSG_META_OBJ_FREE.

Include: object.h

See Also: MSG_META_DETACH, MSG_META_OBJ_FREE, ObjInstantiate()

■ ObjFreeChunkHandles()

```
void      ObjFreeChunkHandles(
          MemHandle          mh,       /* handle portion of optr */
          ChunkHandle        ch);      /* chunk portion of optr */
```

This routine is the same as ObjFreeChunk(); the chunk is specified by its handles rather than by an optr.

Include: object.h

■ ObjFreeDuplicate()

```
void      ObjFreeDuplicate(
          MemHandle mh);         /* handle of duplicate block to be freed */
```

This routine frees a block that had been saved with **ObjSaveBlock()** or created with **ObjDuplicateResource()**. It must be passed the memory handle of the duplicated resource.

Warnings: All objects in the duplicated resource must be properly detached to ensure that nothing tries to send messages to the objects in the block. Additionally, the block's in-use count and interactable count should be zero.

Include: object.h

See Also: ObjDuplicateResource(), ObjSaveBlock(), ObjLMemBlockHeader

■ ObjFreeMessage()

```
void     ObjFreeMessage(
         EventHandle event);        /* event to be freed */
```

This routine frees an event handle and its associated event. This is rarely, if ever, used by anything other than the kernel. The kernel uses this routine to free events after they have been handled.

Include: object.h

■ ObjFreeObjBlock()

```
void     ObjFreeObjBlock(
         MemHandle block);      /* handle of the object block to be freed */
```

This routine frees the specified object block. It first checks the block's in-use count to see if any external references to the block are being kept. If the in-use count is nonzero, **ObjFreeObjBlock()** simply sets the block's auto-free bit and returns; the block will be freed the first time the in-use count reaches zero. If the in-use count is zero (no external references), the block will be freed immediately.

If the object block passed is not run by the calling thread, the operation will be handled by a remote call in the object block's thread.

Include: object.h

See Also: ObjFreeDuplicate(), MSG_META_BLOCK_FREE

■ ObjGetFlags()

```
ObjChunkFlags ObjGetFlags(
         optr   o);             /* optr of subject object */
```

This routine returns the object flags associated with a given object. The object is specified by the passed optr, and the flags are stored in the object's **ObjChunkFlags** record.

Include: object.h

See Also: ObjSetFlags(), ObjChunkFlags

■ ObjGetFlagsHandles()

```
ObjChunkFlags ObjGetFlagsHandles(
          Memhandle          mh,       /* handle portion of optr */
          ChunkHandle        ch);      /* chunk portion of optr */
```

This routine is the same as **ObjGetFlags()**, but the object is specified with its handles rather than with its optr.

Include: object.h

■ ObjGetMessageInfo()

```
Message   ObjGetMessageInfo(
          EventHandle        event,       /* event to be queried */
          optr               * dest);     /* buffer for destination optr */
```

This routine gets information about the specified *event*. The return value is the message number of the event. The *dest* parameter is a pointer to an optr. This routine will return with the optr represinting the event's destination object.

Include: object.h

■ ObjIncDetach()

```
void      ObjIncDetach(
          optr   obj);          /* optr of calling object */
```

This routine increments the number of detach acknowledgments an object must receive before it can safely be detached. Each time the detaching object sends notification of its detachment, it must call **ObjIncDetach()** to indicate that it must receive a corresponding detach acknowledgment (MSG_META_ACK).

The calling object must have previously called **ObjInitDetach()**. Since the detach mechanism is implemented in **MetaClass**, it is highly unlikely you will ever need to call this routine. **ObjIncDetach()** takes a single parameter: the optr of the calling object.

Include: metaC.goh

See Also: MSG_META_DETACH, ObjInitDetach(), ObjEnableDetach(), MSG_META_ACK

■ ObjIncInteractibleCount()

```
void      ObjIncInteractibleCount(
          MemHandle mh);         /* handle of object block */
```

This routine increments the interactable count of the given object block. The interactable count maintains the number of objects currently visible to the user or about to be acted on by the user (e.g. via keyboard accelerator). The interactable count is maintained by the UI; only in extremely special cases may you need to increment or decrement the count. To decrement the count, use **ObjDecInteractibleCount()**.

Visible objects increment the interactable count in their MSG_VIS_OPEN handlers and decrement it in their MSG_VIS_CLOSE handlers. This is built into **VisClass**.

Include: object.h

See Also: ObjDecInteractibleCount(), MSG_VIS_OPEN, MSG_VIS_CLOSE, ObjLMemBlockHeader

■ ObjIncInUseCount()

```
void      ObjIncInUseCount(
          MemHandle mh);         /* handle of object block */
```

This routine increments the given object block's in-use count. The in-use count maintains the number of outside references to this object block which are stored elsewhere and which need to be removed before the block can safely be freed. If you store an optr to an object block, you should increment the in-use count of the block.

When the reference to the block is removed, the in-use count should be decremented with **ObjDecInUseCount()**.

Include: object.h

See Also: ObjDecInUseCount(), ObjIncInteractibleCount(), ObjLMemBlockHeader

■ ObjInitDetach()

```
void      ObjInitDetach(
          MetaMessages        msg,
          optr                obj       /* object being detached */
          word                callerID, /* an identifier token for the caller */
          optr                ackOD);   /* object to which ack is sent */
```

Initialize the detach sequence for the specified object. The detach sequence severs all ties between the system and the object, allowing it to be destroyed without other objects or geodes trying to contact it. It is highly unlikely you will ever call this routine; typically, you will instead use MSG_META_DETACH or one of the generic or visible object messages, which will call this routine. The parameters for this routine are

msg	The detach message.
obj	The optr of the object to be detached.
callerID	The caller object's ID.
ackOD	The optr of the caller object or another object which is to receive acknowledgment notification of the detach.

Include: metaC.goh

See Also: MSG_META_DETACH, MSG_GEN_DESTROY, MSG_VIS_REMOVE, ObjIncDetach(), ObjEnableDetach(), MSG_META_ACK

■ ObjInitializeMaster()

```
void    ObjInitializeMaster(
        optr                obj,          /* object to be initialized */
        ClassStruct         * class);     /* class in master group */
```

This routine initializes the appropriate master part of the passed object, resizing the instance chunk if necessary. It takes two parameters:

obj	The optr of the object whose master part is to be initialized.
class	A pointer to the class definition of a class in the appropriate master group. This does not have to be the master class; it must only be a class in the master goup.

Warnings: This routine may resize and/or move chunks or object blocks, thereby invalidating pointers and segment addresses.

Include: object.h

See Also: ObjResizeMaster(), ObjInitializePart(), ClassStruct

■ ObjInitializeMasterHandles()

```
void    ObjInitializeMasterHandles(
        MemHandle           mh,           /* handle portion of optr */
        ChunkHandle         ch,           /* chunk portion of optr */
        ClassStruct         * class);     /* class in master group */
```

This routine is the same as **ObjInitializeMaster()** except it specifies the object via its handles rather than its optr.

Include: object.h

■ ObjInitializePart()

```
void      ObjInitializePart(
          optr   obj,                /* object to have a part initialized */
          word   masterOffset);      /* offset to master offset in chunk */
```

This routine initializes all master parts of the given object down to and including the master part specified in *masterOffset*. It will resize the chunk if necessary and even resolve variant classes above the master group specified, if necessary. This routine takes two parameters:

obj The optr of the object to be initialized.

masterOffset
The offset within the parent's instance chunk to the master group's offset (the value that would appear in the parent class' *Class_masterOffset* field in its **ClassStruct** structure).

Warnings: This routine may move and/or resize chunks or object blocks, thereby invalidating pointers and segment addresses.

Include: object.h

See Also: ObjResizeMaster(), ObjInitializeMaster(), MSG_META_RESOLVE_VARIANT_SUPERCLASS

■ ObjInitializePartHandles()

```
void      ObjInitializePartHandles(
          Memhandle          mh,              /* handle portion of optr */
          ChunkHandle        ch,              /* chunk portion of optr */
          word               masterOffset);   /* master group offset */
```

This routine is the same as **ObjInitializePart()** except that it specifies the object via its handles rather than an optr.

Include: object.h

■ ObjInstantiate()

```
optr      ObjInstantiate(
          MemHandle          block,       /* block in which new object
                                        * will be instantiated */
          ClassStruct        * class);    /* class of new object */
```

This routine instantiates a new object, allocating the proper size instance chunk. It returns the optr of the new object; this optr can then be used to send setup messages or other messages (such as adding the object to an object tree, setting it usable, etc.).

The new object's instance data will be initialized to all zeroes if it has no master parts (is a direct descendant of **MetaClass**). If it is a member of some master group, only enough space for the base structure (the master offsets and the class pointer) will be allocated. In either case, initialization of the instance data will occur at a later time.

ObjInstantiate() takes two parameters:

block The memory handle of an object block in which the object's instance chunk will be allocated. This block *must* be an object block, though it need not be run by the caller's thread. If the block is run by another thread, the routine will be executed as a remote call.

class A pointer to the **ClassStruct** structure of the class of the new object. This pointer will be set in the object's class pointer (the first four bytes of the instance chunk).

Warnings: This routine, because it allocates a new chunk, may cause LMem and Object blocks to move or resize, thereby invalidating any pointers and segment addresses. Be sure to dereference pointers after calls to this routine.

Include: metaC.goh

■ ObjIsClassADescendant()

```
Boolean  ObjIsClassADescendant(
         ClassStruct         * class1,    /* proposed ancestor */
         ClassStruct         * class2);   /* proposed descendant */
```

This routine checks if *class2* is a descendand of *class1* and returns *true* if it is.

Include: object.h

■ ObjIsObjectInClass()

```
Boolean  ObjIsObjectInClass(
         optr                obj,         /* object to check */
         ClassStruct         * class);    /* proposed class */
```

This routine checks to see if the passed object is a member of the specified class. It checks superclasses as well, but if an unresolved variant class is encountered, the variant will *not* be resolved. If you want to search past variant class links, you should sent MSG_META_DUMMY to the object first. The two parameters for this routine are *obj* The optr of the object to be checked.

class A pointer to the subject class' **ClassStruct** definition.

ObjIsObjectInClass() returns *true* if the object is in the class, *false* (zero) if it is not.

Include: object.h

■ ObjIsObjectInClassHandles()

```
Boolean   ObjIsObjectInClassHandles(
          MemHandle           mh,           /* handle portion of optr */
          ChunkHandle         ch,           /* chunk portion of optr */
          ClassStruct         * class);     /* proposed class */
```

This routine is just like **ObjIsObjectInClass()** except the object is specified via its handles rather than its optr.

Include: object.h

■ ObjLinkFindParent()

```
optr      ObjLinkFindParent(
          optr   obj,              /* child's optr */
          word   masterOffset,     /* offset to master part with link field */
          word   linkOffset);      /* offset in master part to link field */
```

This routine returns the optr of the specified object's parent. It must be passed the following:

obj The optr of the object whose parent is to be found.

masterOffset
The offset within the object's instance chunk to its master group's offset (the value that would appear in the *Class_masterOffset* field in its **ClassStruct** structure).

linkOffset The offset within the object's instance chunk to the link field.

Include: metaC.goh

See Also: MSG_VIS_FIND_PARENT, MSG_GEN_FIND_PARENT

■ ObjLockObjBlock()

```
void      * ObjLockObjBlock(
          MemHandle mh);         /* handle of object block */
```

This routine locks an object block, loading in the block if necessary. It must be passed the handle of the block, and it returns the locked block's segment address. When the caller is done using the block, it should unlock it with **MemUnlock()**.

Be Sure To: Always unlock the block when you are done with it, with **MemUnlock()**.

Include: object.h

See Also: MemLock(), MemUnlock()

■ ObjMapSavedToState()

```
VMBlockHandle ObjMapSavedToState(
          MemHandle mh);          /* handle of object block */
```

This routine returns the VM block handle of the state file block corresponding to the passed object block. If the specified object block has no state file equivalent, a null handle is returned.

Include: object.h

■ ObjMapStateToSaved()

```
MemHandle ObjMapStateToSaved(
          VMBlockHandle       vmbh,      /* VM block handle of state block */
          GeodeHandle         gh);       /* handle of geode owning block */
```

This routine takes a VM block handle and a geode handle and returns the memory block corresponding to the VM block, if any. The two parameters are

vmbh	The VM block handle of the VM block to be mapped.
gh	The geode handle of the owner of the block, or zero to use the calling geode's handle.

If the block is found, **ObjMapStateToSaved()** returns its handle. If the block is not found, it returns a null handle.

Include: object.h

■ ObjMarkDirty()

```
void      ObjMarkDirty(
          optr   o);          /* object to be marked dirty */
```

This routine marks an object dirty, indicating that changes to it should be saved when its object block is saved. If you want changes to objects saved, you should mark the object dirty.

Tips and Tricks: Often you do not need this routine because parameters or flags to other routines will set the object dirty automatically. If there is any doubt, however, you should use this routine.

Include: object.h

See Also: ObjChunkFlags, ObjSetFlags()

■ ObjMarkDirtyHandles()

```
void     ObjMarkDirtyHandles(
         MemHandle           mh,       /* handle portion of optr */
         ChunkHandle         ch);      /* chunk portion of optr */
```

This routine is the same as **ObjMarkDirty()** except that it specifies the object via handles rather than an optr.

■ ObjProcBroadcastMessage()

```
void     ObjProcBroadcastMessage(
         EventHandle event);    /* the event to be broadcast */
```

This routine broadcasts the given event to all threads which have message queues. It must be passed an encapsulated event (usually recorded with **@record**) and returns nothing. This is typically used for notification purposes.

Include: metaC.goh

■ ObjRelocateEntryPoint()

```
void *   ObjRelocateEntryPoint(
         EntryPointRelocation * relocData);
```

■ ObjRelocOrUnRelocSuper()

```
void     ObjRelocOrUnRelocSuper(
         optr                oself,
         ClassStruct         *class,
         word                frame);
```

Call this routine to relocate an object’s superclass.

Include: object.h

■ ObjResizeMaster()

```
void      ObjResizeMaster(
          optr   obj,            /* object to have its master part resized */
          word   masterOffset,   /* master offset of proper master part */
          word   newSize);       /* new size for the master part */
```

This routine resizes a master part of an object's instance chunk. It is typically used to allocate space for a master part or to resize the master part to zero (as when the Vis part of a visible object is removed in MSG_VIS_CLOSE). This routine must be passed the following three parameters:

obj	The optr of the object whose master part is to be resized.
masterOffset	The offset into the object's instance chunk where the offset to the master part is kept (this is the same offset held in the master class' *Class_masterOffset* field).
newSize	The new size of the master part. This can be found in the master class' *Class_instanceSize* field.

Warnings: This routine may resize and/or move chunks or object blocks, thereby invalidating stored segment addresses and pointers.

Include: object.h

See Also: ClassStruct, ObjInitializeMaster(), ObjInitializePart()

■ ObjResizeMasterHandles()

```
void      ObjResizeMasterHandles(
          MemHandle          mh,          /* handle portion of optr */
          ChunkHandle        ch,          /* chunk portion of optr */
          word               masterOffset,/* offset to master part */
          word               newSize);    /* new size of master part */
```

This routine is the same as **ObjResizeMaster()** except that the object is specified with its handles rather than its optr.

Include: object.h

■ ObjSaveBlock()

```
void      ObjSaveBlock(
          MemHandle mh);         /*handle of block to be marked for saving */
```

This routine sets up an object or LMem block to be saved to its owner's state file. The block's handle must be passed in *mh*. The block must be an object block.

Include: object.h

See Also: ObjMapSavedToState(), ObjMapStateToSaved()

■ ObjSetEventInfo()

```
void    ObjSetEventInfo(
        EventHandle         event,    /* handle of the event to be modified */
        Message             msg,      /* the new message for the event */
        optr                dest);    /* the new destination of the event */
```

This routine modifies an event, setting its information to the passed values. The three parameters are

event	The event handle of the event to be modified.
msg	The message to be sent in place of the current message. To use the same message, you must first retrieve it with **ObjGetMessageInfo()**.
dest	The optr of the new destination object for the event. To use the same destination object, you must first retrieve it with **ObjGetMessageInfo()**.

Include: object.h

See Also: ObjGetEventInfo()

■ ObjSetFlags()

```
void    ObjSetFlags(
        optr                o,          /* object whose flags will be set */
        ObjChunkFlags       bitsToSet,  /* flags to set */
        ObjChunkFlags       bitsToClear);/* flags to clear */
```

This routine sets the chunk flags for the specified object. Flags that should be set are passed in *bitsToSet*, and flags that should be cleared are passed in *bitsToClear*. Typically, applications will not use this routine but will rather let the kernel maintain the object’s flags.

Include: object.h

See Also: ObjGetFlags(), ObjChunkFlags

■ ObjSetFlagsHandles()

```
void      ObjSetFlagsHandles(
          MemHandle           mh,            /* handle portion of optr */
          ChunkHandle         ch,            /* chunk portion of optr */
          ObjChunkFlags       bitsToSet,     /* flags to set */
          ObjChunkFlags       bitsToClear);/* flags to clear */
```

This routine is the same as **ObjSetFlags()** except that the object is specified via its handles rather than its optr.

Include: object.h

■ ObjTestIfObjBlockRunByCurThread()

```
Boolean   ObjTestIfObjBlockRunByCurThread(
          MemHandle mh);         /* handle of object block */
```

This routine checks if the calling thread is running the specified object block. This routine can be used to determine if calls to objects in the block are across threads or internal to the calling thread. Pass this routine the handle of the object block to be checked—if the object block is a VM block, the thread for the VM file is checked rather than that for the block.

If the block is run by the calling thread, the return value is *true*. If a different thread runs the block, the return is *false* (zero).

Include: object.h

■ ObjUnrelocateEntryPoint()

```
void      ObjUnrelocateEntryPoint(
          EntryPointRelocation *  relocData,
          void *                  entryPoint);
```

■ ObjVarAddData()

```
void      * ObjVarAddData(
          optr                obj,           /* object to add vardata to */
          VardataKey          dataType,      /* vardata type */
          word                dataSize);     /* vardata data size, if any */
```

This routine adds or alters a variable data entry for the specified object. If the data type does not currently exist in the instance chunk, it will be allocated and added to the chunk. If it does exist, the extra data of the entry will be re-initialized to all zeroes.

This routine returns a pointer to the extra data of the new or modified entry; if the entry has no extra data, an opaque pointer to the entry is passed, and you can use this pointer with **ObjVarDeleteDataAt()**. In either case, the object will be marked dirty.

The parameters of this routine are

obj	The optr of the object affected. This should be the caller's optr.
dataType	The **VardataKey** word declaring the data type and its flags. The VDF_SAVE_TO_STATE flag must be properly set; the VDF_EXTRA_DATA flag is ignored, however, as the routine will set it properly.
dataSize	The size of the extra data for this type. If the type has no extra data, pass zero.

Include: object.h

Warnings: This routine should be called only by the object whose vardata is affected. To operate on other objects' vardata remotely, use messages provided by **MetaClass** (see below under "See Also").

See Also: MSG_META_ADD_VAR_DATA, ObjVarDeleteDataAt()

■ ObjVarAddDataHandles()

```
void      * ObjVarAddDataHandles(
          MemHandle          mh,           /* handle portion of optr */
          ChunkHandle        ch,           /* chunk portion of optr */
          VardataKey         dataType,     /* vardata type */
          word               dataSize);    /* vardata data size, if any */
```

This routine is the same as **ObjVarAddData()** except that the object is specified via its handles rather than its optr.

Include: object.h

■ ObjVarCopyDataRange()

```
void      ObjVarCopyDataRange(
          optr   source,      /* the optr of the source object */
          optr   dest,        /* the optr of the destination (calling) object */
          word   rangeStart,  /* the smallest data type value to be copied */
          word   rangeEnd);   /* the largest data type value to be copied */
```

This routine copies all the vardata entries within the specified range from the *source* object to the *dest* object. The range to be copied is specified by data types and is between *rangeStart* and *rangeEnd*, inclusive. If any data entries are copied, the destination object will be marked dirty.

Warnings: This routine should be called only by the destination object; it is against OOP doctrine for one object to alter another's instance data.

Include: object.h

■ ObjVarDeleteData()

```
Boolean  ObjVarDeleteData(
         optr                obj,          /* object to delete from */
         VardataKey          dataType);    /* data type to delete */
```

This routine deletes a vardata entry from the specified object's instance chunk, if the entry exists. The entry is specified by its data type; to delete an entry specified by a pointer to it, use **ObjVarDeleteDataAt()**, below. It returns an error flag: *true* if the entry was not found, *false* if the entry was successfully deleted. The object will also be marked dirty by the routine.

The parameters for this routine are

obj	The optr of the object affected. This should be the caller's optr.
dataType	The **VardataKey** word declaring the data type and its flags. Both the VDF_SAVE_TO_STATE flag and the VDF_EXTRA_DATA flag are ignored.

Warnings: This routine should be called only by the object whose vardata is affected. To operate on other objects' vardata remotely, use messages provided by **MetaClass** (see below under "See Also").

Include: object.h

See Also: MSG_META_DELETE_VAR_DATA, ObjVarDeleteDataAt()

■ ObjVarDeleteDataHandles()

```
Boolean  ObjVarDeleteDataHandles(
         MemHandle           mh,           /* handle portion of optr */
         ChunkHandle         ch,           /* chunk portion of optr */
         VardataKey          dataType);    /* data type to delete */
```

This routine is the same as **ObjVarDeleteData()** except that the object is specified via its handles rather than its optr.

Include: object.h

■ ObjVarDeleteDataAt()

```
void      ObjVarDeleteDataAt(
          optr   obj,               /* object to delete from */
          word   extraDataOffset);   /* offset to extra data to delete */
```

This routine deletes the specified vardata entry from the given object's instance chunk. The vardata entry is specified by its pointer as returned by **ObjVarAddData()**, **ObjVarFindData()**, and **ObjVarDerefData()**. To delete an entry specified by its data type, use **ObjVarDeleteData()**, above.

Warnings: This routine should be called only by the object whose vardata is affected. To operate on other objects' vardata remotely, use messages provided by **MetaClass** (see below under "See Also").

Include: object.h

See Also: MSG_META_DELETE_VAR_DATA, ObjVarDeleteData()

■ ObjVarDeleteDataAtHandles()

```
void      ObjVarDeleteDataAtHandles(
          MemHandle          mh,       /* handle portion of optr */
          ChunkHandle        ch,       /* chunk portion of optr */
          word   extraDataOffset);      /* offset to extra data to delete */
```

This routine is the same as **ObjVarDeleteDataAt()** except that the object is specified via its handles rather than its optr.

Include: object.h

■ ObjVarDeleteDataRange()

```
void      ObjVarDeleteDataRange(
          optr               obj,             /* object to delete from */
          word               rangeStart,      /* start of range */
          word               rangeEnd,        /* end of range */
          Boolean            useStateFlag);   /* save to state flag */
```

This routine deletes all data entries within a given range for the passed object. The range is specified by beginning and ending data types and is inclusive. The four parameters to this routine are

obj	The optr of the object whose data entries are to be deleted.
rangeStart	The lowest number data type to be deleted. Both the VDF_SAVE_TO_STATE flag and the VDF_EXTRA_DATA flag are ignored.

rangeEnd The highest number data type to be deleted. Both the VDF_SAVE_TO_STATE flag and the VDF_EXTRA_DATA flag are ignored.

useStateFlag
A flag indicating whether entries with their VDF_SAVE_TO_STATE flags should be deleted. Pass *true* (nonzero) to take the state flag into account; pass *false* (zero) to delete all entries in the range.

Warnings: This routine should be called only by the object whose vardata is affected. To operate on other objects' vardata remotely, use messages provided by **MetaClass** (see below under "See Also").

Include: object.h

See Also: MSG_META_DELETE_VAR_DATA

■ ObjVarDeleteDataRangeHandles()

```
void      ObjVarDeleteDataRangeHandles(
          MemHandle          mh,             /* handle portion of optr */
          ChunkHandle        ch,             /* chunk portion of optr */
          word               rangeStart,     /* start of range */
          word               rangeEnd,       /* end of range */
          Boolean            useStateFlag);  /* save to state flag */
```

This routine is the same as **ObjVarDeleteDataRange()** except that the object is specified via its handles rather than its optr.

Include: object.h

■ ObjVarDerefData()

```
void      * ObjVarDerefData(
          optr               obj,          /* object having data type */
          VardataKey         dataType);    /* data type to dereference */
```

This routine is exactly like **ObjVarFindData()**, below, except that it does not return a null pointer if the data type is not found. Do not use this routine unless you are absolutely sure the data type is in the object; otherwise, results are unpredictable.

Include: **object.h**

See Also: ObjVarFindData()

■ ObjVarDerefDataHandles()

```
void        * ObjVarDerefDataHandles(
            MemHandle           mh,           /* handle portion of optr */
            ChunkHandle         ch,           /* chunk portion of optr */
            VardataKey          dataType);    /* data type to dereference */
```

This routine is the same as **ObjVarDerefData()** except that the object is specified via its handles rather than its optr.

Include: **object.h**

■ ObjVarFindData()

```
void        * ObjVarFindData(
            optr                obj,          /* object to be checked */
            VardataKey          dataType);    /* data type to find */
```

This routine searches an object's variable data for a given data type. If the type is found, **ObjVarFindData()** returns a pointer to the entry's extra data; if the entry has no extra data, an opaque pointer is returned which may be used with **ObjVarDeleteDataAt()**. If the entry is not found, a null pointer is returned. The pointer returned by this routine must be used before any subsequent operations on the object's block; the pointer may be invalidated by other LMem or object operations.

The two parameters of this routine are

obj	The optr of the object affected. This should be the caller's optr.
dataType	The **VardataKey** word declaring the data type and its flags. Both the VDF_SAVE_TO_STATE flag and the VDF_EXTRA_DATA flag are ignored.

Warnings: This routine should be called only by the object whose vardata is affected. To operate on other objects' vardata remotely, use messages provided by **MetaClass** (see below under "See Also").

Include: **object.h**

See Also: MSG_META_FIND_VAR_DATA

■ ObjVarFindDataHandles()

```
void      * ObjVarFindDataHandles(
          MemHandle           mh,           /* handle portion of optr */
          ChunkHandle         ch,           /* chunk portion of optr */
          VardataKey          dataType);    /* data type to find */
```

This routine is the same as **ObjVarFindData()** except that the object is specified via its handles rather than its optr.

Include: **object.h**

■ ObjVarScanData()

```
void      ObjVarScanData(
          optr                obj,              /* object to be scanned */
          word                numHandlers,      /* number of handlers in table */
          VarDataCHandler     * handlerTable,   /* pointer to handler table */
          void                * handlerData);   /* pointer to handler data */
```

This routine scans an object's vardata and calls all the vardata handlers specified in the passed handler table. Pass it the following parameters:

obj The optr of the object whose variable data table is to be scanned.

numHandlers The number of handlers specified in the passed handler table.

handlerTable A pointer to a list of **VarDataCHandler** structures. Each of these structures contains a vardata data type and a pointer to the routine that is to handle it. All the handler routines must be in the same segment as the handler table.

handlerData A pointer to a buffer that is passed on to the handlers. This can contain any information of specific interest to the application or handlers.

Vardata Handler Format:
A vardata handler routine must have the following format:

```
void _pascal (MemHandle mh, ChunkHandle chnk,
      VarDataEntry *extraData, word dataType,
      void *handlerData)
```

The handler should not free the object chunk or destroy the object; it can do anything else it pleases. The handler returns nothing and takes the following parameters:

mh:chnk The memory handle and chunk handle of the object being referenced. Together, these comprise the optr of the object.

extraData A pointer to the data type's extra data, if it has any. This pointer may be parsed with the macros **VarDataTypePtr()**, **VarDataFlagsPtr()**, and **VarDataSizePtr()**.

dataType The data type of the data entry being handled. This is a record of type **VardataKey**.

handlerData
A pointer to a buffer passed through by **ObjVarScanData()**. This buffer may be used for passing additional data to the handlers.

Structures: The **VarDataCHandler** structure contains two elements:

```
typedef struct {
    word     VDCH_dataType;
    void_pascal (*VDCH_handler) (
                    MemHandle    mh,
                    ChunkHandle  chnk,
                    VarDataEntry * extraData,
                    word         dataType
                    void         * handlerData);
} VarDataCHandler;
```

The first element is the data type, a record containing the data type and the vardata flags. The second element is a far pointer to the handler routine for the type.

Include: **object.h**

■ ObjVarScanDataHandles()

```
void     ObjVarScanDataHandles(
         MemHandle          mh,          /* handle portion of optr */
         ChunkHandle        ch,          /* chunk portion of optr */
         word               numHandlers,     /* number of handlers in table */
         VarDataCHandler    * handlerTable,  /* pointer to handler table */
         void               * handlerData);  /* pointer to handler data */
```

This routine is the same as **ObjVarScanData()** except that the object is specified via its handles rather than its optr.

Include: **object.h**

■ offsetof()

```
word     offsetof(struc, field);
```

This macro returns the offset of the specified field within the specified structure.

■ OptrToChunk()

```
ChunkHandle OptrToChunk(op);
         optr   op;
```

This macro extracts the chunk handle portion of the given optr.

See Also: ConstructOptr(), OptrToHandle()

■ OptrToHandle()

```
MemHandle OptrToHandle(op);
         optr   op;
```

This macro extracts the MemHandle portion of the given optr.

See Also: ConstructOptr(), OptrToChunk()

■ ParallelClose()

```
StreamError ParallelClose(
         GeodeHandle         driver,
         ParallelUnit        unit,
         Boolean             linger);
```

Close the stream to a parallel port.

Include: **streamC.h**

■ ParallelOpen()

```
StreamError ParallelOpen(
         GeodeHandle         driver,
         ParallelUnit        unit,
         StreamOpenFlags     flags,
         word                outBuffSize,
         word                timeout);
```

This routine opens a stream to the specified parallel port. It is passed the following arguments:

driver	The **GeodeToken** of the parallel driver.
unit	The parallel port to open.

flags	This specifies whether the call should fail if the port is busy, or wait for a time to see if it will become free.
outBuffSize	The size of the stream buffer used for output to the parallel port.
timeout	The number of clock ticks to wait for the port to become free. (This argument is ignored if *flags* is not STREAM_OPEN_TIMEOUT.)

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

Include: **streamC.h**

■ ParallelWrite()

```
StreamError ParallelWrite(
          GeodeHandle         driver,
          ParallelUnit        unit,
          StreamBlocker       blocker,
          word                buffSize,
          const byte *        buffer,
          word *              numBytesWritten);
```

Write data to a parallel port.

Include: **streamC.h**

■ ParallelWriteByte()

```
StreamError ParallelWrite(
          GeodeHandle         driver,
          ParallelUnit        unit,
          StreamBlocker       blocker,
          word                buffSize,
          byte                dataByte);
```

Write one byte of data to a parallel port.

Include: **streamC.h**

■ PCB()

```
#define PCB(return_type, pointer_name, args) \
          return_type _pascal (*pointer_name) args
```

This macro is useful for declaring pointers to functions that use the pascal calling conventions. For example, to declare a pointer to a function which is passed two strings and returns an integer, one could write

```
PCB(int, func_ptr, (const char *, const char *));
```

which would be expanded to

```
int _pascal (*func_ptr) (const char *, const char *);
```

See Also: CCB()

■ PCCOMABORT()

```
void PCCOMABORT(void);
```

This routine aborts the current file transfer operation being carried out by the PCCom library. It is the third entry point in the PCCom library.

Include: **pccom.goh**

■ PCCOMEXIT()

```
PCComReturnType PCCOMEXIT();
```

This routine kills a pccom thread such as those started by PCCOMINIT(). It is the second entry point in the PCCom library.

Structures:

```
typedef ByteEnum PCComReturnType;
#define PCCRT_NO_ERROR 0
#define PCCRT_CANNOT_LOAD_SERIAL_DRIVER 1
#define PCCRT_CANNOT_CREATE_THREAD 2
#define PCCRT_CANNOT_ALLOC_STREAM 3
#define PCCRT_ALREADY_INITIALIZED 4
```

Include: **pccom.goh**

■ PCCOMINIT()

```
PCComReturnType PCCOMINIT(
          SerialPortNum       port,
          SerialBaud          baud,
          word                timeout,
          optr                callbackOptr,
          PCComInitFlags      flags);
```

This entry point of the PCCom library spawns a new thread which monitors a serial port and acts as a passive pccom terminal. This routine is the first entry point in the PCCom library.

This routine takes the following arguments:

port	A **SerialPortNum** value specifying which serial port to use for the pccom connection. Pass -1 for the system default value: **com1** for the Zoomer, **com2** for the desktop product.

baud	A **SerialBaud** value specifying what speed to use. Pass -1 for the system default value: 19200 baud for the Zoomer, 38400 baud for the desktop product.
timeout	Number of clock ticks (one tick is 1/60 second) to allow for connection.
callbackOptr	An object which will receive notification messages of certain events. A value of zero means no notification will be sent.
flags	If an object will be receiving notification messages, these flags determine what sort of notifications will be sent.

Structures:

```
typedef ByteEnum PCComReturnType;
#define PCCRT_NO_ERROR 0
#define PCCRT_CANNOT_LOAD_SERIAL_DRIVER 1
#define PCCRT_CANNOT_CREATE_THREAD 2
#define PCCRT_CANNOT_ALLOC_STREAM 3
#define PCCRT_ALREADY_INITIALIZED 4

typedef WordFlags PCComInitFlags;
      /* send notifications when text is available for display */
#define PCCIF_NOTIFY_OUTPUT 0x8000
      /* send notification when the remote machine shuts down the
       * serial line */
#define PCCIF_NOTIFY_EXIT 0x4000
```

Include: **pccom.goh**

■ ProcCallFixedOrMovable_cdecl()

```
dword   ProcCallFixedOrMovable_cdecl(
        void  (*routine),
        ...)
```

This routine calls the routine pointed to, passing the other arguments through to the called routine. The called routine must use C calling conventions.

Include: **resource.h**

■ ProcCallFixedOrMovable_pascal()

```
dword     ProcCallFixedOrMovable_pascal(
          …,
          void   (*routine))
```

This routine calls the routine pointed to, passing the other arguments through to the called routine. The called routine must use Pascal calling conventions.

Include: **resource.h**

■ ProcGetLibraryEntry()

```
void *    ProcGetLibraryEntry(
          GeodeHandle         library,
          word                entryNumber)
```

This routine returns the pointer to a library's entry-point.

Include: **resource.h**

■ ProcInfo()

```
ThreadHandle ProcInfo(
          GeodeHandle gh);    /* handle of geode to check */
```

This routine returns the first thread of the process geode specified. If the geode is not a process, the routine will return a null handle.

Include: **geode.h**

■ PtrToOffset()

```
word      PtrToOffset(ptr);
          dword  ptr;
```

This macro returns just the lower 16 bits of the given dword. It is most useful for extracting the offset portion of a far pointer.

■ PtrToSegment()

```
word      PtrToSegment(ptr);
          dword  ptr;
```

This macro returns just the upper 16 bits of the given dword. It is most useful for extracting the segment address of a far pointer.

■ qsort

```
extern void _pascal qsort(
         void *array,
         word count,
         word elementSize,
         PCB(int, compare, (const void *, const void *)));
```

This is a standard quicksort routine. The callback routine must be decared _pascal.

■ QueueGetInfo()

```
word     QueueGetInfo(
         QueueHandle qh);       /* queue to query */
```

This routine returns information about a specific event queue. Pass the handle of the queue; for information about the current process' queue, pass a null handle. This routine returns the number of events (or messages) currently in the queue.

Include: geode.h

■ QueueGetMessage()

```
EventHandle QueueGetMessage(
         QueueHandle qh);       /* queue to query */
```

This routine returns the next message on the given queue, blocking if the queue is empty. When a new message is added to the empty queue, this routine will unblock the thread and return the message. This routine is used almost exclusively by the kernel.

Include: geode.h

■ QueuePostMessage()

```
void     QueuePostMessage(
         QueueHandle        qh,       /* queue to add event to */
         EventHandle        event,    /* event to be added to queue */
         MessageFlags       flags);   /* MF_INSERT_AT_FRONT or zero */
```

This routine adds the specified *event* to the passed *queue*. The only valid flag for this routine is MF_INSERT_AT_FRONT, which will put the event in the first spot of the queue.

Include: geode.h

■ RangeEnum()

```
Boolean  RangeEnum(
         CellFunctionParameters  * cfp,       /* cell function parameters */
         RangeEnumParams         * params);   /* special other parameters */
```

This routine calls a callback routine for each cell in a specified range. This routine is passed pointers to two structures, both of which are shown below. It returns *false* if all the cells were processed, *true* if any of the cells caused the routine to abort before the end of the range was reached.

Callback Parameters:

The callback routine, which must be declared _pascal, receives a **RangeEnumCallbackParams** structure, which has the following definition:

```
typedef struct {
      RangeEnumParams     *RECP_params; /* see below */
   /* current row, column, and cell data of cell */
      word                RECP_row;
      word                RECP_column;
      word                RECP_cellData;
} RangeEnumCallbackParams;
```

The callback routine can do anything with the cell information. It should return *false* after successfully processing the cell; if an error occurs, or if it wants to abort the **RangeEnum()**, it should return *true*.

Structures: The **CellFunctionParameters** structure has the following definition:

```
typedef struct {
      CellFunctionParameterFlags CFP_flags;
            /* can have the following flags:
             * CFPF_DIRTY
             *     set parameter block dirty
             * CFPF_NO_FREE_COUNT
             *     counts the number of calls to
             *     a non-special RangeEnum() */
      VMFileHandle        CFP_file;
            /* VM file handle of cell file */
      VMBlockHandle       CFP_rowBlocks[N_ROW_BLOCKS];
            /* array of handles to grouped row blocks */
} CellFunctionParameters;
```

Include: cell.h

■ RangeExists()

```
Boolean   RangeExists( /* returns non-zero if there are cells in range */
          CellFunctionParameters  * cfp,        /* see RangeEnum() */
          word                      firstRow,   /* range delimiters */
          byte                      firstColumn,
          word                      lastRow,
          byte                      lastColumn);
```

This routine returns *true* if there are any cells in the specified range. It is passed a pointer to the **CellFunctionParameters** structure for the cell file, as well as the indices of the first and last row, and the first and last column, of the range to check.

Include: cell.h

■ RangeInsert()

```
void      RangeInsert(
          CellFunctionParameters  * cfp,        /* see RangeEnum() */
          RangeInsertParams       * rep);       /* parameters structure */
```

This routine shifts existing cells to make room for new ones. (It does not actually create new cells.) Which cells are shifted, and in what direction, is specified by the **RangeInsertParams()** structure. This structure has three fields:

RIP_bounds	A **Rectangle** structure which specifies which cells should be shifted. The cells currently in this range will be shifted across or down, depending on the value of *RIP_delta*; the shifted cells displace more cells, and so on, to the edge of the visible portion of the cell file. To insert an entire row (which is much faster than inserting a partial row), set *RIP_bounds.R_left* = 0 and *RIP_bounds.R_right* = LARGEST_COLUMN.
RIP_delta	A **Point** structure which specifies how far the cells should be shifted and in which direction. If the range of cells is to be shifted horizontally, *RIP_delta.P_x* should specify how far the cells should be shifted over, and *RIP_delta.P_y* should be zero. If the cells are to be shifted vertically, *RIP_delta.P_y* should specify how far the cells should be shifted over, and *RIP_delta.P_x* should be zero.
RIP_cfp	This is the address of the **CellFunctionParameters** structure. You don't have to initialize this; the routine will do so automatically.

Include: cell.h

Warnings: If cells are shifted off the “visible” portion of the cell file, you will be unable to access them by row or column numbers; but they will not be deleted. For this reason, you should free all such cells *before* calling **RangeInsert()**. (You can find out if there are any cells at the edges by calling **RangeExists()**.) For an explanation of the “visible” and “scratch-pad” portions of a cell file, see Section 19.4.1 of the Concepts book.

■ realloc()

```
void *   realloc(
         void *              blockPtr,     /* address of memory to resize */
         size_t              newSize);     /* New size of memory in bytes */
```

The **malloc()** family of routines is provided for Standard C compatibility. If a geode needs a small amount of fixed memory, it can call one of the routines. The kernel will allocate a fixed block to satisfy the geode’s **malloc()** requests; it will allocate memory from this block. When the block is filled, it will allocate another fixed malloc-block. When all the memory in the block is freed, the memory manager will automatically free the block.

If a geode needs to change the size of a section of memory assigned to it by the **malloc()** family of routines, it should use **realloc()**. **realloc()** resizes the piece of memory specified and returns the memory’s new base address.

If the new size is smaller then the previous size, bytes will be cut off from the end. The request is guaranteed to succeed. Furthermore, the memory will not be moved; the address returned will be the same as the address passed.

If the new size is larger than the previous size, **realloc()** may move the data to accommodate the request. If so, it will return the new address. The new memory added will *not* be zero-initialized. If **realloc()** cannot fulfill the request, it will return a null pointer, and the memory will not be altered.

Resizing a stretch of memory down to zero bytes is exactly the same as freeing it with **free()**. If you pass a null address to **realloc()**, it will allocate the memory the same way **malloc()** does.

The memory must be in a malloc-block assigned to the geode calling **realloc()**. If you want to resize memory in another geode’s malloc-block, call **GeoReAlloc()**.

Warnings: Pass exactly the same address as the one returned to you when you allocated the memory. If you pass a different address, the results are undefined.

See Also: calloc(), free(), malloc(), GeoReAlloc()

■ SerialClose()

```
StreamError SerialClose(
          GeodeHandle         driver,
          SerialUnit          unit,
          Boolean             linger);
```

Close the stream to a serial port.

■ SerialCloseWithoutReset()

```
StreamError SerialClose(
          GeodeHandle         driver,
          SerialUnit          unit,
          Boolean             linger);
```

Close the stream to a serial port, without actually resetting the port.

■ SerialFlush()

```
StreamError SerialFlush(
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamRoles         roles);
```

Flush all data pending in a serial port's input or output buffer (depending on the value of *roles*).

■ SerialGetFormat()

```
StreamError SerialGetFormat(
          GeodeHandle         driver,
          SerialUnit          unit,
          SerialFormat *      format,
          SerialMode *        mode,
          SerialBaud *        baud);
```

Get the format of a stream to a specified serial port.

■ SerialGetModem()

```
StreamError SerialGetModem(
          GeodeHandle         driver,
          SerialUnit          unit,
          SerialModem *       modem);
```

Read a modem's hardware flow control bits.

■ SerialOpen()

```
StreamError SerialOpen(
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamOpenFlags     flags,
          word                inBuffSize,
          word                outBuffSize,
          word                timeout);
```

This routine opens a stream to the specified serial port. It is passed the following arguments:

driver	The **GeodeToken** of the serial driver.
unit	The serial port to open.
flags	This specifies whether the call should fail if the port is busy, or wait for a time to see if it will become free.
inBuffSize	The size of the stream buffer used for input from the serial port.
outBuffSize	The size of the stream buffer used for output to the serial port.
timeout	The number of clock ticks to wait for the port to become free. (This argument is ignored if *flags* is not STREAM_OPEN_TIMEOUT.)

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ SerialQuery()

```
StreamError SerialQuery(
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamRoles role,
          word *              bytesAvailable);
```

Find out how much space is available in a serial buffer, or how much data is waiting to be read.

■ SerialRead()

```
StreamError SerialRead (
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamBlocker       blocker,
          word                buffSize,
          byte *              buffer,
          word *              numBytesRead);
```

Read data from a serial port and write it to a passed buffer.

■ SerialReadByte()

```
StreamError SerialReadByte (
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamBlocker       blocker,
          word                buffSize,
          byte *              dataByte);
```

Read a byte of data from a serial port and write it to a passed variable.

■ SerialSetFormat()

```
StreamError SerialSetFormat(
          GeodeHandle         driver,
          SerialUnit          unit,
          SerialFormat        format,
          SerialMode          mode,
          SerialBaud          baud);
```

Set the format for a stream to a specified serial port.

■ SerialSetModem()

```
StreamError SerialSetModem(
          GeodeHandle         driver,
          SerialUnit          unit,
          SerialModem         modem);
```

Set a modem's hardware flow control bits.

■ SerialWrite()

```
StreamError SerialWrite(
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamBlocker       blocker,
          word                buffSize,
          const byte *        buffer,
          word *              numBytesWritten);
```

Write data to a serial port.

■ SerialWriteByte()

```
StreamError SerialWrite(
          GeodeHandle         driver,
          SerialUnit          unit,
          StreamBlocker       blocker,
          word                buffSize,
          byte                dataByte);
```

Write one byte of data to a serial port.

■ SGC_MACHINE

```
byte      SGC_MACHINE(val);
          dword  val;
```

This macro is used to extract the machine type from a **SysGetConfig()** return value.

Include: system.goh

■ SGC_PROCESSOR

```
byte      SGC_PROCESSOR(val);
          dword  val;
```

This macro is used to extract the processor type from a **SysGetConfig()** return value.

Include: system.goh

■ SoundAllocMusic()

```
MemHandle SoundAllocMusic(
          const word          *song,
          word                voices );
```

This routine takes a pointer to a fixed buffer of music and returns a **MemHandle** which may then be passed to **SoundPlayMusic()** to play the music. If the music buffer is in a movable resource, you must initialize it using **SoundInitMusic()** instead of **SoundAllocMusic()**. To find out how to set up one of these buffers of music, see "Sound Library," Chapter 13 of the Concepts book. The *voices* argument is the number of voices in the buffer.

■ SoundAllocMusicNote()

```
MemHandle SoundAllocMusicNote(
          word _far          instrument,
          word               frequency,
          word               volume,
          word               DeltaType,
          word               duration);
```

This routine allocates a **MemHandle** which may be passed to **SoundPlayMusicNote()**. You must provide all information about the note: its frequency, volume, and duration. You may specify an *instrument*, passing a value corresponding to a standard instrument (such as IP_PIANO). Specify the frequency in Hertz or use one of the constants such as MIDDLE_C_b to specify a standard note frequency. Volume ranges from zero to 0xffff—you may wish to use a constant value such as DYNAMIC_FFF if you want help trying to choose a loudness. The note's duration is determined by its delta type, one of SSDTT_MSEC, SSDTT_TICKS, and SSDTT_TEMPO. If you pass SSDTT_MSEC or SSDTT_TICKS, the duration is measured in milliseconds or ticks (each tick is one sixtieth of a second). If you pass SSDTT_TEMPO, you may set the size of your time unit when you call **SoundPlayMusicNote()**. The *duration* determines how many time units the note should play. If the delta type is SSDTT_TICKS and *duration* is 30, then the note will sound for half a second.

■ SoundAllocMusicStream()

```
MemHandle SoundAllocMusicStream(
          word               streamType,
          word               priority,
          word               voices,
          word               tempo);
```

This routine returns a token suitable for passing to **SoundPlayMusicToStream()**. It is passed several arguments. The **SoundStreamType** determines how much space to allocate for the stream and will determine how much data can be written to the stream at one time. If you pass SST_ONE_SHOT, it indicates that the stream will not be explicitly destroyed, and that your stream should destroy the stream when the song is done. You must specify how many voices there are in the music buffer. You must also pass a starting *tempo* for the music stream.

■ SoundAllocSampleStream()

```
MemHandle SoundAllocSampleStream(void);
```

This routine allocates a sample stream handle. If the returned handle is *null*, the library was unavailable (i.e. some other thread has grabbed exclusive access).

■ SoundDisableSampleStream()

```
void      SoundDisableSampleStream(
          MemHandle           mh);
```

This routine disassociates the DAC player from the passed sample handle. Before you play more sounds using the handle, you will have to call **SoundEnableSampleStream()** again.

■ SoundEnableSampleStream()

```
Boolean   SoundEnableSampleStream(
          MemHandle           mh,
          word                priority,
          word                rate,
          word                manufacturerID,
          word                format);
```

This routine associates a DAC player with the allocated sample handle. You must pass the sound handle, as returned by **SoundAllocSampleStream()**. You must also pass certain pieces of information about the sound you will be playing on the DAC device: the *priority* with which to grab the DAC player (e.g. SP_STANDARD), the sampling rate, and the *format* of the sample (as identified by a *manufacturerID* and a **DACSampleFormat** value).

■ SoundFreeMusic()

```
void      SoundFreeMusic(
          MemHandle           mh);
```

This routine frees up a music handle. The music must not be playing; call **SoundStopMusic()** if you are not sure. You may not use the music handle after calling this routine on it.

■ SoundFreeMusicNote()

```
void SoundFreeMusicNote(
          MemHandle           mh);
```

This routine frees up the passed note handle. The note must not be playing when you call this routine; call **SoundStopMusicNote()** if you are not sure. You should not try to use the note's handle after freeing it.

■ SoundFreeMusicStream()

```
void SoundFreeMusicStream(
          MemHandle         mh);
```

This routine frees up the music stream's token. No music must be playing via the stream; call **SoundDisableMusicStream()** if you are not sure. Do not try to use the stream after calling this routine on it.

■ SoundFreeSampleStream()

```
void      SoundFreeSampleStream(
          MemHandle         mh);
```

This routine frees the passed sampled sound handle. You must not try to use this handle after calling this routine on it.

■ SoundGetExclusive()

```
void      SoundGetExclusive(void);
```

This routine grabs the exclusive semaphore for the sound library; if another thread has already grabbed the exclusive, this routine will wait until the exclusive is released. Sounds which are playing now will be permitted to finish, but from now on, only the thread calling this routine will be allowed to play new sounds. When done with the sound library exclusive, call **SoundReleaseExclusive()**.

■ SoundGetExclusiveNB()

```
Boolean   SoundGetExclusiveNB(void);
```

This routine grabs the exclusive semaphore for the sound library, doing so even if some other thread has already grabbed the exclusive. Sounds which are playing now will be permitted to finish, but from now on, only the thread calling this routine will be allowed to play new sounds. This routine will return *true* if another thread already has exclusive access.

When done with the sound library exclusive, call **SoundReleaseExclusive()**.

■ SoundInitMusic()

```
void SoundInitMusic(
          MemHandle         mh,
          byte              voices);
```

This routine initializes a pre-defined simple music buffer structure. If the music buffer is stored in a fixed block, you can call **SoundAllocMusic()**

instead. This allows a music buffer stored in a block referenced by a pointer to be playable using **SoundPlayMusic()**.

■ SoundPlayMusic()

```
Boolean SoundPlayMusic(
        MemHandle           mh,
        word                priority,
        word                tempo,
        char                flags);
```

This routine plays a buffer of music previously initialized by **SoundInitMusic()** or allocated by **SoundAllocMusic()**. The priority value will determine whether your sound will play if other sounds are already occupying the voices—pass a value such as SP_STANDARD. The *tempo* value will be used to determine the length of a 1/128th note. If your music buffer contained any notes whose lengths were measured by SSDTT_TEMPO delta type, then you should set this value accordingly. The *flags* argument determines whether the music's handle should be automatically freed when the sound is done playing. You may pass either or both of the flags UNLOCK_ON_EOS or DESTROY_ON_EOS.

Remember that you must have called **SoundInitMusic()** on the music handle before you may use it to play music.

Include: sound.h

■ SoundPlayMusicNote()

```
Boolean SoundPlayMusicNote(
        MemHandle           mh,          /* handle of note */
        word                priority,
        word                tempo,
        word                flags);
```

This routine plays a buffer of music previously allocated by **SoundAllocMusicNote()**—the return value of that function is passed as *mh*. The priority value will determine whether your sound will play if other sounds are already occupying the voices—pass a value such as SP_STANDARD. The *tempo* value will be used to determine the length of a 1/128th note. If your note's delta type is SSDTT_TEMPO, then you should set this value accordingly. The *flags* argument determines whether the notes's handle should be automatically freed when the note is done playing. You may pass either or both of the flags UNLOCK_ON_EOS or DESTROY_ON_EOS.

This routine returns *true* if the library was unavailable (i.e. if some other thread had grabbed the sound exclusive).

Include: sound.h

■ SoundPlayToMusicStream()

```
Boolean  SoundPlayToMusicStream(
         MemHandle           mh,
         const word          * sample,
         word                size,
         SampleFormatDescription *format);
```

This routine plays a music buffer to a stream. Specify which stream to play to by means of the token returned by **SoundAllocMusicStream()**. To play music to the buffer, pass the size of the buffer you are playing and a pointer to the start of the piece. This piece of buffer must be made up of whole events—it should not start or end in the middle of an event (e.g. you can't specify that you want to play a note but not give its frequency, even if you plan to play another buffer to the stream that might begin with a frequency).

If you don't know the size of the buffer, it may be all right—any data in the buffer after the GE_END_OF_SONG will be ignored.

■ SoundPlayToSampleStream()

```
Boolean SoundPlayToSampleStream(
         MemHandle           mh,
         word _far           * sample,
         word                size,
         SampleFormatDescription * format);
```

This routine passes sampled sound data to a DAC player. You must pass a sample sound handle to this routine—to acquire such a handle, call **SoundAllocSampleStream()**. The sample sound handle must be associated with a DAC player—to so associate the handle, call **SoundEnableSampleStream()**. You must pass a pointer to the *sample* data, along with the *size* of the sample as measured in bytes. You may change the *format* information which will determine how the DAC player handles the data.

■ SoundReallocMusic()

```
Boolean  SoundReallocMusic(
         MemHandle           mh,
         word _far           * song);
```

This routine allows you to associate a new music buffer with an existing music handle. The new music buffer must not have more voices than was originally requested with **SoundAllocMusic()**. Do not call this routine with the handle of a sound that may be playing; call **SoundStopMusic()** on the handle if you are not sure. See "Sound Library," Chapter 13 of the Concepts book to find out how to set up the buffer of music.

■ SoundReallocMusicNote()

```
Boolean   SoundReallocMusicNote(
          MemHandle           mh,
          word                freq,
          word                vol,
          word                timer,
          word                durat,
          word _far           * instrum);
```

This routine allows you to associate new note values with an existing note handle. Do not call this routine with the handle of a note that may be playing; call **SoundStopMusicNote()** on the handle if you are not sure.

■ SoundReleaseExclusive()

```
void      SoundReleaseExclusive(void);
```

This routine releases the sound library exclusive semaphore. You will not need to call this routine unless your code calls **SoundGrabExclusive()** or **SoundGrabExclusiveNB()**. This routine allows other threads to play sounds. If another thread called **SoundGrabExclusive()** while your thread had the exclusive, it will now grab the exclusive.

■ SoundStopMusic()

```
Boolean   SoundStopMusic(
          MemHandle           mh);   /* Handle of music buffer */
```

This routine stops the playing of a simple music buffer. It returns true if the library was unavailable (i.e. some other thread has grabbed the exclusive).

■ SoundStopMusicNote()

```
Boolean   SoundStopMusicNote(
          MemHandle           mh);
```

This routine stops a note that is playing. Pass the handle of the note, as was returned by **SoundAllocMusicNote()**. This routine returns true if the sound library was unavailable (i.e. some other thread has grabbed the exclusive).

■ SoundStopMusicStream()

```
Boolean   SoundStopMusicStream(
          MemHandle          mh);
```

This routine stops any music being played to the stream. All sounds are flushed from the stream. It takes one argument, the token of the sound stream, as returned by **SoundAllocMusicStream()**.

■ SoundStopSampleStream()

```
void      SoundStopSampleStream(
          MemHandle          mh);
```

This routine stops a sound playing through a previously allocated sampled sound stream.

■ SpoolConvertPaperSize()

```
word      SpoolConvertPaperSize(
          int                width,    /* width of paper */
          int                height,   /* height of paper */
          PageType           pt);      /* type of page */
```

This routine converts a width and height into a page size number.

Include: spool.goh

■ SpoolCreatePaperSize()

```
Boolean   SpoolCreatePaperSize( /* Returns true if failed */
          word   * retValue,      /* returns paper size value */
          char   * name,          /* name of paper size */
          int    width,           /* width of paper */
          int    length,          /* length of paper */
          PageLayout laytout);    /* default page layout */
```

This routine defines and stores a new paper size for later use by the user.

Include: spool.goh

■ SpoolCreatePrinter()

```
Boolean  SpoolCreatePrinter( /* Returns true if error
                             (printer already exists) */
         char                 *name,       /* name of printer */
         PrinterDriverType    type,        /* driver type */
         int                  * retVal);   /* Will hold printer number */
```

Adds the printer to the list of currently installed printers and returns the new printer number. This routine is normally called from within the Preferences manager. Returns *true* if the printer already exists.

Include: spool.goh

■ SpoolDeletePaperSize()

```
Boolean  SpoolDeletePaperSize(
         word   size);           /* size number to delete */
```

This routine deletes a user-defined paper size.

Include: spool.goh

■ SpoolDeletePrinter()

```
void     SpoolDeletePrinter(
         int    prtrNum);        /* printer number to delete */
```

Deletes the requested printer from the system.

Include: spool.goh

■ SpoolGetDefaultPrinter()

```
int      SpoolGetDefaultPrinter(); /* Returns printer number */
```

Returns the system-default printer, which is used (for example) by the **PrintControlClass** as the default printer to print to.

Include: spool.goh

■ SpoolGetNumPaperSizes()

```
int      SpoolGetNumPaperSizes(
         PageType type);         /* type of page */
```

Use this routine to find the number of paper sizes, both pre-defined and user-defined, that should appear in a paper size list.

Include: spool.goh

■ SpoolGetNumPrinters()

```
int       SpoolGetNumPrinters(
          PrinterDriverType   type);    /* driver type */
```

This routine returns the number of installed printers with the given type.

Structures:

```
          typedef ByteEnum PrinterDriverType;
          /* The driver type may be one of the following:
                 PDT_PRINTER,
                 PDT_PLOTTER,
                 PDT_FACSIMILE,
                 PDT_CAMERA,
                 PDT_OTHER,
                 PDT_ALL*/
```

Include: spool.goh

■ SpoolGetPaperSize()

```
XYSizeAsDWord SpoolGetPaperSize(
          int                 size,     /* This must be between 0 and the return
                                         * value of SpoolGetNumPaperSizes() */
          PageType            pt,       /* type of page */
          PageLayout          *layout); /* Will hold returned page layout */
```

Use this routine to determine the dimensions of a paper size.

Include: spool.goh

■ SpoolGetPaperSizeOrder()

```
dword     SpoolGetPaperSizeOrder( /* High byte is number of unused sizes;
                                   * Low byte is # of ordered sizes */
          PageType          type,
          byte              *order,      /* buffer of size MAX_PAPER_SIZES */
                                      /* On return, this buffer will be
                                       * filled with the page size numbers
                                       * arranged in the order
                                       * corresponding to their display */
          byte              *userSizes); /* buffer of size MAX_PAPER_SIZES */
                                         /* On return, will hold ordered
                                          * array of user paper sizes. */
```

■ SpoolGetPaperSizeString()

```
Boolean   SpoolGetPaperSizeString( /* true if error*/
          char              * retValue,  /* buffer for returned value */
          int               size,  /* Must be between 0 and the return
                                    * value of SpoolGetNumPaperSizes() */
          PageType          pt);   /* type of page */
```

Use this routine to determine the string to be displayed for a specific paper size. Upon return, *retValue* will point to a character string and the Boolean return value will be *false* if successful. If any error occurs, or if the page type couldn't be found, the returned value will be *true*.

Include: spool.goh

■ SpoolGetPrinterString()

```
Boolean   SpoolGetPrinterString( /* Returns true if error */
          int    *retValue,    /* On return, will point to length of string */
          char   *string,      /* returned name string */
          int    prtrNum);     /* printer number */
```

This routine fills a buffer with the requested null-terminated printer name string. If the printer could not be found, the return value will be *true* (set for error).

Include: spool.goh

■ SpoolSetDefaultPrinter()

```
void      SpoolSetDefaultPrinter(
          int prtrNum);        /* printer number */
```

Sets the system-default printer, used (for example) by **PrintControlClass** as the default printer. This routine is normally called from within the Preferences manager.

Include: spool.goh

■ SpoolSetDocSize()

```
void      SpoolSetDocSize(
          Boolean             open;     /* false if document is closed */
          PageSizeInfo        * psr);   /* NULL if document is closed */
```

This routine tells the application's PageSizeControl object the document's size.

Include: spool.goh

■ SpoolSetPaperSizeOrder()

```
void      SpoolSetPaperSizeOrder(
          void   * ptr,          /* Array of PageSizeOrder entries */
          word   number);        /* number of entries in array */
```

This routine resets the order in which paper sizes are displayed to the user.

Include: spool.goh

■ SpreadsheetInitFile()

```
VMBlockHandle SpreadsheetInitFile(
          const SpreadsheetInitFileData * ifd);
```

This routine initializes a VM file for use by the spreadsheet object. It allocates a spreadsheet map block in the file and initializes this block. The routine returns the map block's handle; applications will need to remember this handle. It does not change any existing blocks in the VM file.

The *ifd* parameter is pointer to a **SpreadsheetInitFileData** structure containing the file handle and the number of rows and columns to allocate.

Structures: The **SpreadsheetInitFileData** structure is defined as follows:

```
typedef struct {
      word                        SIFD_file;
      word                        SIFD_numRows;
      word                        SIFD_numCols;
      SpreadsheetDrawFlags        SIFD_drawFlags;
} SpreadsheetInitFileData;

/* SpreadsheetDrawFlags:
 * SDF_DRAW_GRAPHICS
 * SDF_DRAW_NOTE_BUTTON
 * SDF_DRAW_HEADER_FOOTER_BUTTON
 * SDF_DRAW_GRID           */
```

Include: ssheet.goh

■ StreamClose()

```
StreamError StreamClose (
          GeodeHandle         driver,
          StreamToken         stream,
          Boolean             linger);
```

This routine shuts down a stream. It is passed the following arguments:

driver	The **GeodeToken** of the stream driver.
stream	The **StreamToken** of the stream.
linger	Set *true* (i.e., non-zero) if the data currently in the stream should be kept until it's read; set *false* to flush the data immediately.

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ StreamFlush()

```
StreamError StreamFlush (
          GeodeHandle         driver,
          StreamToken         stream);
```

This routine flushes all the data pending in a stream. It is passed the following arguments:

driver	The **GeodeToken** of the stream driver.
stream	The **StreamToken** of the stream.

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ StreamOpen()

```
StreamError StreamOpen (
          GeodeHandle         driver,
          word                buffSize,
          GeodeHandle         owner,
          HeapFlags           heapFlags,
          StreamToken *       stream);
```

This routine opens a stream. It is passed the following:

driver	The **GeodeToken** of the stream driver.
buffSize	The size of the stream buffer, in bytes.
owner	The geode which will own the stream.

heapFlags The flags for the creation of the buffer block.

**stream* The stream token will be written here.

If **StreamOpen()** is successful, it returns zero and writes the stream's token to **stream*. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ StreamQuery()

```
StreamError StreamQuery (
          GeodeHandle          driver,
          StreamToken          stream,
          StreamRoles          role,
          word *               bytesAvailable);
```

This routine finds out either how much free space is available in a stream's buffer, or how much data is waiting to be read. It is passed the following arguments:

driver The **GeodeToken** of the stream driver.

stream The **StreamToken** of the stream.

role If this is STREAM_ROLES_WRITER, the routine will return the amount of free space available in the stream buffer. If it is STREAM_ROLES_READER, it will return the amount of data waiting to be read.

**bytesAvailable*
The routine will write the number of bytes available (for writing or reading) to this variable.

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ StreamRead()

```
StreamError StreamRead (
          GeodeHandle          driver,
          StreamToken          stream,
          StreamBlocker        blocker,
          word                 buffSize,
          byte *               buffer,
          word *               numBytesRead);
```

This routine reads data from a stream. The routine takes the following arguments:

driver The **GeodeToken** of the stream driver.

stream	The **StreamToken** of the stream.
blocker	Specify whether to block if there is not enough data waiting to be read.
buffsize	Size of passed buffer (i.e. amount of data to read from stream).
buffer	Pointer to buffer where data from stream will be written.
**numBytesReadRead*	**StreamRead()** will write to this variable the number of bytes actually read from the stream.

If **StreamRead()** is successful, it returns zero. If it is unsuccessful, or could not read all the data requested from the stream, it returns a member of the **StreamError** enumerated type.

■ StreamReadByte()

```
StreamError StreamWriteByte (
          GeodeHandle         driver,
          StreamToken         stream,
          StreamBlocker       blocker,
          byte *              dataByte);
```

This routine reads a single byte from a stream. It takes the following arguments:

driver	The **GeodeToken** of the stream driver.
stream	The **StreamToken** of the stream.
blocker	Specify whether to block if there is not enough room to write the data.
**dataByte*	Read a byte from the stream, and write it to this variable.

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ StreamWrite()

```
StreamError StreamWrite (
          GeodeHandle         driver,
          StreamToken         stream,
          StreamBlocker       blocker,
          word                buffSize,
          const byte *        buffer,
          word *              numBytesWritten);
```

This routine writes data to a stream. The routine takes the following arguments:

driver	The **GeodeToken** of the stream driver.
stream	The **StreamToken** of the stream.
blocker	Specify whether to block if there is not enough room to write all the data.
buffsize	Size of passed data buffer (i.e. amount of data to write to stream).
buffer	Pointer to data to write to stream.
**numBytesWritten*	**StreamWrite()** will write to this variable the number of bytes actually written to the stream.

If **StreamWrite()** is successful, it returns zero. If it is unsuccessful, or could not write all the data to the stream, it returns a member of the **StreamError** enumerated type.

■ StreamWriteByte()

```
StreamError StreamWriteByte (
          GeodeHandle          driver,
          StreamToken          stream,
          StreamBlocker        blocker,
          byte                 dataByte);
```

This routine writes a single byte to a stream. It takes the following arguments:

driver	The **GeodeToken** of the stream driver.
stream	The **StreamToken** of the stream.
blocker	Specify whether to block if there is not enough room to write the data.
dataByte	Write this byte to the stream.

If the routine is successful, it returns zero. If it is unsuccessful, it returns a member of the **StreamError** enumerated type.

■ SysGetConfig()

```
dword     SysGetConfig();
```

This routine returns a set of values defining the system configuration. The returned dword contains four byte values, listed below from least significant byte to most significant byte:

configuration flags
This byte contains a record of **SysConfigFlags** reflecting the system status. This record includes information on how the system was started, whether Swat is running it, whether the system was restarted, etc.

reserved byte
This byte contains reserved information unusable by applications.

processor type
This byte contains a value reflecting the processor type of the machine running GEOS. This is of type **SysProcessorType** and is one of SPT_8088, SPT_8086, SPT_80186, SPT_80286, SPT_80386, or SPT_80486. Use the macro SGC_PROCESSOR to extract this value from the returned dword.

machine type
This byte contains a value of **SysMachineType** indicating the type of the machine running GEOS. It may be one of the following values: SMT_UNKNOWN, SMT_PC, SMT_PC_CONV, SMT_PC_JR, SMT_PC_XT, SMT_PC_XT_286, SMT_PC_AT, SMT_PS2_30, SMT_PS2_50, SMT_PS2_60, SMT_PS2_80, or SMT_PS1. Use the macro SGC_MACHINE to extract this value from the returned dword.

Include: system.h

■ SysGetDosEnvironment()

```
Boolean SysGetDosEnvironment( /* true if error (not found) */
        const char          * variable,  /* environment variable */
        char                * buffer,    /* buffer for return value */
        word                bufSize);    /* maximum return string length */
```

This routine looks up a specified DOS environment variable in the environment buffer. It takes three parameters:

variable A pointer to the null-terminated character string representing the name of the variable to be searched for.

buffer A pointer to a locked or fixed buffer in which the variable's value will be returned.

bufSize The size of the passed buffer in bytes (the maximum number of characters that can be returned including the terminating null character).

If the variable is not found, the error flag returned will be *true*.

Include: system.h

■ SysGetECLevel()

```
ErrorCheckingFlags SysGetECLevel(
          MemHandle * checksumBlock);
```

This routine checks the current error-checking level of the system. The returned record of **ErrorCheckingFlags** describes which levels of error checking are turned on and which are off. If checksum error checking (ECF_BLOCK_CHECKSUM) is on, pass a pointer to the handle of a block on which the checksum will be done.

Include: ec.h

■ SysGetInfo()

```
dword     SysGetInfo(
          SysGetInfoType info);   /* type of information to retrieve */
```

This routine returns general system information. Pass the type of information to be returned; the value returned depends on the type passed in *info*. Note that the largest returned value is a dword; many different return values should be cast to the appropriate type when calling **SysGetInfo()**.

The *info* parameter (of **SysGetInfoType**) can have one of the following values:

SGIT_TOTAL_HANDLES
Returns the total number of handles in the kernel's handle table.

SGIT_HEAP_SIZE
Returns the total heap size in bytes.

SGIT_LARGEST_FREE_BLOCK
Returns the size (in bytes) of the largest possible block that may be allocated at the moment.

SGIT_TOTAL_COUNT
Returns the total number of clock ticks since the current session of GEOS started (subtracts the initial system clock value from the current time).

SGIT_NUMBER_OF_VOLUMES
Returns the total number of volumes registered with the system.

SGIT_TOTAL_GEODES
Returns the total number of geodes currently loaded.

SGIT_NUMBER_OF_PROCESSES
Returns the total number of processes currently loaded.

SGIT_NUMBER_OF_LIBRARIES
Returns the total number of libraries currently loaded.

SGIT_NUMBER_OF_DRIVERS
Returns the total number of drivers currently loaded.

SGIT_CPU_SPEED
Returns the CPU speed of the processor. The value returned will be ten times the ratio of the CPU speed relative to a base XT processor.

SGIT_SYSTEM_DISK
Returns the disk handle of the disk on which GEOS (the GEOS.INI file) resides.

SGIT_UI_PROCESS

Include: sysstats.h

■ SysGetPenMode()

```
Boolean  SysGetPenMode();
```

This routine returns true if GEOS is running on a pen-based system, false if it is not.

Include: system.h

■ SysLocateFileInDosPath()

```
DiskHandle SysLocateFileInDosPath( /* sets thread's error value */
          const char          * fname,     /* file name */
          char                * buffer);   /* returned path of file */
```

This routine searches for a specified file along the search path specified in the DOS environment variable PATH. The parameters are

fname	A pointer to the null-terminated file name to search for.
buffer	A pointer to a locked or fixed buffer into which the full path of the file will be placed.

This routine returns the disk handle of the disk on which the file resides as well as the file's full path (with drive name) in the buffer pointed to by *buffer*. The path returned is a null-terminated character string. If the file could not be found, a null disk handle will be returned. The error value can be retrieved with **ThreadGetError()**.

Include: system.h

■ SysNotify()

```
word      SysNotify(
          SysNotifyFlags     flags,       /* options to offer user */
          const char         * string1,   /* first string to display */
          const char         * string2);  /* second string to display */
```

This routine causes the kernel to put up a standard notification dialog box on the screen. This dialog box is white with a black border and is used nearly exclusively for error notification by the kernel. Pass this routine the following parameters:

flags	A record of **SysNotifyFlags** indicating the options the dialog presents to the user. These flags are shown below.
string1	A pointer to a null-terminated character string put up in the dialog box (may be a null pointer).
string2	A pointer to a second null-terminated string presented in the dialog box (may be a null pointer).

The returned word is the user's response, based on the **SysNotifyFlags** passed (see below).

Structures: **SysNotifyFlags** is a record of several flags; none, any, or all of the flags may be set at a time. The five flags are

SNF_RETRY	Allow the user to retry the operation that brought up the notification box. If the user selects this option, it will be returned by the routine.
SNF_EXIT	Allow the user to exit GEOS entirely. If the user selects this option, it will be returned by the routine after an SST_CLEAN_FORCED shutdown has been initiated.
SNF_ABORT	Allow the user to abort the operation that brought up the notification box. If the user selects this option, it will be returned by the routine.
SNF_CONTINUE	Allow the user to continue the operation. If the user selects this option, it will be returned by the routine.
SNF_REBOOT	Allow the user to shut down and reboot GEOS directly. If the user selects this option, the routine will not return.

Include: system.h

■ SysRegisterScreen()

```
void      SysRegisterScreen(
          GeodeHandle         driver,
          WindowHandle        root);
```

■ SysSetECLevel()

```
void      SysSetECLevel(
          ErrorCheckingFlags  flags,           /* level of error checking */
          MemHandle           checksumBlock);  /* block to check, if any */
```

This routine sets the error-checking level of the software. Pass it a record of **ErrorCheckingFlags** indicating which levels of error checking should be employed. If checksum checking (ECF_BLOCK_CHECKSUM) is turned on, also pass the handle of a block on which the checksum will be performed.

Include: ec.h

■ SysSetExitFlags()

```
word      SysGetExitFlags(
          ExitFlags           bitsToSet,
          ExitFlags           bitsToClear);
```

■ SysShutdown()

```
Boolean   SysShutdown(
          SysShutdownType type,
          ...);
```

This routine causes the system to shut down, exiting to the native operating system (typically DOS). It takes variable parameters depending on the first parameter. The first parameter is the type of shutdown requested, and it determines the calling format of the routine. **SysShutdown()** returns a Boolean value dependent on the type of shutdown.

The parameters and calling format for this routine depend on the value in the *type* parameter. The possible values (**SysShutdownType**) are listed below with the associated parameter and return information.

SST_CLEAN Shut down all applications cleanly, allowing any that wish to to abort the shutdown. The routine will return *true* if a system shutdown is already in progress at the time of the call. This type of shutdown will send MSG_META_CONFIRM_SHUTDOWN to all objects registered on the MANUFACTURER_ID_GEOWORKS:GCNSLT_SHUTDOWN_CONTROL GCN list (but only if the shutdown is not cancelled). Each object on that list must return an acknowledgment of the shutdown. The parameter format and parameters are

```
Boolean SysShutdown(
        SysShutdownType     type,
        optr                notificationOD,
        Message             msg);
```

notificationOD
The optr of an object which will receive the message passed in *msg* after the shutdown has been acknowledged. Pass a null optr to use the default notification (MSG_META_DETACH sent to the UI).

msg The message to be sent to the object in *notificationOD*.

SST_CLEAN_FORCED
Shut down all applications cleanly without the possibility of cancellation. This type takes no additional parameters and does not allow other geodes to abort the shutdown. It will return, but the return value will be meaningless.

SST_DIRTY Attempt to exit device drivers and close all files without shutting down applications. Does not return. The parameters of this type are

```
Boolean SysShutdown{
        SysShutdownType     type,        /* SST_DIRTY */
        const char          * reason);
```

The *reason* parameter is a pointer to a text string presented to the user as a reason for the dirty shutdown. The string is null-terminated. Pass -1 if no reason is to be given.

SST_PANIC Exit system device drivers (GA_SYSTEM) without exiting applications or closing files. This can be bad for the system and should be used only in emergency situations. This type of shutdown takes no additional parameters and does not return.

SST_REBOOT
This is used by GEOS when the user hits *Ctrl-Alt-Del*. Applications should not call it.

SST_RESTART
This is like SST_CLEAN_FORCED above, but it reloads GEOS after shutting down rather than exit completely. It takes no additional parameters; it will return TRUE if the system could not be restarted, FALSE if the shutdown has been initiated.

SST_FINAL Perform the final phase of a shutdown. This routine is called *only* by the UI when the SST_CLEAN_FORCED shutdown is complete. This type does not return, and it takes one additional parameter. The calling format and parameters of this type are

```
Boolean SysShutdown(
        SysShutdownType     type,
        const char          * reason);
```

The *reason* parameter is a character string explaining the reason (typically an error) for the final shutdown.

SST_SUSPEND
: Suspend system operation in preparation for task switching, and broadcast MSG_META_CONFIRM_SHUTDOWN to all objects on the MANUFACTURER_ID_GEOWORKS:GCNSLT_SHUTDOWN_CONTROL GCN list (see **MetaClass**). All notified objects must return acknowledgment of the shutdown. This type of **SysShutdown()** returns *true* if another system shutdown is already in progress. It takes two additional parameters:

```
Boolean SysShutdown(
        SysShutdownType     type,
        optr                notificationOD,
        Message             msg);
```

notificationOD
: The optr of an object which will receive the message passed in *msg* after the shutdown has been acknowledged. Pass a null optr to use the default notification (MSG_META_DETACH sent to the UI), though this is not usually the intent of the call.

msg
: The message to be sent to the object in *notificationOD*.

SST_CONFIRM_START
: Called by the recipient of MSG_META_CONFIRM_SHUTDOWN; this allows shutdown confirmation dialog boxes to be presented in order to the user. The caller of this type will be blocked until all previous callers have finished their confirmation procedure. When **SysShutdown()** returns, the caller may present its confirmation dialog and continue or abort the shutdown. If **SysShutdown()** returns *true* from a call with this type, the caller should *not* present the confirmation dialog to the user and need not call **SysShutdown()** with SST_CONFIRM_END; another thread has already cancelled the shutdown. This type takes no additional parameters.

SST_CONFIRM_END
: The counterpart of SST_CONFIRM_START, this ends the confirmation sequence in an object's MSG_META_CONFIRM_SHUTDOWN handler. It takes one additional parameter and returns nothing. The calling format is shown below:

```
void    SysShutdown(
        SysShutdownType     type,
        Boolean             confirm);
```

The *confirm* parameter should be TRUE if the shutdown is to be continued, FALSE if the shutdown should be aborted.

Include: system.h

Warnings: Most applications should not call **SysShutdown()**. Any that do should do so with extreme care.

■ SysStatistics()

```
void      SysStatistics(
          SysStats * stats);     /* returned statistics */
```

This routine returns system performance statistics. Pass it a pointer to an empty **SysStats** structure; the routine will fill in the appropriate fields. **SysStats** has the following structure:

```
typedef struct {
    dword           SS_idleCount;
    SysSwapInfo     SS_swapOuts;
    SysSwapInfo     SS_swapIns;
    word            SS_contextSwitches;
    word            SS_interrupts;
    word            SS_runQueue;
} SysStats;
```

Include: sysstats.h

■ SysUnlockBIOS()

```
void      SysUnlockBIOS(void);
```

■ TextSearchInString()

```
char *    TextSearchInSTring(
          const char          *str1,
          conat char          *startPtr,
          const char          *endPtr,
          word                strSize,
          const char          *str2,
          word                str2Size,
          word                searchOptions,
          word                *matchLen);
```

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 passed buffers.

str1 is a pointer to the main string you will be searching in.

startPtr and *endPtr* are pointers to locations within *str1* to begin and end the search.

strSize stores the size of *str1*, or zero if null-terminated.

str2 stores the match string, which may include wildcards (type **WildCard**).

str2Size stores the size of *str2*, or zero if null-terminated.

searchOptions stores the **SearchOptions** to use by the search mechanism. The high byte should be zeroed.

matchLen stores a buffer to store the size of the matched word. (The matched word itself is returned by the routine.)

Include: Objects/vTextC.goh

■ TextSearchInHugeArray()

```
dword    TextSearchInSTring(
         char                *str2,
         word                str2Size,
         dword               str1Size,
         dword               curOffset,
         dword               endOffset,
         FileHandle          hugeArrayFile,
         VMBlockHandle       hugeArrayBlock,
         word                searchOptions,
         word                *matchLen);
```

TextSearchInHugeArray() searches in a huge array for a passed text string. If a match is found, a dword offset to the match (and the length of the match) are returned in passed buffers.

str2 stores the match string, which may include wildcards (type **WildCard**).

str2Size stores the size of *str2*, or zero if null-terminated.

str1Size stores the total length of the string being searched.

curOffset stores the offset from the start of str1 to the first character to check.

endOffset stores the offset from the start of str1 to the last character to check.

hugeArrayFile stores the file handle of the huge array.

hugeArrayBlock stores the VM block handle of the huge array.

searchOptions stores the **SearchOptions** to use by the search mechanism. The high byte should be zeroed.

matchLen stores a buffer to store the size of the matched word. (The matched word itself is returned by the routine.)

Include: Objects/vTextC.goh

■ TGI_PRIORITY()

```
byte      TGI_PRIORITY(val);
          word   val;
```

This macro extracts the thread priority from the value returned by **ThreadGetInfo()**.

■ TGI_RECENT_CPU_USAGE()

```
byte      TGI_RECENT_CPU_USAGE(val);
          word   val;
```

This macro extracts the recent CPU usage from the value returned by **ThreadGetInfo()**.

■ ThreadAllocSem()

```
SemaphoreHandle ThreadAllocSem(
          word   value);            /* allowable locks on the semaphore */
```

This routine allocates and initializes a new semaphore for private use by a multithreaded application. Pass the value with which to initialize the semaphore; this value represents the number of threads that can grab the semaphore before other grab attempts will block. Typically, the passed value will be one. The routine returns the handle of the new semaphore.

Include: sem.h

■ ThreadAllocThreadLock()

```
ThreadLockHandle ThreadAllocThreadLock();
```

This routine allocates a special semaphore called a thread lock. With a normal semaphore, a thread that grabs the semaphore twice without releasing it will deadlock; with a thread lock, a thread can grab it more than once in succession. The thread has to release it once for each time it grabs the thread lock, however.

In all other aspects, however, the thread lock resembles a normal semaphore. **ThreadAllocThreadLock()** returns the handle of the new thread lock.

Include: sem.h

■ ThreadAttachToQueue()

```
void      ThreadAttachToQueue(
          QueueHandle        qh,           /* queue to attach */
          ClassStruct        * class);     /* primary class of thread */
```

This routine attaches the calling thread to the passed event queue. This is used only for event-driven threads. Typically, this routine is called when a thread is created; attaching to queues is automatic in nearly all cases, and you will rarely need this routine.

Pass the handle of the queue in *qh* and a class pointer in *class*. The class will be attached to the event queue and will handle all messages sent directly to the thread. This class should nearly always be a subclass of **ProcessClass**.

If a queue handle of zero is passed, the thread wants to “reattach” to the current queue. This is used typically during shutdown of event-driven threads, and it is nearly always taken care of automatically by **ProcessClass**.

Include: thread.h

■ ThreadCreate()

```
ThreadHandle ThreadCreate(
          word   priority,          /* Initial base priority of new thread */
          word   valueToPass,       /* Optional data to pass to new thread */
          word   (*startRoutine)(word valuePassed),
                                /* Pointer to entry routine */
          word   stackSize,         /* Size of the stack for the new thread */
          GeodeHandle owner);       /* Geode that will own the new thread */
```

This routine creates a new procedural thread for a process. If you need a new event-driven thread, send MSG_PROCESS_CREATE_EVENT_THREAD to your process object instead.

Pass the following parameters to this routine:

priority The priority of the new thread. Typically this will be one of the standard thread priorities (see below).

valueToPass
A word of optional data to be passed to the entry routine of the new thread. This can be used, for example, to indicate the thread’s initial context or for initializing thread variables.

startRoutine
A pointer to the entry routine to be executed immediately for the thread. This may be in either fixed or movable memory. The

segment must be a virtual segment. Note that if the routine is in movable memory, it may degrade heap performance for the life of the thread (its movable block will remain locked for extended stretches of time). The routine may return the thread's exit code or may call **ThreadDestroy()** directly.

stackSize The stack size allocated for the thread. 512 bytes is typically enough for threads doing neither UI nor file system work; threads working with the file system will require 1 K. Threads working with UI objects will require 3 K.

owner The geode handle of the geode that will own the thread. If the calling thread's geode will own the new thread, it can call **GeodeGetProcessHandle()** prior to calling **ThreadCreate()**.

ThreadCreate() returns the thread handle of the new thread. If an error occurs, the calling thread's error code will be set and a null handle returned; you should likely call **ThreadGetError()** to retrieve the error code after creating the new thread. A return of NO_ERROR_RETURNED from **ThreadGetError()** means no error occurred.

The standard thread priorities that may be passed in the *priority* parameter are listed below:

PRIORITY_TIME_CRITICAL
The highest priority of all; you should not use this in general because it will pre-empt nearly all other threads. (It may be useful, however, during debugging.)

PRIORITY_HIGH
A high priority; generally only used for highly important threads.

PRIORITY_UI
Another high priority; this is used for User Interface threads to provide quick response to user actions.

PRIORITY_FOCUS
A medium-level priority; this is used for whatever thread has the current input focus (whichever thread the user is currently working with).

PRIORITY_STANDARD
The standard application thread priority; you should typically use this when creating new threads.

PRIORITY_LOW
A low priority for tasks that can be done in the background.

PRIORITY_LOWEST
The lowest standard priority; it is used for threads that can take any amount of time to complete.

Include: thread.h

■ ThreadDestroy()

```
void      ThreadDestroy(
          word   errorCode,    /* Error code to indicate cause of destruction */
          optr   ackObject,    /* Object to receive destruction acknowledgment */
          word   ackData);     /* Additional word of data to pass (as the low
                                * word of optr for source of MSG_META_ACK) */
```

This routine causes the current (calling) thread to exit and then destroy itself. The thread is responsible for ensuring that it has no leftover resources allocated or semaphores locked.

Pass it an error code or exit code meaningful to the application and the other threads in the application. This error code will be used by the debugger to determine the cause of the thread's exit; a null error code usually indicates successful completion of the thread's task.

Pass also the optr of the object to receive acknowledgement of the thread's destruction. The object specified will receive MSG_META_ACK after the calling thread is completely destroyed.

Be Sure To: Always clean up before exiting a thread. Unlock locked resources, free allocated memory, etc. You do not have to do these things for the application's primary thread; the process object (the primary thread) will automatically clean up after itself.

Include: thread.h

■ ThreadFreeSem()

```
void      ThreadFreeSem(
          SemaphoreHandle sem);   /* semaphore to be freed */
```

This routine frees the specified semaphore that had been allocated with **ThreadAllocSem()**. You must be sure that no threads are using the semaphore or will use it after it has been freed. Subsequent access attempts could cause illegal handle errors or worse.

Include: sem.h

■ ThreadFreeThreadLock()

```
void      ThreadFreeThreadLock(
          ThreadLockHandle sem);     /* thread lock to be freed */
```

This routine frees the specified thread lock that had been allocated with **ThreadAllocThreadLock()**. You must be sure that no threads are using or will use the thread lock after it has been freed. Subsequent attempts to grab or release the thread lock could cause illegal handle errors.

Include: sem.h

■ ThreadGetError()

```
word      ThreadGetError(void)
```

This routine returns the thread's current error value.

■ ThreadGetInfo()

```
word      ThreadGetInfo(
          ThreadHandle        th,       /* thread to get information about */
          ThreadGetInfoType   info);    /* type of information to get */
```

This routine gets information about the specified thread. The information desired is specified in the *info* parameter; the subject thread is specified in the *th* parameter. If the thread handle passed is zero or a null handle, the routine will return information about the calling thread.

The *info* parameter is one of the following values of **ThreadGetInfoType**, specifying the type of information to be returned by **ThreadGetInfo()**:

TGIT_PRIORITY_AND_USAGE
: The returned word will contain both the thread's priority and the thread's recent CPU usage. To extract the priority of the thread, use the macro TGI_PRIORITY; to extract the recent CPU usage, use the macro TGI_RECENT_CPU_USAGE.

TGIT_THREAD_HANDLE
: Useful only when the *th* parameter is zero, this will return the thread handle of the subject thread. If *th* is zero, the handle of the calling thread will be returned.

TGIT_QUEUE_HANDLE
: The returned word will contain the queue handle of the event-driven thread specified in *th*. If the thread specified is not event-driven, a null queue handle will be returned.

Include: thread.h

■ ThreadGrabThreadLock()

```
void      ThreadGrabThreadLock(
          ThreadLockHandle sem);     /* thread lock to grab */
```

This routine attempts to grab the thread lock for the calling thread. If the thread lock is currently held by another thread, the caller will block until the lock becomes available. If the caller already has the thread lock, it will grab the lock again and continue executing.

Be Sure To: Thread locks must be released with **ThreadReleaseThreadLock()** once for each time they are grabbed.

Warnings: This routine provides no deadlock protection for multiple threads. If multiple threads will be grabbing multiple thread locks, the locks should always be grabbed in the same order to minimize the potential for deadlock.

Include: sem.h

■ ThreadHandleException()

```
void      ThreadHandleException(
          ThreadHandle        th,            /* thread to handle the exception */
          ThreadExceptions    exception,     /* exception to handle */
          void   (*handler)   ());           /* pointer to handler */
```

This routine allows a thread to set up a handler for a processor exception. This can be useful for debugging purposes. Pass the following three parameters:

th	The handle of the thread to handle the exception. Pass zero for the current thread.
exception	A **ThreadException** type (see below).
handler	A pointer to a handler in fixed or locked memory. Pass a null pointer to use the GEOS default exception handler.

Structures: The **ThreadException** type has the following values:

```
TE_DIVIDE_BY_ZERO
TE_OVERFLOW
TE_BOUND
TE_FPU_EXCEPTION
TE_SINGLE_STEP
TE_BREAKPOINT
```

Include: thread.h

■ ThreadModify()

```
void        ThreadModify(
            ThreadHandle        th,               /* thread to modify */
            word                newBasePriority, /* thread’s new base priority */
            ThreadModifyFlags   flags);          /* flags (see below) */
```

This routine modifies the priority of the specified thread. Use it to either set the base priority of the thread or reset the current CPU usage to zero. The parameters should have the following values:

th The thread handle; pass zero to change the priority of the calling thread.

newBasePriority
The new base priority of the thread. Use one of the standard priorities—see **ThreadCreate()**—or use a value between zero and 255.

flags A record of **ThreadModifyFlags**; pass TMF_BASE_PRIO to change the thread’s base priority or TMF_ZERO_USAGE to reset the thread’s recent CPU usage to zero.

Warnings: Unless the thread is timing-critical, you should not set the base priority to zero.

Include: thread.h

■ ThreadPrivAlloc()

```
word        ThreadPrivAlloc(
            word                wordsRequested,  /* number of words to allocate */
            GeodeHandle         owner);          /* handle of geode to own data */
```

This routine allocates a number of contiguous words in the private data of all geodes (loaded and yet-to-be loaded). It is exactly the same as **GeodePrivAlloc()**; see the entry for that routine.

Include: thread.h

See Also: GeodePrivAlloc()

■ ThreadPrivFree()

```
void        ThreadPrivFree(
            word   range,               /* offset to first word to be freed */
            word   wordsRequested);     /* number of words to free */
```

This routine frees a number of contiguous private-data words previously allocated with **ThreadPrivAlloc()**. It is similar to **GeodePrivFree()**; see the entry for that routine for full information.

Include: thread.h

See Also: GeodePrivFree()

■ ThreadPSem()

```
SemaphoreError ThreadPSem(
            SemaphoreHandle sem);      /* semaphore to grab */
```

This routine attempts to grab the passed semaphore via a “P” operation. If the semaphore has already been grabbed, the thread will block until the semaphore becomes available, even if it was grabbed by the same thread.

ThreadPSem() returns an error code of type **SemaphoreError**, described in **ThreadPTimedSem()**, below. The error code is intended to indicate abnormal return by the previous thread; if the semaphore never becomes available, the thread will block indefinitely and the routine will not return.

Be Sure To: When the thread no longer needs the semaphore, it should release it with **ThreadVSem()**.

Warnings: This routine provides no deadlock protection. If threads will grab multiple common semaphores, they should always grab/release them in the same order, minimizing the chances for deadlock.

A thread may not try to grab a particular semaphore twice without releasing it in between grabs. The thread will block on itself and will deadlock. If a thread may need to grab the semaphore twice in a row, it should use a thread lock instead (see **ThreadAllocThreadLock()** for more information).

Include: sem.h

■ ThreadPTimedSem()

```
SemaphoreError ThreadPTimedSem(
          SemaphoreHandle     sem,          /* semaphore to grab */
          word                timeout);     /* ticks before timeout */
```

This routine attempts to grab the passed semaphore via a “P” operation. If the semaphore has already been grabbed, the thread will block for at most the number of ticks specified in *timeout*.

ThreadPTimedSem() returns an error code of type **SemaphoreError**, which has two value:

SE_NO_ERROR
: No error occurred and the semaphore was grabbed properly.

SE_TIMEOUT
: The time elapsed and the semaphore was not grabbed. If this value is returned, the thread should *not* proceed with whatever protected operation was to happen. Instead, it should either attempt to grab the semaphore again or should proceed with other tasks.

SE_PREVIOUS_OWNER_DIED
: The previous owner of the semaphore exited abnormally. If the thread currently holding the semaphore exited without releasing the semaphore, for example, this would be returned.

Often *timeout* is passed as zero to indicate that if the semaphore isn’t available right now, the thread will go on with some other action.

Be Sure To: When the thread no longer needs the semaphore, it should release it with **ThreadVSem()**.

Warnings: This routine provides no deadlock protection. If threads will grab multiple common semaphores, they should always grab/release them in the same order, minimizing the chances for deadlock.

A thread may not try to grab a particular semaphore twice without releasing it in between grabs. The thread will block on itself and will deadlock. If a thread may need to grab the semaphore twice in a row, it should use a thread lock instead, though there is no timeout equivalent for thread locks (see **ThreadAllocThreadLock()** for more information).

Include: sem.h

■ ThreadReleaseThreadLock()

```
void      ThreadReleaseThreadLock(
          ThreadLockHandle sem);     /* threadlock to release */
```

This routine releases the specified thread lock previously grabbed with **ThreadGrabThreadLock()**. Pass the handle of the thread lock as returned by **ThreadAllocThreadLock()**.

Do not try to release a thread lock that has not previously been grabbed. The results are unpredictable.

Include: sem.h

■ ThreadVSem()

```
void      ThreadVSem(
          SemaphoreHandle sem);      /* semaphore to release */
```

This routine releases a semaphore that was grabbed with **ThreadPSem()** or **ThreadPTimedSem()**. Pass the handle of the semaphore as returned by **ThreadAllocSem()**.

Do not try to release a semaphore that has not previously been grabbed with one of the above routines. The results are unpredictable.

Include: sem.h

■ TimerGetCount()

```
dword     TimerGetCount();
```

This routine returns the value of the system counter. The returned value is the number of ticks since GEOS started.

Include: timer.h

■ TimerGetDateAndTime()

```
void      TimerGetDateAndTime(
          TimerDateAndTime * dateAndTime); /* buffer for returned values */
```

This routine returns the current time and date. Pass it a pointer to an empty **TimerDateAndTime** structure to be filled in by the routine.

Include: timedate.h

■ TimerSetDateAndTime()

```
void      TimerSetDateAndTime(
          word                    flags,          /* which item to set */
          const TimerDateAndTime  * dateAndTime); /* new values */
```

This routine sets the current date and/or time of the system. Pass it the following:

flags A word of flags. Pass TIME_SET_DATE to set the day, month, and year; pass TIME_SET_TIME to set the hour, minute, and second. Pass both to set both.

dateAndTime
A pointer to a **TimerDateAndTime** structure containing the information to be set.

Include: timedate.h

■ TimerSleep()

```
void      TimerSleep(
          word   ticks);        /* number of ticks the thread should sleep */
```

This routine invokes a “sleep timer” that will put the calling thread to sleep for the given number of ticks. At the end of the time, the thread will continue executing with the next instruction.

Warnings: Do not use sleep timers as a substitute for semaphores for thread synchronization.

Include: timer.h

■ TimerStart()

```
TimerHandle TimerStart(
          TimerType           timerType,    /* type of timer to start */
          optr                destObject,   /* object to receive notification
                                         * message when timer expires */
          word                ticks,        /* amount of time to run */
          Message             msg,          /* notification message */
          word                interval,     /* interval for continual timers */
          word                * id);        /* buffer for returned timer ID */
```

This routine starts a timer of any type. The timer will run for the specified number of ticks and then will send the given message to the destination object. The message is sent with the flags MF_FORCE_QUEUE, MF_CHECK_DUPLICATE and MF_REPLACE, so it will always be put in the recipient’s queue and will always replace any duplicates already in the queue. Pass this routine the following:

timerType	A value of **TimerType** indicating the type of timer to start.
destObject	The optr of the object that will be sent the specified message when the time is up.
ticks	The number of ticks for the timer to run. (Sixty ticks equals one second.)
msg	The message to be sent to the destination object when time is up.
interval	For continual timers, the interval (number of ticks) at which to send out the message to the destination object. The timer will send the message once at the end of each interval. The first message will be sent *ticks* ticks after the timer is started. The second message will be sent *interval* ticks after that.
id	A pointer to a word in which the timer's ID will be returned. You will need this ID for **TimerStop()**.

This routine returns the handle of the timer as well as an ID pointed to by the *id* parameter. You will need the handle and the ID for **TimerStop()**.

TimerType: The **TimerType** enumerated type defines what type of timer should be initiated. It has the following values:

TIMER_ROUTINE_ONE_SHOT
: Start a timer that will call a routine and then free itself when the time is expired. This type is supported in assembly but not in C.

TIMER_ROUTINE_CONTINUAL
: Start a timer that will call a routine once per time interval until **TimerStop()** is called. This type is supported in assembly but not in C.

TIMER_EVENT_ONE_SHOT
: Start a timer that will send a message to a given object, then free itself, when time is expired.

TIMER_EVENT_CONTINUAL
: Start a timer that will send a message to a given object once per time interval until **TimerStop()** is called.

TIMER_MS_ROUTINE_ONE_SHOT
: Start a timer that has millisecond accuracy. For this timer, the number of ticks will actually be the number of milliseconds. The timer will call a specified routine and then free itself when time is expired. This type is supported in assembly but not in C.

Routines

TIMER_EVENT_REAL_TIME
Start a timer that will call a routine at some particular date and time. On devices that support such a timer, this event will wake a sleeping machine.

Include: timer.h

■ TimerStop()

```
Boolean TimerStop(
        TimerHandle         th,       /* handle of timer to be stopped */
        word                id);      /* timer ID (returned by TimerStart() */
```

This routine stops a timer that had been started with **TimerStart()**. Pass it the timer handle and the ID as returned by that routine (the ID of a continual timer will always be zero).

The returned error flag will be *true* if the timer could not be found.

Warnings: If you call **TimerStop()** to stop a continual timer that sends its message across threads, there may be timer events left in the recipient's event queue. It is unsafe in this situation to assume that all timer events have been handled. To ensure the timer message has been handled, you can send the destination an "all-safe" message with the MF_FORCE_QUEUE flag.

Include: timer.h

■ TocDBLock()

```
void * TocDBLock(
        DBGroupAndItem      thing);
```

Use this routine to lock a name array maintained by a PrefTocList object.

Include: config.goh

■ TocDBLockGetRef()

```
void * TocDBLockGetRef(
        DBGroupAndItem      thing,
        optr                *refPtr);
```

This routine locks a name array maintained by a PrefTocList object, returning the item's pointer and optr.

Include: config.goh

■ TocFindCategory()

```
Boolean TocFindCategory(
          TocCategoryStruct   *cat);
```

This routine searches a PrefTocList object's name lists for a given token.

Structures:

```
          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;
```

Include: config.goh

■ TocGetFileHandle()

```
word TocGetFileHandle();
```

Use this routine to get the handle of the file used by PrefTocLists to store their name array data.

Include: config.goh

■ TocNameArrayAdd()

```
word TocNameArrayAdd(
          DBGroupAndItem      array,
          const char          *nameToFind,
          const void          *data);
```

Use this routine to add a name to a name array maintained by a PrefTocList object.

Include: config.h

■ TocNameArrayFind()

```
word TocNameArrayGetElement(
          DBGroupAndItem      array,
          word                element,
          void                *buffer);
```

Use this routine to find a name in the name list maintained by a PrefTocList object.

Include: config.goh

■ TocNameArrayGetElement()

```
word TocNameArrayGetElement(
          DBGroupAndItem      array,
          word                element,
          void                *buffer);
```

Use this routine to retrieve a given element from a name array maintained by a PrefTocList object.

Include: config.goh

■ TocSortedNameArrayAdd()

```
word TocSortedNameArrayAdd(
          word                arr,
          const char          *nameToAdd,
          NameArrayAddFlags   flags,
          const void          *data);
```

This routine adds a name to a sorted name array associated with a PrefTocList object.

Structures:

```
          typedef WordFlags NameArrayAddFlags;
          #define NAAF_SET_DATA_ON_REPLACE 0x8000
```

Include: config.goh

■ TocSortedNameArrayFind()

```
Boolean TocSortedNameArrayFind(
          word                        arr,
          const char                  *nameToFind,
          SortedNameArrayFindFlags    flags,
          void                        *buffer,
          word                        *elementNum);
```

This routine looks up a name in a sorted name array associated with a PrefTocList object.

Structures:

```
          typedef WordFlags SortedNameArrayFindFlags;
          #define SNAFF_IGNORE_CASE 0x0080
```

Include: config.goh

■ TocUpdateCategory()

```
void TocUpdateCategory(
          TocUpdateCategoryParams *params);
```

Use this routine to update a PrefTocList object based upon the files in a given directory with a given token.

Structures:

```
          typedef struct {
                TocUpdateCategoryFlags  TUCP_flags;
                TokenChars              TUCP_tokenChars;
                byte                    TUCP_fileArrayElementSize;

                TocUpdateAddCallback    *TUCP_addCallback;
                byte                    TUCP_pad; /* Wants to be word-aligned */
          } TocUpdateCategoryParams;

          typedef word _pascal TocUpdateAddCallback(
                const char *filename,
                optr chunkArray);
          /* Return 0 if add aborted, else return offset of new element within
           * block */
```

Include: config.goh

■ TOKEN_CHARS()

```
dword     TOKEN_CHARS(a, b, c, d);
          char   a, b, c, d;
```

This macro creates a single dword value from four given characters. This is useful when creating a token characters value for a specific token.

■ TokenDefineToken()

```
word      TokenDefineToken(
          dword              tokenChars,       /* four token characters */
          ManufacturerID     manufacturerID,   /* manufacturer ID for token */
          optr               monikerList,      /* optr of moniker list */
          TokenFlags         flags);           /* token flags */
```

This routine adds a new token and moniker list to the token database. If the token already exists in the token DB, the old will be replaced with the new. This routine must only be called by a thread that can lock the block in which the passed moniker or moniker list resides. This routine must be passed the following parameters:

tokenChars The four token characters that identify this moniker or moniker list in the token database. Create this dword value from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

monikerList
The optr of the moniker list to be added to the token database.

flags A record of **TokenFlags** indicating the relocation status of the moniker list.

Warnings: This routine may legally move locked LMem blocks (token database items), thereby invalidating all pointers to token database items.

Include: token.h

■ TokenGetTokenInfo()

```
Boolean TokenGetTokenInfo(
        dword               tokenChars,      /* four characters of token */
        ManufacturerID      manufacturerID,  /* manufacturer ID of token */
        TokenFlags          * flags);        /* returned token flags */
```

This routine returns information about a specified token. Pass it the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

flags A pointer to an empty flags record; the flags set (if any) for the specified token (if it exists) will be returned here.

This routine returns a (non-zero) value of **VMStatus** if the token was not found in the token database. It returns zero if successful.

Include: token.h

■ TokenListTokens()

```
dword    TokenListTokens(
         TokenRangeFlags     tokenRangeFlags,
         word                headerSize,
         ManufacturerID      manufacturerID));
```

This routine lists all the tokens in the token database. It allocates a new block on the global heap and writes in it an array of **GeodeToken** structures. This routine returns the actual tokens, not the token groups.

The returned dword consists of two values: The high word represents the number of tokens in the returned block and may be extracted with the macro **TokenListTokensCountFromDWord()**. The low word represents the handle of the newly-allocated block and can be extracted with the macro **TokenListTokensHandleFromDWord()**.

Include: token.h

■ TokenListTokensCountFromDWord()

```
word     TokenListTokensCountFromDWord(d);
         dword  d;
```

This macro extracts the number of tokens from the value returned by **TokenListTokens()**.

■ TokenListTokensHandleFromDWord()

```
word     TokenListTokensHandleFromDWord(d);
         dword  d;
```

This routine extracts the MemHandle from the value returned by **TokenListTokens()**.

■ TokenLoadMonikerBlock()

```
Boolean  TokenLoadMonikerBlock(
         dword               tokenChars,       /* four characters of token */
         ManufacturerID      manufacturerID,   /* manufacturer ID of token */
         DisplayType         displayType,      /* type of display for token */
         VisMonikerSearchFlags searchFlags,    /* flags for finding token */
         word                * blockSize,      /* returned block size */
         MemHandle           * blockHandle);   /* returned block handle */
```

This routine loads a specified token's moniker, allocating a new global memory block for the moniker. The returned Boolean will be *false* if the moniker was found, *true* otherwise. Information about the moniker is returned in the values pointed to by *blockSize* (the size of the newly allocated block) and *blockHandle* (the handle of the new block). If the moniker is not found, both return pointers will be NULL and no block will be allocated.

Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

displayType
A value of **DisplayType** indicating the size of the display (used to indicate small-screen devices, primarily).

searchFlags
A record of **VisMonikerSearchFlags** indicating what type of moniker is being requested.

blockSize A pointer to a word in which the new block's size will be returned.

blockHandle
A pointer to a handle in which the new block's handle will be returned.

Include: token.h

■ TokenLoadMonikerBuffer()

```
Boolean  TokenLoadMonikerBuffer(
         dword                tokenChars,      /* four characters of token */
         ManufacturerID       manufacturerID,  /* manufacturer ID of token */
         DisplayType          displayType,     /* type of display for token */
         VisMonikerSearchFlags searchFlags,    /* flags for finding token */
         void                 * buffer,        /* pointer to buffer for token */
         word                 bufSize,         /* size of passed buffer */
         word                 * bytesReturned);   /* number of bytes returned */
```

This routine loads a specified token's moniker into a provided buffer. The return value will be *false* if the moniker was found, *true* otherwise. The size of the returned moniker will be returned in the word pointed to by the *bytesReturned* parameter.

Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

displayType
A value of **DisplayType** indicating the size of the display (used to indicate small-screen devices, primarily).

searchFlags
A record of **VisMonikerSearchFlags** indicating what type of moniker is being requested.

buffer A pointer to a locked or fixed buffer into which the moniker will be copied.

bufSize The size of the passed buffer; also the maximum size of the moniker that may be returned.

bytesReturned
The size of the moniker actually returned in the buffer.

Include: token.h

■ TokenLoadMonikerChunk()

```
Boolean  TokenLoadMonikerChunk(
         dword                tokenChars,      /* four characters of token */
         ManufacturerID       manufacturerID,  /* manufacturer ID of token */
         DisplayType          displayType,     /* type of display for token */
         VisMonikerSearchFlags searchFlags,    /* flags for finding token */
         MemHandle            lmemBlock,       /* locked block for new chunk */
         word                 * chunkSize,     /* returned new chunk size */
         ChunkHandle          * chunkHandle);  /* returned new chunk handle */
```

This routine loads a specified token's moniker, allocating a new chunk in a local memory block for the moniker. The returned error flag will be *true* if the moniker was not found, *false* otherwise.

Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

displayType
A value of **DisplayType** indicating the size of the display (used to indicate small-screen devices, primarily).

searchFlags
A record of **VisMonikerSearchFlags** indicating what type of moniker is being requested.

lmemBlock The MemHandle of the local memory block in which the new chunk will be allocated. If the block is locked, you must dereference the global handle after calling this routine.

chunkSize A pointer to a word in which the size of the allocated chunk will be returned.

chunkhandle
A pointer to a chunk handle in which the handle of the newly allocated chunk will be returned.

Warnings: This routine can move chunks in the passed block, thereby invalidating pointers to any chunk in the block.

Include: token.h

■ TokenLoadTokenBlock()

```
Boolean  TokenLoadTokenBlock(
         dword              tokenChars,      /* four characters of token */
         ManufacturerID     manufacturerID,  /* manufacturer ID of token */
         word               * blockSize,     /* returned size of new block */
         MemHandle          * blockHandle);  /* returned handle of block */
```

This routine loads the specified token's **TokenEntry** structure into a newly-allocated global memory block. If the token is not found, the returned error flag will be *true*; otherwise, it will be *false*.

Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

blockSize A pointer to a word in which the size of the newly-allocated block will be returned.

blockHandle
A pointer to a global handle in which the handle of the newly-allocated block will be returned.

Include: token.h

■ TokenLoadTokenBuffer()

```
Boolean  TokenLoadTokenBuffer(
         dword               tokenChars,      /* four characters of token */
         ManufacturerID      manufacturerID,  /* manufacturer ID of token */
         TokenEntry          * buffer);       /* buffer for returned token */
```

This routine loads the specified token's **TokenEntry** structure into a passed buffer. The returned error flag will be *true* if the token was not found, *false* otherwise. Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

buffer A pointer to a locked or fixed buffer into which the token entry will be copied.

Include: token.h

■ TokenLoadTokenChunk()

```
Boolean  TokenLoadTokenChunk(
         dword               tokenChars,      /* four characters of token */
         ManufacturerID      manufacturerID,  /* manufacturer ID of token */
         MemHandle           lmemBlock,       /* handle of block for chunk */
         word                * chunkSize,     /* returned size of new chunk */
         ChunkHandle         * chunkHandle);  /* returned chunk handle */
```

This routine loads the specified token's **TokenEntry** structure into a newly-allocated chunk. The returned error flag will be *true* if the token could not be found, *false* otherwise.

Pass this routine the following:

tokenChars The four token characters that identify the token database entry. Create this dword from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

lmemBlock The MemHandle of the local memory block in which the new chunk will be allocated. If the block is locked, you must manually dereference this handle after the routine call.

chunksize A pointer to a word in which the size of the newly-allocated chunk will be returned.

chunkHandle
A pointer to a chunk handle in which the handle of the newly-allocated chunk will be returned.

Warnings: This routine can move chunks in the passed block, thereby invalidating pointers to any chunk in the block.

Include: token.h

■ TokenLockTokenMoniker()

```
void     * TokenLockTokenMoniker(
         TokenMonikerInfo    tokenMonikerInfo);/* The DB group and item numbers
                                          * as returned by TokenLookupMoniker() */
```

This routine locks a token's moniker so it may be drawn; it returns a pointer to the locked chunk containing the moniker information. Pass it the structure returned by **TokenLookupMoniker()**.

Be Sure To: Unlock the moniker with **TokenUnlockMoniker()** after you have finished drawing it.

Include: token.h

■ TokenLookupMoniker()

```
Boolean  TokenLookupMoniker(
         dword               tokenChars,      /* four characters of token */
         ManufacturerID      manufacturerID,  /* manufacturer ID of token */
         DisplayType         displayType,     /* display type of token */
         VisMonikerSearchFlags searchFlags,   /* flags for finding token */
         TokenMonikerInfo *  tokenMonikerInfo);/* DB group and item of token */
```

This routine finds and retrieves a pointer to the specific moniker for the specified token, given also the token's display type and other attributes. Pass the following:

tokenChars The four token characters that identify this moniker or moniker list in the token database. Create this dword value from the four characters with the macro TOKEN_CHARS.

manufacturerID
The manufacturer ID number of the manufacturer responsible for the token database entry.

displayType
A value of **DisplayType** indicating the size of the display (used to indicate small-screen devices, primarily).

searchFlags
A record of **VisMonikerSearchFlags** indicating what type of moniker is being requested.

tokenDBItem
A pointer to an empty **TokenMonikerInfo** structure, in which the token's group and item numbers will be returned.

The return value is an error flag: it will be *true* if the item could not be found in the token database, *false* otherwise.

Include: token.h

■ TokenCloseLocalTokenDB()

```
void      TokenCloseLocalTokenDB()
```

This routine closes the local token database.

■ TokenListTokens()

```
dword TokenListTokens(
          TokenRangeFlags     tokenRangeFlags,
          word                headerSize,
          ManufacturerID      manufacturerID)
```

This routine locates all the tokens that meet specified criteria, allocates a block, and copies the tokens to that block. The upper word of the return value is the number of matching tokens found; the lower word is the handle of the block which was allocated.

■ TokenOpenLocalTokenDB()

```
word      TokenOpenLocalTokenDB()
```

This routine opens the local token database. It returns zero on success, and a **VMStatus** error code on failure.

Include: token.h

■ TokenRemoveToken

```
Boolean  TokenRemoveToken(
         dword              tokenChars,      /* four characters of token */
         ManufacturerID     manufacturerID,  /* manufacturer ID of token */
```

This routine removes the specified token and its moniker list from the token database. It returns an error flag: if the token could not be found, the returned flag is *true*; otherwise it is *false*. Pass the following:

tokenChars	The four token characters that identify this moniker or moniker list in the token database. Create this dword value from the four characters with the macro TOKEN_CHARS.
manufacturerID	The manufacturer ID number of the manufacturer responsible for the token database entry.

Include: token.h

■ TokenUnlockTokenMoniker()

```
void     TokenUnlockTokenMoniker(
         void * moniker);
```

This routine unlocks a moniker that had been locked with **TokenLockMoniker()**. Pass a pointer to the locked moniker, as returned by the locking routine.

Include: token.h

■ TypeFromFormatID()

```
word     TypeFromFormatID(id);
         ClipboardItemFormatID id;
```

This macro extracts the word-sized format ID (of type **ClipboardItemFormat**) from a **ClipboardFormatID** argument.

Routines

■ UserAddAutoExec()

```
void      UserAddAutoExec(
          const char *        appName);
```

This routine adds an application to the list of those, like Welcome, that are automatically started by the UI when it loads. It is passed one argument:

appName	This is a pointer to a null-terminated string containing the name of the application. The application must be in SP_APPLICATION or SP_SYS_APPLICATION.

Include: ui.goh

■ UserCreateDialog()

```
optr      UserCreateDialog(
          optr   dialogBox);
```

This routine duplicates a template dialog box, attaches the dialog box to an application object, and sets it fully GS_USABLE so that it may be called with **UserDoDialog()**. Dialog boxes created in such a manner should be removed and destroyed with **UserDestroyDialog()** when no longer needed.

dialogBox	Optr to template dialog box (within a template object block). The block must be sharable, read-only and the top GenInteraction called with this routine must not be linked into any generic tree. The optr returned is a created, fully-usable dialog box.

See Also: UserDestroyDialog()

■ UserCreateInkDestinationInfo()

```
MemHandle UserCreateInkDestinationInfo(
          optr                dest,
          GStateHandle        gs,
          word                brushSize,
          GestureCallback     *callback);
```

This routine creates an **InkDestinationInfo** structure to be returned with MSG_META_QUERY_IF_PRESS_IS_INK. The callback routine must be declared _pascal.

Include: ui.goh

Structures:

```
typedef Boolean _pascal GestureCallback (
      Point *arrayOfInkPoints,
      word numPoints,
      word numStrokes);
```

■ UserDestroyDialog()

```
void      UserDestroyDialog(
          optr   dialogBox);
```

This routine destroys the passed dialog box, usually created with **UserCreateDialog()**. This routine may only be used to destroy dialog boxes occupying a single block; the block must also hold nothing other than the dialog box to be destroyed. It is for this reason that it is wise to only use this routine to destroy dialogs created with **UserCreateDialog()**.

See Also: UserCreateDialog()

■ UserDoDialog()

```
InteractionCommand UserDoDialog(
          optr   dialogBox);
```

UserDoDialog() brings a pre-instantiated dialog box on-screen, blocking the calling thread until the user responds to the dialog. You must pass the optr of a GIV_DIALOG Interaction that is set both GIA_INITIATED_VIA_USER_DO_DIALOG and GIA_MODAL.

This routine returns the **InteractionCommand** of the particular response trigger selected by the user. This **InteractionCommand** may be either a predefined type (such as IC_YES) or a custom one defined using IC_CUSTOM_START.

The pre-defined **InteractionCommand**s are:

```
IC_NULL
IC_DISMISS
IC_APPLY
IC_RESET
IC_OK
IC_YES
IC_NO
IC_STOP
IC_EXIT
IC_HELP
IC_INTERACTION_COMPLETE
```

This routine may return IC_NULL for those cases in which a system shutdown causes the dialog to be dismissed before the user has entered a response.

Warnings: This routine blocks the calling thread until the dialog box receives a MSG_GEN_GUP_INTERACTION_COMMAND. Since the application thread is blocked, it cannot be responsible for sending this message or for handling messages from the response triggers.

See Also: **UserStandardDialog()**, **UserStandardDialogOptr()**

■ UserGetInterfaceLevel()

```
UIInterfaceLevel UserGetInterfaceLevel(void)
```

This routine returns the current **UIInterfaceLevel**. This is a word-sized enumerated type. It has the following values:

```
UIIL_NOVICE
UIIL_BEGINNING_INTERMEDIATE
UIIL_ADVANCED_INTERMEDIATE
UIIL_ADVANCED
UIIL_GURU
```

Include: ui.goh

■ UserLoadApplication

```
extern GeodeHandle UserLoadApplication(
          AppLaunchFlags      alf,
          Message             attachMethod,
          MemHandle           appLaunchBlock,
          char                *filename,
          StandardPath        sPath,
          GeodeLoadError      *err);
```

Loads an application. Changes to standard application directory before attempting GeodeLoad on filename passed. Stores the filename being launched into the AppLaunchBlock, so that information needed to restore this application instance will be around later if needed.

■ UserRemoveAutoExec()

```
void      UserRemoveAutoExec(
          const char *        appName);
```

This routine removes an application from the list of those to be launched on start-up. It is passed one argument:

appName	This is a pointer to a null-terminated string containing the name of the application.

Include: ui.goh

■ UserStandardDialog()

```
word     UserStandardDialog(
         char *           helpContext,
         char *           customTriggers,
         char *           arg2,
         char *           arg1,
         char *           string,
         CustomDialogBoxFlagsdialogFlags);
```

UserStandardDialog() creates and displays either a custom dialog box or one of several pre-defined standard dialog boxes.

Most often, you will use this routine to create a custom dialog box that conforms to a standardized dialog. In this case, pass the **CustomDialogType** of SDBT_CUSTOM as the routine's first argument. You must then supply other parameters to create the custom dialog box.

If instead you wish to use one of the pre-defined **CustomDialogType** types, you should pass that type as the first argument to this routine. Some of these standard types require you to pass string parameters. Other arguments should be passed as null.

For custom dialog boxes you must pass a **CustomDialogType** (CDT_WARNING, CDT_NOTIFICATION, CDT_QUESTION, or CDT_ERROR). This chooses the proper icon glyph to display within the dialog box. (For example, a CDT_WARNING dialog might contain a large exclamation-point glyph.) Make sure that you use CDBF_DIALOG_TYPE_OFFSET to pass this value.

You should also pass a valid **GenInteractionType**. In most cases, this will be either GIT_NOTIFICATION, GIT_AFFIRMATION, or GIT_MULTIPLE_RESPONSE. Make sure that you use CDBF_INTERACTION_TYPE_OFFSET to pass this value.

Also pass the routine a string to display to the user. This string may be either text or graphics based.

If the **CustomDialogType** is GIT_MULTIPLE_RESPONSE, you must also set up a Response Trigger Table with several trigger parameters.

■ UserStandardDialogOptr()

```
word      UserStandardDialogOptr(
          char                *helpContext,
          optr                customTriggers,
          optr                arg2,
          optr                arg1,
          optr                string
          CustomDialogBoxFlagsdialogFlags);
```

UserStandardDialogOptr() performs the same functionality as **UserStandardDialog()** except that optrs to strings and string parameters are passed instead of fptrs. This is useful for localized strings in resource blocks.

See Also: **UserStandardDialog()**, **UserDoDialog()**

■ UserStandardSound()

```
void      UserStandardSound(
          StandardSoundType   type,
          ...);
```

This routine plays a simple sequence of notes. It can be used to play a standard system sound, a single custom tone, or a sequence of tones.

The routine takes a variable number of arguments. The first argument is a member of the **StandardSoundType** enumerated type. This argument specifies what kind of tone or tones will be played. Depending on the **StandardSoundType** passed, zero, one, or two additional arguments may be needed. **StandardSoundType** contains the following members:

SST_ERROR	This is the sound played when an “Error” dialog comes up. No further arguments are needed.
SST_WARNING	This is a general warning sound. No further arguments are needed.
SST_NOTIFY	This is a general notification sound. No further arguments are needed.
SST_NO_INPUT	This is the sound played when a user’s input is not going anywhere (e.g. when he clicks the mouse outside a modal dialog box).
SST_KEY_CLICK	This is the sound produced when the keyboard is pressed, or

when the user clicks on a floating keyboard. No further arguments are required.

SST_CUSTOM_SOUND
: Play a custom sampled sound. This requires one more argument, the memory handle of the sound to be played.

SST_CUSTOM_BUFFER
: Play a custom buffer of instrumental sound. This requires one further argument, a pointer to the memory block containing the sound buffer. Note that the “tempo” value used to play this buffer will be one tick per thirty-second note, probably much faster than you would otherwise expect.

SST_CUSTOM_NOTE
: By passing this argument, you can have a single custom note played. You must provide one further argument, the handle of the note (such as returned by **SoundAllocNote()**).

■ UtilAsciiToHex32()

```
Boolean  UtilAsciiToHex32(
         const char *        string,
         dword *             value);
```

This routine converts a null-terminated ASCII string into a 32-bit integer. The string may begin with a hyphen, indicating a negative number. Aside from that, the string may contain nothing but numerals until the null termination. It may not contain whitespace.

If the routine is successful, it will return *false* and write an equivalent signed long integer to **value*. If it fails, it will return *true* and write a member of the **UtilAsciiToHexError** enumerated type to **value*. This type contains the following members:

UATH_NON_NUMERIC_DIGIT_IN_STRING
: This string contained a non-numeric character before the trailing null (other than the allowed leading hyphen).

UATH_CONVERT_OVERFLOW
: The string specified a number to large to be expressed as a signed 32-bit integer.

Include: system.h

■ UtilHex32ToAscii()

```
word      UtilHex32ToAscii(
          char *               buffer,
          sdword               value,
          UtilHexToAsciiFlags  flags);
```

This routine converts a 32-bit unsigned integer to its ASCII representation and writes it to the specified buffer. It returns the length of the string (not counting the nulll termination, if any). The routine is passed the following arguments:

buffer This is a pointer to a character buffer. The buffer must be long enough to accommodate the largest string; that is, there must be ten bytes for the characters, plus one for the trailing null (if necessary).

value This is the value to convert to ASCII.

flags This is a record of **UtilHexToAscii** flags. The following flags are available:

UHTAF_INCLUDE_LEADING_ZEROS
Pad the string with leading zeros to a length of ten total characters.

UHTAF_NULL_TERMINATE
Add a null to the end of the string. If this flag is set, the buffer must be at least 11 bytes long. If it is clear, the buffer may be ten bytes long.

Include: system.h

■ VarDataFlagsPtr()

```
VarDataFlags VarDataFlagsPtr(
          void * ptr);
```

This macro fetches the flags of a variable data type when given a pointer to the extra data for the type. The flags are stored in a **VarDataFlags** record. Only the flags VDF_EXTRA_DATA and/or VDF_SAVE_TO_STATE will be returned.

Include: object.h

Warnings: You must pass a pointer to the *beginning* of the vardata entry's extra data space.

■ VarDataSizePtr()

```
word      VarDataSizePtr(
          void * ptr);
```

This macro fetches the size of a variable data entry when given a pointer to the extra data for the type.

Include: object.h

Warnings: You must pass a pointer to the *beginning* of the vardata entry's extra data space.

■ VarDataTypePtr()

```
word      VarDataTypePtr(
          void * ptr);
```

This macro fetches the type of a variable data entry when given a pointer to the extra data of the entry. The type is stored in a **VarDataFlags** record. All flags outside the VDF_TYPE section will be cleared.

Include: object.h

Warnings: You must pass a pointer to the *beginning* of the vardata entry's extra data space.

■ VisObjectHandlesInkReply()

```
void      VisObjectHandlesInkReply(void);
```

■ VisTextGraphicCompressGraphic()

```
extern VMChain VisTextGraphicCompressGraphic(
          VisTextGraphic      *graphic,
          FileHandle          sourceFile,
          FileHandle          destFile,
          BMFormat format,
          word xRes,
          word yRes);
```

This routine compresses the bitmaps in a VisTextGraphic.

■ VMAlloc()

```
VMBlockHandle VMAlloc(
          VMFileHandle        file,
          word                size,     /* Size of a file in bytes */
          word                userID);  /* ID # to associate with block */
```

This routine creates a VM block. The block is not initialized. Before you use the block, you must lock it with **VMLock()**. If you pass a size of zero bytes, the VM block will be given an entry in the VM handle table, but no space in memory or in the file will be used; a global memory block will have to be assigned with **VMAttach()**.

Include: vm.h

See Also: VMAllocLMem(), VMAttach()

■ VMAllocLMem()

```
VMBlockHandle VMAllocLmem(
          VMFileHandle        file,
          LMemType            ltype,       /* Type of LMem heap to create */
          word                headerSize); /* Size to leave for LMem header...
                                         * pass zero for standard header */
```

This routine allocates a VM block and initializes it to contain an LMem heap. You must pass the type of LMem heap to create. If you want a fixed data space, you must pass the total size to leave for a header (including the **LMemBlockHeader**); otherwise, pass a zero header size, indicating that only enough space for an **LMemBlockHeader** should be left. You do not need to specify a block size, since the heap will automatically expand to accommodate chunk allocations.

The block's user ID number is undefined. You will need to lock the block with **VMLock()** before accessing the chunks.

Include: vm.h

Be Sure To: When you access chunks, remember to pass the block's *global memory* handle to the LMem routines (not the block's VM handle).

See Also: LMemInitHeap(), VMAlloc(), VMAttach()

■ VMAttach()

```
VMBlockHandle VMAttach(
          VMFileHandle        file,
          VMBlockHandle       vmBlock,
          MemHandle           mh);
```

This routine attaches an existing global memory block to a VM block. It is passed the following arguments:

file	The file's **VMFileHandle**.
vmBlock	The handle of the VM block to which the memory block should be attached. Any data associated with that block will be lost. If you pass a null **VMBlockHandle**, a new VM block will be allocated.
mh	The handle of the global memory block to attach.

The routine returns the handle of the VM block to which the memory block was attached.

If you attach to a pre-existing VM block, its user ID will be preserved. If you create a new block (by passing a null *vmBlock* argument), the user ID will be undefined.

Include: vm.h

■ VMCheckForModifications()

```
Boolean   VMCheckForModifications(
          VMFileHandle        file);
```

This routine returns *true* if the VM file has been dirtied or updated since the last full save.

Include: vm.h

■ VMClose()

```
word      VMClose(
          VMFileHandle        file,
          Boolean             noErrorFlag);
```

This routine updates and closes a VM file. If it is successful, it returns *false*. If it fails, it returns a member of the **FileError** enumerated type. Note that the routine closes the file even if it could not successfully update the file; in this case, any changes since the last update will be lost. For this reason, it is safest to call **VMUpdate()** first, then (after the file has been successfully updated) call **VMClose()**.

If *noErrorFlag* is *true*, **VMClose()** will fatal-error if it could not succesfully update and close the file.

Include: vm.h

■ VMCompareVMChains()

```
Boolean VMCompareVMChains(
        VMFileHandle        sourceFile,
        VMChain             sourceChain,
        VMFileHandle        destFile,
        VMChain             destChain);
```

This routine compares two VM chains or DB items. It returns *true* if the two are identical; otherwise it returns *false*.

Include: vm.h

■ VMCopyVMBlock()

```
VMBlockHandle VMCopyVMBlock(
        VMFileHandle        sourceFile,
        VMBlockHandle       sourceBlock,
        VMFileHandle        destFile);
```

This routine creates a duplicate of a VM block in the specified destination file (which may be the same as the source file). It returns the duplicate block's handle. The duplicate will have the same user ID as the original block.

Include: vm.h

■ VMCopyVMChain()

```
VMChain VMCopyVMChain(
        VMFileHandle        sourceFile,
        VMChain             sourceChain,
        VMFileHandle        destFile);
```

This routine creates a duplicate of a VM chain (or DB item) in the specified destination file (which may be the same as the source file). It returns the duplicate's **VMChain** structure. All blocks in the duplicate will have the same user ID numbers as the corresponding original blocks.

Include: vm.h

■ VMDetach()

```
MemHandle VMDetach(
          VMFileHandle        file,
          VMBlockHandle       block,
          GeodeHandle         owner);   /* Pass zero to have block owned by
                                                * current thread's owner */
```

This routine detaches a global memory block from a VM block. If the VM block is not currently in memory, **VMDetach()** allocates a memory block and copies the VM block into it. If the VM block is dirty, **VMDetach()** will update the block to the file before detaching it.

Include: vm.h

■ VMDirty()

```
void      VMDirty(
          MemHandle           mh);
```

This routine marks a locked VM block as dirty.

Include: vm.h

■ VMFind()

```
VMBlockHandle VMFind(
          VMFileHandle        file,
          VMBlockHandle       startBlock,
          word                userID);
```

This routine finds a VM block with the specified user ID number. If the second argument is **NullHandle** the routine will return the matching block with the lowest handle. If the second argument is non-null, it will return the first matching block whose handle is larger than the one passed (in numerical order).

Include: vm.h

■ VMFree()

```
void      VMFree(
          VMFileHandle        file,
          VMBlockHandle       block);
```

This routine frees the specified VM block. If a global memory block is currently attached to the VM block, it is freed too.

Include: vm.h

■ VMFreeVMChain()

```
void        VMFreeVMChain(
            VMFileHandle        file,
            VMChain             chain);
```

This routine frees the specified VM chain or DB item. If a chain is specified, all blocks in the chain will be freed.

Include: vm.h

■ VMGetAttributes()

```
word        VMGetAttributes(
            VMFileHandle        file);
```

Each VM file contains a set of **VMAttributes** flags. These determine how the VM manager will treat the file. This routine returns the current flags.

Include: vm.h

Tips and Tricks: When the Document Control objects create files, they automatically initialize the attributes appropriately.

See Also: VMSetAttributes()

■ VMGetDirtyState()

```
word        VMGetDirtyState(
            VMFileHandle        file);
```

This routine finds out if a file has been dirtied. It returns a word-sized value. The upper byte of the return value is non-zero if the file has not been dirtied since the last save, auto-save, or update; the lower byte is non-zero if the file has not been dirtied since the last save. Thus, if the return value is zero, the file must be updated.

Include: vm.h

Tips and Tricks: **VMUpdate()** is optimized for updating clean files. For this reason, it is faster to call **VMUpdate()** then it is to first check the dirty state, then call **VMUpdate()** only if the file is dirty.

■ VMGetMapBlock()

```
VMBlockHandle VMGetMapBlock(
            VMFIleHandle        file);
```

This routine returns the VM block handle of the file's map block.

Include: vm.h

■ VMGrabExclusive()

```
VMStartExclusiveReturnValue VMGrabExclusive(
          VMFileHandle        file,
          word                timeout,
          VMOperation         operation,
          VMOperation *       currentOperation);
```

This routine gets exclusive access to a VM file for this thread.

Include: vm.h

■ VMInfo()

```
Boolean   VMInfo(
          VMFileHandle        file,
          VMBlockHandle       block,
          VMInfoStruct *      info
```

This routine writes the memory handle, block size, and user ID number of the block. It returns *false* if the handle is invalid or free.

Include: vm.h

■ VMLock()

```
void *    VMLock(
          VMFileHandle        file,
          VMBlockHandle       block,
          MemHandle*          mh);
```

This routine locks a VM block into the global heap. It returns the block's base address.

Include: vm.h

■ VMMemBlockToVMBlock()

```
VMBlockHandle VMMemBlockToVMBlock(
          MemHandle           mh,
          VMFileHandle*       file);
```

This routine gets the VM block and file handles for a specified memory block. It returns the VM block handle and copies the VM file handle into **file*.

The memory handle passed must be the handle of a block which is attached to a VM file. If it is not, the results are undefined.

Include: vm.h

■ VMModifyUserID()

```
void      VMModifyUserID(
          VMFileHandle        file,
          VMBlockHandle       block,
          word                userID);
```

This routine changes a VM block's user ID number.

Include: vm.h

■ VMOpen()

```
VMFileHandle VMOpen(
          char *              name,         /* Name of file to open/create */
          VMAccessFlags       flags,
          VMOpenType          mode,
          word                compression); /* Compression threshold percentage
                                          * passed as an integer */
```

This routine opens or creates a VM file. It returns the handle of the opened file. If it is unable to open the file, it sets the error value for **ThreadGetError()**. **VMOpen()** looks for the file in the thread's working directory (unless a temporary file is being created, as described below). The routine takes four arguments:

name A pointer to a string containing the name of the file to open. The file will be opened in the thread's current working directory. If a temporary file is being opened, this buffer should contain the full path of the directory in which to create the file, followed by fourteen null bytes (counting the string-ending null). **VMOpen()** will write the name of the temporary file in those trailing nulls.

flags This specifies what kind of access to the file you need. The flags are described below.

mode This specifies how the file should be opened. The types are described below.

compression The compression threshold percentage, passed as an integer. For example, to set a compression threshold of 50%, pass the integer '50'. When the percentage of used space in the file drops below the compression threshold, the VM manager will automatically compress the file. To use the system default threshold, pass a threshold of zero. The compression threshold is set only when the file is created; this argument is ignored if an existing file is opened.

The **VMAccessFlags** specify what kind of access to the file the caller wants. The following flags are available:

VMAF_FORCE_READ_ONLY
: If set, the file will be opened read-only, even if the default would be to open the file read/write. Blocks in read-only files cannot be dirtied, and changes in memory blocks will not be updated to the disk VM blocks.

VMAF_FORCE_READ_WRITE
: If set, the file will be opened for read/write access, even if the default would be to open the file for read-only access.

VMAF_SHARED_MEMORY
: If set, the VM manager should try to use shared memory when locking VM blocks; that is, the same memory block will be used for a given VM block no matter which thread locks the block.

VMAF_FORCE_DENY_WRITE
: If set, then open the file deny-write; that is, no other threads will be allowed to open the file for read/write access.

VMAF_DISALLOW_SHARED_MULTIPLE
: If this flag is set, files with the file attribute GFHF_SHARED_MULTIPLE cannot be opened.

VMAF_USE_BLOCK_LEVEL_SYNCHRONIZATION
: If set, the block-level synchronization mechanism of the VM manager is assumed to be sufficient; the more restrictive StartExclusive/EndExclusive mechanism is not used. This is primarily intended for system software.

You must also specify how the file should be opened. To do this, you pass a member of the **VMOpenType** enumerated type. The following types are available:

VMO_TEMP_FILE
: If this is passed, the file will be a temporary data file. When you create a temporary file, you pass a directory path, not a file name. The path should be followed by fourteen null bytes, including the string's terminating null. The system will choose an appropriate file name and add it to the path string.

VMO_CREATE_ONLY
: If this is passed, the document will be created. If a document with the specified name already exists in the working directory, **VMOpen()** will return an error condition.

Routines

VMO_CREATE
: If this is passed, the file will be created if it does not already exist; otherwise it will be opened.

VMO_CREATE_TRUNCATE
: If this is passed, the file will be created if it does not already exist; otherwise, it will be opened and truncated (all data blocks will be freed).

VMO_OPEN
: Open existing file. If file does not exist, return an error condition.

If for any reason **VMOpen()** is unable to open the requested file, it will returns a null file handle. It will also set the error value for **ThreadGetError()**. The possible error conditions are:

VM_FILE_EXISTS
: **VMOpen()** was passed VMO_CREATE_ONLY, but the file already exists.

VM_FILE_NOT_FOUND
: **VMOpen()** was passed VMO_OPEN, but the file does not exist.

VM_SHARING_DENIED
: The file was opened by another geode, and access was denied.

VM_OPEN_INVALID_VM_FILE
: **VMOpen()** was instructed to open an invalid VM file (or a non-VM file).

VM_CANNOT_CREATE
: **VMOpen()** cannot create the file (but it does not already exist).

VM_TRUNCATE_FAILED
: **VMOpen()** was passed VMO_CREATE_TRUNCATE; the file exists, but could not be truncated.

VM_WRITE_PROTECTED
: **VMOpen()** was passed VMAF_FORCE_READ_WRITE, but the file was write-protected.

Include: vm.h

Tips and Tricks: If you use the document control objects, they will take care of opening files as necessary; you will not need to call **VMOpen()**.

See Also: FileOpen()

■ VMPreserveBlocksHandle()

```
void      VMPreserveBlocksHandle(
          VMFileHandle       file,
          VMBlockHandle      block);
```

Keep the same global memory block with this VM block until the block is explicitly detached or the VM block is freed.

Include: vm.h

■ VMReleaseExclusive()

```
void VMReleaseExclusive(
          VMFileHandle       file);
```

This routine releases a thread's exclusive access to a VM file.

Include: vm.h

■ VMRevert()

```
void      VMRevert(
          VMFileHandle       file,);
```

This routine reverts a file to its last-saved state.

Include: vm.h

■ VMSave()

```
void      VMSave(
          VMFileHandle       file);
```

This routine updates and saves a file, freeing all backup blocks.

Include: vm.h

■ VMSaveAs()

```
VMFileHandle VMSaveAs(
          VMFileHandle       file,
          const char         *name,
          VMAccessFlags      flags.
          VMOpenTypes        mode,
          word               compression);
```

This routine saves a file under a new name. The old file is reverted to its last-saved condition.

Include: vm.h

Routines

■ VMSetAttributes()

```
word      VMSetAttributes(
          VMFileHandle        file,
          VMAttributes        attrToSet,    /* Turn these flags on... */
          VMAttributes        attrToClear); /* after turning these flags off */
```

This routine changes a VM file's **VMAttributes** settings. The routine returns the new attribute settings.

Include: vm.h

Tips and Tricks: When the Document Control objects create files, they automatically initialize the attributes appropriately.

Warnings: If you turn off VMA_BACKUP, make sure you do it right after a save or revert (when there are no backup blocks).

See Also: VMGetAttributes()

■ VMSetExecThread()

```
void      VMSetExecThread(
          VMFileHandle        file,
          ThreadHandle        thread);
```

Set which thread will execute methods of all objects in the file.

Include: vm.h

■ VMSetMapBlock()

```
void      VMSetMapBlock(
          VMFileHandle        file,
          VMBlockHandle       block);
```

This routine sets the map block for a VM file.

Include: vm.h

■ VMSetReloc()

```
void      VMSetReloc(
          VMFileHandle        file,
          void (*reloc)       (VMFileHandle    file,
                               VMBlockHandle   block,
                               MemHandle       mh,
                               void            *data,
                               VMRelocTypes    type));
```

This routine sets a data-relocation routine for the VM file.

Include: vm.h

■ VMUnlock()

```
void    VMUnlock(
        MemHandle          mh);
```

This routine unlocks a locked VM block. Note that the block's *global memory handle* is passed (not its VM handle).

Include: vm.h

■ VMUpdate()

```
word    VMUpdate(
        VMFileHandle       file);
```

This routine updates dirty blocks to the disk.

Include: vm.h

Tips and Tricks: **VMUpdate()** is optimized for updating clean files to the disk. Therefore, it is faster to call **VMUpdate()** whenever you think it might be necessary, than it is to check the dirty state and then call **VMUpdate()** only if the file is actually dirty.

■ VMVMBlockToMemBlock()

```
MemHandle VMVMBlockToMemBlock(
        VMFileHandle       file,
        VmBlockHandle      block);
```

This routine returns the global handle of the memory block attached to a specified VM block. If no global block is currently attached, it will allocate and attach one.

Include: vm.h

■ WinAckUpdate()

```
void    WinAckUpdate(
        WindowHandle       win);
```

This routine acknowledges that the application has received MSG_META_EXPOSED for the specified window, but chooses not to do any updating.

Include: win.h

■ WinApplyRotation()

```
void      WinApplyRotation(
          WindowHandle        win,
          WWFixedAsDWord      angle,
          WinInvalFlag        flag);
```

This routine applies the specified rotation to the window's transformation matrix.

Include: win.h

■ WinApplyScale()

```
void      WinApplyScale(
          WindowHandle        win,
          WWFixedAsDWord      xScale,
          WWFixedAsDWord      yScale,
          WinInvalFlag        flag);
```

This routine applies the specified scale factor to the window's transformation matrix.

Include: win.h

■ WinApplyTranform()

```
void      WinApplyTransform(
          WindowHandle        win,
          const TransMatrix * tm,
          WinInvalFlag        flag);
```

This routine concatenates the passed transformation matrix with the window's transformation matrix. The result will be the window's new transformation matrix.

Include: win.h

■ WinApplyTranslation()

```
void      WinApplyTranslation(
          WindowHandle        win,
          WWFixedAsDWord      xTrans,
          WWFixedAsDword      yTrans,
          WinInvalFlag        flag);
```

This routine applies the specified translation to the window's transformation matrix.

Include: win.h

■ WinApplyTranslationDWord()

```
void      WinApplyExtTranslation(
          WindowHandle        win,
          sdword              xTrans,
          sdword              yTrans,
          WinInvalFlag        flag);
```

This routine applies the specified translation to the window's transformation matrix. The translations are specified as 32-bit integers.

Include: win.h

■ WinChangeAck()

```
WindowHandle WinChangeAck(
          WindowHandle        win,
          sword               x,
          sword               y,
          optr *              winOD);
```

Include: win.h

■ WinChangePriority()

```
void      WinChangePriority(
          WindowHandle        win,
          WinPassFlags        flags,
          word                layerID);
```

This routine changes the priority for the specified window.

Include: win.h

■ WinClose()

```
void      WinClose(
          WindowHandle        win);
```

This routine closes and frees the specified window.

Include: win.h

■ WinDecRefCount()

```
void      WinDecRefCount(
          WindowHandle        win);
```

This routine is part of the window closing mechanism.

■ WinEnsureChangeNotification()

```
void      WinEnsureChangeNotification(void);
```

Include: win.h

■ WinGeodeGetInputObj()

```
optr      WinGeodeGetInputObj(
          GeodeHandle         obj);
```

This routine fetches the optr of the input object for the specified geode. If there is no such object, it returns a null optr.

Include: win.h

■ WinGeodeGetParentObj()

```
optr      WinGeodeGetParentObj(
          GeodeHandle         obj);
```

This routine fetches the optr of the parent object of the specified geode. If there is no such object, it returns a null optr.

Include: win.h

■ WinGeodeSetActiveWin()

```
void      WinGeodeSetActiveWin(
          GeodeHandle         gh,
          WindowHandle        win);
```

This routine sets the active window for the specified geode.

Include: win.h

■ WinGeodeSetInputObj()

```
void      WinGeodeSetInputObj(
          GeodeHandle         gh,
          optr                iObj);
```

This routine sets the input object for the specified geode.

Include: win.h

■ WinGeodeSetParentObj()

```
void      WinGeodeSetParentObj(
          GeodeHandle         gh,
          optr                pObj);
```

This routine sets the parent object for the specified geode.

Include: win.h

■ WinGeodeSetPtrImage()

```
void      WinGeodeSetPtrImage(
          GeodeHandle         gh,
          optr                ptrCh);
```

This routine sets the pointer image for the specified geode.

Include: win.h

■ WinGetInfo()

```
dword     WinGetInfo(
          WindowHandle        win,
          WinInfoTypes        type,
          void *              data);
```

This routine retrieves the private data from a GState.

Include: win.h

■ WinGetTransform()

```
void      WinGetTransform(
          WindowHandle        win,
          TransMatrix *       tm);
```

This routine retrieves the transformation matrix for the specified window. It writes the matrix to **tm*.

Include: win.h

■ WinGetWinScreenBounds()

```
void      WinGetWinScreenBounds(
          WindowHandle        win,
          Rectangle *         bounds);
```

This routine returns the bounds of the on-screen portion of a window (specified in screen co-ordinates). It writes the bounds to **bounds*.

Include: win.h

■ WinGrabChange()

```
Boolean  WinGrabChange(
         WindowHandle        win,
         optr                newObj);
```

This routine allows an object to grab pointer events. It returns zero if it was successful; otherwise it returns non-zero.

Include: win.h

■ WinInvalReg()

```
void     WinInvalReg(
         WindowHandle        win,
         const Region *      reg,
         word                axParam,
         word                bxParam,
         word                cxParam,
         word                dxParam);
```

This routine invalidates the specified region or rectangle.

Include: win.h

■ WinMove()

```
void     WinMove(
         WindowHandle        win,
         sword               xMove,
         sword               yMove,
         WinPassFlags        flags);
```

This routine moves a window. If the WPF_ABS bit of *flags* is set, the window's new position is specified relative to its parent's position. If it is clear, the window's new position is specified relative to its current position.

Include: win.h

■ WinOpen()

```
WindowHandle WinOpen(
          Handle              parentWinOrVidDr,
          optr                inputRecipient,
          optr                exposureRecipient,
          WinColorFlags       colorFlags,
          word                redOrIndex,
          word                green,
          word                blue,
          word                flags,
          word                layerID,
          GeodeHandle         owner,
          const Region *      winReg,
          word                axParam,
          word                bxParam,
          word                cxParam,
          word                dxParam);
```

This routine allocates and initializes a window and (optionally) an associated GState.

Include: win.h

■ WinReleaseChange()

```
void      WinReleaseChange(
          WindowHandle        win,
          optr                obj);
```

This routine releases an object’s grab on the change OD.

Include: win.h

■ WinResize()

```
void      WinResize(
          WindowHandle        win,
          const Region *      reg,
          word                axParam,
          word                bxParam,
          word                cxParam,
          WinPassFlags        flags);
```

This routine resizes a window. It can move it as well.

Include: win.h

■ WinScroll()

```
void      WinScroll(
          WindowHandle        win,
          WWFixedAsDWord      xMove,
          WWFixedAsSWord      yMove,
          PointWWFixed *      scrollAmt);
```

This routine scrolls a window.

Include: win.h

■ WinSetInfo()

```
void      WinSetInfo(
          WindowHandle        win,
          WinInfoType         type,
          dword               data);
```

This routine sets some data for the specified window.

Include: win.h

■ WinSetNullTransform()

```
void      WinSetNullTransform(
          WindowHandle        win,
          WinInvalFlag        flag);
```

This routine changes a window's transformation matrix to the null (or identity) matrix.

Include: win.h

■ WinSetPtrImage()

```
void      WinSetPtrImage(
          WindowHandle        win,
          WinSetPtrImageLevel ptrLevel,
          optr                ptrCh);
```

This routine sets the pointer image within the range handled by the specified window.

Include: win.h

■ WinSetTransform()

```
void      WinSetTransform(
          WindowHandle         win,
          const TransMatrix *  tm,
          WinInvalFlag         flag);
```

This routine replaces the window's transformation matrix with the one passed in **tm*.

Include: win.h

■ WinSuspendUpdate()

```
void      WinSuspendUpdate(
          WindowHandle         win);
```

This routine suspends the sending of update messages to the window. The messages will be sent when **WinUnSuspendUpdate()** is called.

Include: win.h

■ WinTransform()

```
XYValueAsDWord WinTransform(
          WindowHandle         win,
          sword                x,
          sword                y);
```

This routine translates the passed document coordinates into screen coordinates.

Include: win.h

■ WinTransformDWord()

```
void      WinTransformDWord(
          WindowHandle         win,
          sdword               xCoord,
          sdword               yCoord,
          PointDWord *         screenCoordinates);
```

This routine translates the passed document coordinates into screen coordinates. The translated coordinates are written to **screenCoordinates*.

Include: win.h

■ WinUnSuspendUpdate()

```
void      WinUnSuspendUpdate(
          WindowHandle        win);
```

This routine cancels a previous **WinSuspendUpdate()** call.

Include: win.h

■ WinUntransform

```
XYValueAsDWord WinUntransform(
          WindowHandle        win,
          sword               x,
          sword               y);
```

This routine translates the passed screen coordinates into document coordinates.

Include: win.h

■ WinUnTransformDWord()

```
void      WinTransformDWord(
          WindowHandle        win,
          sdword              xCoord,
          sdword              yCoord,
          PointDWord *        documentCoordinates);
```

This routine translates the passed screen coordinates into document coordinates. The translated coordinates are written to **documentCoordinates*.

Include: win.h

■ WWFixedToFrac

```
word      WWFixedToFrac(WWFixed wwf)
```

This macro lets you address the fractional portion of a **WWFixed** value. It is legal to use this to assign a value to the fractional protion; that is,

```
WWFixedToFrac(myWWFixed) = 5;
```

is perfectly legal.

Include: geos.h

■ WWFixedToInt

```
word     WWFixedToInt(WWFixed wwf)
```

This macro lets you address the intetgral portion of a **WWFixed** value. It is legal to use this to assign a value to the integral protion; that is,

```
WWFixedToInt(myWWFixed) = 5;
```

is perfectly legal.

Include: geos.h

Data Structures

4

Global data structures and types are listed alphabetically on the following pages. Some data structures used by only a few routines or by only one or two classes are documented with those routines or classes.

■ AddUndoActionFlags

```
typedef WordFlags AddUndoActionFlags;
    #define AUAF_NOTIFY_BEFORE_FREEING                      0x8000
    #define AUAF_NOTIFY_IF_FREED_WITHOUT_BEING_PLAYED_BACK 0x4000
```

■ AddUndoActionStruct

```
typedef struct {
    UndoActionStruct         AUAS_data;
    optr                     AUAS_output;
    AddUndoActionFlags       AUAS_flags;
} AddUndoActionStruct;
```

The "undo" structures work together to provide information vital to processes which will be working with undo events.

■ AppAttachFlags

```
typedef WordFlags AppAttachFlags;
    #define AAF_RESTORING_FROM_STATE 0x8000
    #define AAF_STATE_FILE_PASSED    0x4000
    #define AAF_DATA_FILE_PASSED     0x2000
```

These flags are passed to the process when the application is launching or being restored from a state file. The flags indicate whether the application is being launched from a state file, has a state file, and/or has a data file.

Note that if AAF_RESTORING_FROM_STATE is set, then AAF_STATE_FILE_PASSED will also be set.

■ AppInstanceReference

```
typedef struct {
    /* AIR_fileName:
     * Application being launched. Pathname is relative to application
     * directory (which, of course, may be overriden with a direct path
     * to the application). */
    PathName                 AIR_fileName;
    /* AIR_stateFile:
     * State filename. File is assumed to be in standard directory for
     * GEOS state files. If the first byte is "0", then there is no
     * state file for this application. This structure is copied into the
     * field as an aid in restarting applications, and if it comes across
     * one with this byte as 0, it will not restart it. */
    FileLongName             AIR_stateFile;
    /* AIR_diskHandle:
     * Disk handle for app (passed in) IF 0, use System disk, if -1, use
     * AIR_diskName. In the field, if this is a placeholder structure, this
     * word is the handle of the application object we are waiting to detach. */
    DiskHandle                AIR_diskHandle;
    /* AIR_savedDiskData:
     * Start of data stored by DiskSave when instance is saved to state file. */
    byte                      AIR_savedDiskData[1];
} AppInstanceReference;
```

■ AppLaunchBlock

```
typedef struct {
    /* ALB_appRef:
     * Instance reference. Contains full pathname to application, as
     * referenced from app directory, plus the name of a state file.
     * Is enough info to launch application again, restored. (State file
     * need not be passed to GeodeLoad) */
    AppInstanceReference      ALB_appRef;
    /* ALB_appMode:
     * Application attach mode method. Should be one of the following:
     * MSG_GEN_PROCESS_RESTORE_FROM_STATE:
     *      State file must be passed; no data file should be passed.
     * MSG_GEN_PROCESS_OPEN_APPLICATION:
     *      State file normally should not be passed, although one could be to
     *      accomplish ui templates. A data file may be passed into the
     *      application as well.
     * MSG_GEN_PROCESS_OPEN_ENGINE:
     *      State file normally should not be passed. The data file on which the
     *      engine will operate must be passed. If zero, the default data file
     *      should be used (enforced by app, not GenProcessClass).*/
    Message                   ALB_appMode;
    /* ALB_launchFlags:
     * Miscellaneous flags to specify desired application launch type. */
    AppLaunchFlags            ALB_launchFlags;
    /* ALB_diskHandle:
     * Disk handle for data path. (Set as application’s current path in
     * GenProcess’ MSG_META_ATTACH handler.) */
    MemHandle                 ALB_diskHandle;
    /* ALB_path:
     * Data path for application to use as initial path. (Usually this is
     * a directory of any data file passed.) (Set as application current
     * path in GenProcess’ MSG_META_ATTACH handler.)
    char                      ALB_path[PATH_BUFFER_SIZE];
    /* ALB_dataFile:
     * Name of data file passed in to be opened (0 if none). Pathname is
     * relative to above path. */
    char                      ALB_dataFile[PATH_BUFFER_SIZE];
```

Routines

```
    /* ALB_genParent:
     * Generic parent for new application (0 to put on default field). (Should
     * be passed NULL to MSG_GEN_FIELD_LAUNCH_APPLICATION).
    optr                      ALB_genParent;
    /* ALB_userLoadAckOutput, ALB_userLoadAckMessage:
     * Together, these form an Action Descriptor which will be activated when
     * the application has been launched (used in conjunction with
     * ALF_SEND_LAUNCH_REQUEST_TO_UI_TO_HANDLE). (Set to NULL/0 if you don’t
     * want to send anything).
     * The acknowledgement will come with three arguments: the GeodeHandle
     * (non-NULL if successful), a word value which will be zero if there was
     * an error, and the word value set in ALB_userLoackAckID (below).*/
    optr                      ALB_userLoadAckOutput;
    Message                   ALB_userLoadAckMessage;
    /* ALB_userLoadAckID:
     * ID sent out via above action descriptor, if any. */
    word                      ALB_userLoadAckID;
    /* ALB_extraData:
     * Extra data to send to process (possibly a handle to
     * block containing arguments). */
    word                      ALB_extraData;
} AppLaunchBlock;
```

This structure is used when an application is first starting up. It is an argument of various messages which will be intercepted by system classes. The first fields (*ALB_appRef*, *ALB_appMode*, *ALB_launchFlags*, and *ALB_uiLevel*) are preserved in the application’s state file. The other information must be set correctly on launch.

■ AppLaunchFlags

```
typedef ByteFlags AppLaunchFlags;
    #define ALF_SEND_LAUNCH_REQUEST_TO_UI_TO_HANDLE   0x80
    #define ALF_OPEN_IN_BACK                          0x40
```

■ ApplicationStates

```
typedef ByteFlags ApplicationStates;
    #define AS_QUITTING                     0x80
    #define AS_DETACHING                    0x40
    #define AS_FOCUSABLE                    0x20
    #define AS_MODELABLE                    0x10
    #define AS_NOT_USER_INTERACTABLE        0x08
    #define AS_RECEIVED_APP_OBJECT_DETACH   0x04
    #define AS_ATTACHED_TO_STATE_FILE       0x02
    #define AS_ATTACHING                    0x01
```

■ ArcCloseType

```
typedef enum /* word */ {
    ACT_OPEN,
    ACT_CHORD,
    ACT_PIE
} ArcCloseType;
```

This structure is used when filling arcs.

■ AreaAttr

```
typedef struct {
    byte         AA_colorFlag;
    RGBValue     AA_color;
    SysDrawMask  AA_mask;
    ColorMapMode AA_mapMode;
} AreaAttr;
```

■ ArgumentStackElement

```
typedef struct {
    EvalStackArgumentType ASE_type;
    EvalStackArgumentData ASE_data;
} ArgumentStackElement;
```

■ BBFixed

```
typedef struct {
    byte BBF_frac;
    byte BBF_int;
} BBFixed;
```

This structure represents an 8.8 fixed point number.

■ BBFixedAsWord

```
typedef word BBFixedAsWord;
```

This structure represents an 8.8 fixed point number.

■ Bitmap

```
typedef struct {
    word    B_width;              /* In bitmap pixels */
    word    B_height;             /* In bitmap pixels */
    byte    B_compact;            /* A BMCompact value */
    byte    B_type;               /* A BMFormat | BMType value */
} Bitmap;
```

This data structure provides some information about a simple graphics bitmap. It normally acts as the header for a set of bitmap data.

Routines

The bitmap data itself is organized into scan lines. If the bitmap has a mask (if the BMT_MASK bit is set in the *B_type* field), the first information for the scan line will be its mask information. There will be one bit of mask information for each pixel in the scan line (i.e. a number of bits equal to the bitmap width). The actual bitmap data for the scan line starts at the next byte boundary. For each pixel there will be a number of bits of color data, said number depending on the **BMFormat** value in the *B_type* field. The data for the next scan line will begin at the next byte boundary.

Thus, a 7x7 bitmap depicting an inverse “x” might appear:

```
(Bitmap)     {7, 7, BMC_UNCOMPACTED, BMF_MONO };
(byte)[]     {0x82,       /* 10000010 */
              0x44,       /* 01000100 */
              0x28,       /* 00101000 */
              0x10,       /* 00010000 */
              0x28,       /* 00101000 */
              0x44,       /* 01000100 */
              0x82 };     /* 10000010 */
```

A 3x3 color “-” shape with a a “+” shaped mask might appear:

```
(Bitmap)     { 3, 3, BMC_UNCOMPACTED,
              (BMF_4BIT | BMT_MASK)};
(byte) []    {/* scan line 1: */
              0x40,       /* mask: 010 */
              0, 0        /* data: 000 */

              /* scan line 2: */
              0xE0,       /* mask: 111 */
              0x43, 0x20, /* data: 432 */

              /* scan line 3: */
              0x40,       /* mask: 010 */
              0, 0 };     /* data: 000 */
```

If standard BMC_PACKBITS compression is used, then the mask (if any) and color data for the bitmap is compressed using the Macintosh PackBits standard. Under this system, to uncompress the data for a scan line, follow the loop:

1 Read a byte.

2 If the byte read in step (1) is between -1 and -127, read the *next* byte and copy it into the target buffer from +2 to +128 times.

3 If the byte read in step (1) is between +1 and +127, read the next 1 to 127 bytes and copy them into the target buffer.

4 If the byte read in step (1) is -128, ignore it.

Routines

5 You’re ready to read in the next batch of data; go back to step (1).

Thus a 16x4 color “=” with a matching mask would appear:

```
(Bitmap) {15, 3, BMC_PACKBITS, BMF_4BIT | BMT_MASK } ;
(byte) []     {/* scan line 1: */
                     /* mask: 2 repetitions of 0xff */
               0xff, 0xff,
                     /* data: 16 repetitions of 0x14 */
               0xf0, 0x14,
              /* scan line 2: */
                     /* mask: 2 repetitions of 0x00 */
                     /* data: 16 repetitions of 0x00 */
                     /* total: 18 repetitions of 0x00 */
               0xee, 0x00,
              /* scan line 3: */
                     /* mask: 2 repetitions of 0x00 */
                     /* data: 16 repetitions of 0x00 */
                     /* total: 18 repetitions of 0x00 */
               0xee, 0x00,
              /* scan line 4: */
                     /* mask: 2 repetitions of 0xff */
               0xff, 0xff,
                     /* data: 16 repetitions of 0x14 */
               0xf0, 0x14};
```

See Also: **CBitmap**.

■ BitmapMode

```
typedef WordFlags BitmapMode;
    #define BM_EDIT_MASK          0x0002
    #define BM_CLUSTERED_DITHER   0x0001
```

■ BLTMode

```
typedef enum /* word */ {
    BLTM_COPY,
    BLTM_MOVE,
    BLTM_CLEAR
} BLTMode;
```

■ BMCompact

```
typedef ByteEnum ByteCompact;
    #define BMC_UNCOMPACTED   0
    #define BMC_PACKBITS      1
    #define BMC_USER_DEFINED  0x80
```

Routines

This data structure is used to specify what sort of compaction is used to store a graphics bitmap.

■ BMDestroy

```
typedef ByteEnum BMDestroy;
    #define BMD_KILL_DATA     0
    #define BMD_LEAVE_DATA    1
```

■ BMFormat

```
typedef ByteEnum BMFormat
    #define BMF_MONO 0
    #define BMF_4BIT 1
    #define BMF_8BIT 2
    #define BMF_24BIT 3
    #define BMF_4CMYK 4
```

This enumerated type determines a graphics bitmap's depth.

■ BMType

```
typedef ByteFlags BMType;
    #define BMT_PALETTE       0x40
    #define BMT_HUGE          0x20
    #define BMT_MASK          0x10
    #define BMT_COMPLEX       0x08
    #define BMT_FORMAT        0x07
```

This structure is used to store various facts about a graphics bitmap.

■ Boolean

```
typedef word Boolean;
```

Booleans represent true/false values. If the Boolean is *false*, it will evaluate to zero; otherwise, it will be non-zero.

■ Button

```
typedef ByteEnum Button;
    #define BUTTON_0          0
    #define BUTTON_1          1
    #define BUTTON_2          2
    #define BUTTON_3          3
```

■ ButtonInfo

```
typedef ByteFlags ButtonInfo;
    #define BI_PRESS          0x80
    #define BI_DOUBLE_PRESS   0x40
    #define BI_B3_DOWN        0x20
    #define BI_B2_DOWN        0x10
    #define BI_B1_DOWN        0x08
    #define BI_B0_DOWN        0x04
    #define BI_BUTTON         0x03
```

This structure contains the state of a mouse's buttons.

■ byte

```
typedef unsigned char byte;
```

■ ByteEnum

```
typedef byte ByteEnum;
```

■ ByteFlags

```
typedef byte ByteFlags;
```

■ CallbackType

```
typedef ByteEnum CallbackType;
    #define CT_FUNCTION_TO_TOKEN  0
    #define CT_NAME_TO_TOKEN      1
    #define CT_CHECK_NAME_EXISTS  2
    #define CT_CHECK_NAME_SPACE   3
    #define CT_EVAL_FUNCTION      4
    #define CT_LOCK_NAME          5
    #define CT_UNLOCK             6
    #define CT_FORMAT_FUNCTION    7
    #define CT_FORMAT_NAME        8
    #define CT_CREATE_CELL        9
    #define CT_EMPTY_CELL         10
    #define CT_NAME_TO_CELL       11
    #define CT_FUNCTION_TO_CELL   12
    #define CT_DEREF_CELL         13
    #define CT_SPECIAL_FUNCTION   14
```

■ CBitmap

```
typedef struct {
    Bitmap  CB_simple;
    word    CB_startScan;
    word    CB_numScans;
    word    CB_devInfo;
    word    CB_data;
    word    CB_palette;
    word    CB_xres;
    word    CB_yres;
} CBitmap;
```

Routines

The CBitmap structure contains the information for a "complex" bitmap. Use the CBitmap structure to hold bitmaps which need to keep track of resolution information, a palette, or a mask.

■ CellFunctionParameterFlags

```
typedef ByteFlags CellFunctionParameterFlags;
    #define CFPF_DIRTY 0x80 /* apps may read or change this. */
    #define CFPF_NO_FREE_COUNT 0x07
```

■ CellFunctionParameters

```
typedef struct {
    CellFunctionParameterFlagsCFP_flags;
    VMFileHandle              CFP_file;          /* File containing cells */
    VMBlockHandle             CFP_rowBlocks[N_ROW_BLOCKS];
} CellFunctionParameters;
```

This structure is used to pass specifics about a cell file to the cell library routines. Some of the data in the **CellFunctionParameters** structure is opaque to the application; others may be examined or changed by the application. The **CellFunctionParameters** structure contains the following fields:

CFP_flags The cell library uses this byte for miscellaneous bookkeeping. When you create the structure, initialize this field to zero. There is only one flag which you should check or change; that is the flag *CFPF_dirty*. The cell library routines set this bit whenever they change the **CellFunctionParameters** structure, thus indicating that the structure ought to be resaved. After you save it, you may clear this bit.

CFP_file This field must contain the VM file handle of the cell file. This field must be set each time you open the file.

CFP_rowBlocks
This field is an array of VM block handles, one for every existing or potential row block. The length of this array is N_ROW_BLOCKS (defined in **cell.h**). When you create a cell file, initialize all of these handles to zero; do not access or change this field thereafter.

Include: cell.h

Warnings: The cell library expects the **CellFunctionParameters** structure to remain motionless for the duration of a call. Therefore, if you allocate it as a DB item in the cell file, you must *not* have the structure be an ungrouped item.

Routines

■ CellRange

```
typedef struct {
    CellReference   CR_start;
    CellReference   CR_end;
} CellRange;
```

■ CellReference

```
typedef struct {
    CellRowColumn   CR_row;
    CellRowColumn   CR_column;
} CellReference;
```

■ CellRowColumn

```
typedef WordFlags CellRowColumn;
    #define CRC_ABSOLUTE      0x8000
    #define CRC_VALUE         0x7fff
```

■ CharacterSet

```
typedef ByteEnum CharacterSet;
    #define CS_BSW            0
    #define CS_CONTROL        0xff
    #define CS_UI_FUNCS       0xfe
    #define VC_ISANSI         CS_BSW
    #define VC_ISCTRL         CS_CONTROL
    #define VC_ISUI           CS_UI_FUNCS
```

■ CharFlags

```
typedef ByteFlags CharFlags;
    #define CF_STATE_KEY      0x80
    #define CF_EXTENDED       0x10
    #define CF_TEMP_ACCENT    0x08
    #define CF_FIRST_PRESS    0x04
    #define CF_REPEAT_PRESS   0x02
    #define CF_RELEASE        0x01
```

■ Chars

```
typedef ByteEnum Chars;
    #define C_NULL            0x0 /* NULL */
    #define C_CTRL_A          0x1 /* <ctrl>-A */
    #define C_CTRL_B          0x2 /* <ctrl>-B */
    #define C_CTRL_C          0x3 /* <ctrl>-C */
    #define C_CTRL_D          0x4 /* <ctrl>-D */
    #define C_CTRL_E          0x5 /* <ctrl>-E */
    #define C_CTRL_F          0x6 /* <ctrl>-F */
    #define C_CTRL_G          0x7 /* <ctrl>-G */
```

```
#define C_CTRL_H          0x8 /* <ctrl>-H */
#define C_TAB             0x9 /* TAB */
#define C_LINEFEED        0xa /* LINE FEED */
#define C_CTRL_K          0xb /* <ctrl>-K */
#define C_CTRL_L          0xc /* <ctrl>-L */
#define C_ENTER           0xd /* ENTER or CR */
#define C_SHIFT_OUT       0xe /* <ctrl>-N */
#define C_SHIFT_IN        0xf /* <ctrl>-O */
#define C_CTRL_P          0x10 /* <ctrl>-P */
#define C_CTRL_Q          0x11 /* <ctrl>-Q */
#define C_CTRL_R          0x12 /* <ctrl>-R */
#define C_CTRL_S          0x13 /* <ctrl>-S */
#define C_CTRL_T          0x14 /* <ctrl>-T */
#define C_CTRL_U          0x15 /* <ctrl>-U */
#define C_CTRL_V          0x16 /* <ctrl>-V */
#define C_CTRL_W          0x17 /* <ctrl>-W */
#define C_CTRL_X          0x18 /* <ctrl>-X */
#define C_CTRL_Y          0x19 /* <ctrl>-Y */
#define C_CTRL_Z          0x1a /* <ctrl>-Z */
#define C_ESCAPE          0x1b /* ESC */
#define C_NULL_WIDTH      0x19 /* null width character */
#define C_GRAPHIC         0x1a /* Graphic in text. */
#define C_THINSPACE       0x1b /* 1/4 width space */
#define C_ENSPACE         0x1c /* En-space, fixed width */
#define C_EMSPACE         0x1d /* Em-space, fixed width. */
#define C_NONBRKHYPHEN    0x1e /* Non breaking hyphen. */
#define C_OPTHYPHEN       0x1f /* Optional hyphen, only drawn at eol */
#define C_SPACE           ‘ ’
#define C_EXCLAMATION     ‘!’
#define C_QUOTE           ‘”’
#define C_NUMBER_SIGN     ‘#’
#define C_DOLLAR_SIGN     ‘$’
#define C_PERCENT         ‘%’
#define C_AMPERSAND       ‘&’
#define C_SNG_QUOTE       0x27
#define C_LEFT_PAREN      ‘(’
#define C_RIGHT_PAREN     ‘)’
#define C_ASTERISK        ‘*’
#define C_PLUS            ‘+’
#define C_COMMA           ‘,’
#define C_MINUS           ‘-’
#define C_PERIOD           ’.’
#define C_SLASH           ‘/’
#define C_ZERO            ‘0’
#define C_ONE             ‘1’
#define C_TWO             ‘2’
#define C_THREE           ‘3’
#define C_FOUR            ‘4’
#define C_FIVE            ‘5’
#define C_SIX             ‘6’
```

```
#define C_SEVEN           ‘7’
#define C_EIGHT           ‘8’
#define C_NINE            ‘9’
#define C_COLON           ‘:’
#define C_SEMICOLON       ‘;’
#define C_LESS_THAN       ‘<’
#define C_EQUAL           ‘=’
#define C_GREATER_THAN    ‘>’
#define C_QUESTION_MARK   ‘?’
#define C_AT_SIGN         0x40
#define C_CAP_A           ‘A’
#define C_CAP_B           ‘B’
#define C_CAP_C           ‘C’
#define C_CAP_D           ‘D’
#define C_CAP_E           ‘E’
#define C_CAP_F           ‘F’
#define C_CAP_G           ‘G’
#define C_CAP_H           ‘H’
#define C_CAP_I           ‘I’
#define C_CAP_J           ‘J’
#define C_CAP_K           ‘K’
#define C_CAP_L           ‘L’
#define C_CAP_M           ‘M’
#define C_CAP_N           ‘N’
#define C_CAP_O           ‘O’
#define C_CAP_P           ‘P’
#define C_CAP_Q           ‘Q’
#define C_CAP_R           ‘R’
#define C_CAP_S           ‘S’
#define C_CAP_T           ‘T’
#define C_CAP_U           ‘U’
#define C_CAP_V           ‘V’
#define C_CAP_W           ‘W’
#define C_CAP_X           ‘X’
#define C_CAP_Y           ‘Y’
#define C_CAP_Z           ‘Z’
#define C_LEFT_BRACKET    ‘[’
#define C_BACKSLASH       0x5c
#define C_RIGHT_BRACKET   ‘]’
#define C_ASCII_CIRCUMFLEX‘^’
#define C_UNDERSCORE      ‘_’
#define C_BACKQUOTE       ‘‘’
#define C_SMALL_A         ‘a’
#define C_SMALL_B         ‘b’
#define C_SMALL_C         ‘c’
#define C_SMALL_D         ‘d’
#define C_SMALL_E         ‘e’
#define C_SMALL_F         ‘f’
#define C_SMALL_G         ‘g’
#define C_SMALL_H         ‘h’
```

Routines

```
#define C_SMALL_I         ‘i’
#define C_SMALL_J         ‘j’
#define C_SMALL_K         ‘k’
#define C_SMALL_L         ‘l’
#define C_SMALL_M         ‘m’
#define C_SMALL_N         ‘n’
#define C_SMALL_O         ‘o’
#define C_SMALL_P         ‘p’
#define C_SMALL_Q         ‘q’
#define C_SMALL_R         ‘r’
#define C_SMALL_S         ‘s’
#define C_SMALL_T         ‘t’
#define C_SMALL_U         ‘u’
#define C_SMALL_V         ‘v’
#define C_SMALL_W         ‘w’
#define C_SMALL_X         ‘x’
#define C_SMALL_Y         ‘y’
#define C_SMALL_Z         ‘z’
#define C_LEFT_BRACE      ‘{’
#define C_VERTICAL_BAR    ‘|’
#define C_RIGHT_BRACE     ‘}’
#define C_ASCII_TILDE     ‘~’
#define C_DELETE          0x7f
#define C_UA_DIERESIS     0x80
#define C_UA_RING         0x81
#define C_UC_CEDILLA      0x82
#define C_UE_ACUTE        0x83
#define C_UN_TILDE        0x84
#define C_UO_DIERESIS     0x85
#define C_UU_DIERESIS     0x86
#define C_LA_ACUTE        0x87
#define C_LA_GRAVE        0x88
#define C_LA_CIRCUMFLEX   0x89
#define C_LA_DIERESIS     0x8a
#define C_LA_TILDE        0x8b
#define C_LA_RING         0x8c
#define C_LC_CEDILLA      0x8d
#define C_LE_ACUTE        0x8e
#define C_LE_GRAVE        0x8f
#define C_LE_CIRCUMFLEX   0x90
#define C_LE_DIERESIS     0x91
#define C_LI_ACUTE        0x92
#define C_LI_GRAVE        0x93
#define C_LI_CIRCUMFLEX   0x94
#define C_LI_DIERESIS     0x95
#define C_LN_TILDE        0x96
#define C_LO_ACUTE        0x97
#define C_LO_GRAVE        0x98
#define C_LO_CIRCUMFLEX   0x99
#define C_LO_DIERESIS     0x9a
```

```
#define C_LO_TILDE        0x9b
#define C_LU_ACUTE        0x9c
#define C_LU_GRAVE        0x9d
#define C_LU_CIRCUMFLEX   0x9e
#define C_LU_DIERESIS     0x9f
#define C_DAGGER          0xa0
#define C_DEGREE          0xa1
#define C_CENT            0xa2
#define C_STERLING        0xa3
#define C_SECTION         0xa4
#define C_BULLET          0xa5
#define C_PARAGRAPH       0xa6
#define C_GERMANDBLS      0xa7
#define C_REGISTERED      0xa8
#define C_COPYRIGHT       0xa9
#define C_TRADEMARK       0xaa
#define C_ACUTE           0xab
#define C_DIERESIS        0xac
#define C_NOTEQUAL        0xad
#define C_U_AE            0xae
#define C_UO_SLASH        0xaf
#define C_INFINITY        0xb0
#define C_PLUSMINUS       0xb1
#define C_LESSEQUAL       0xb2
#define C_GREATEREQUAL    0xb3
#define C_YEN             0xb4
#define C_L_MU            0xb5
#define C_L_DELTA         0xb6
#define C_U_SIGMA         0xb7
#define C_U_PI            0xb8
#define C_L_PI            0xb9
#define C_INTEGRAL        0xba
#define C_ORDFEMININE     0xbb
#define C_ORDMASCULINE    0xbc
#define C_U_OMEGA         0xbd
#define C_L_AE            0xbe
#define C_LO_SLASH        0xbf
#define C_QUESTIONDOWN    0xc0
#define C_EXCLAMDOWN      0xc1
#define C_LOGICAL_NOT     0xc2
#define C_ROOT            0xc3
#define C_FLORIN          0xc4
#define C_APPROX_EQUAL    0xc5
#define C_U_DELTA         0xc6
#define C_GUILLEDBLLEFT   0xc7
#define C_GUILLEDBLRIGHT  0xc8
#define C_ELLIPSIS        0xc9
#define C_NONBRKSPACE     0xca
#define C_UA_GRAVE        0xcb
#define C_UA_TILDE        0xcc
```

Routines

```
#define C_UO_TILDE        0xcd
#define C_U_OE            0xce
#define C_L_OE            0xcf
#define C_ENDASH          0xd0
#define C_EMDASH          0xd1
#define C_QUOTEDBLLEFT    0xd2
#define C_QUOTEDBLRIGHT   0xd3
#define C_QUOTESNGLEFT    0xd4
#define C_QUOTESNGRIGHT   0xd5
#define C_DIVISION        0xd6
#define C_DIAMONDBULLET   0xd7
#define C_LY_DIERESIS     0xd8
#define C_UY_DIERESIS     0xd9
#define C_FRACTION        0xda
#define C_CURRENCY        0xdb
#define C_GUILSNGLEFT     0xdc
#define C_GUILSNGRIGHT    0xdd
#define C_LY_ACUTE        0xde
#define C_UY_ACUTE        0xdf
#define C_DBLDAGGER       0xe0
#define C_CNTR_DOT        0xe1
#define C_SNGQUOTELOW     0xe2
#define C_DBLQUOTELOW     0xe3
#define C_PERTHOUSAND     0xe4
#define C_UA_CIRCUMFLEX   0xe5
#define C_UE_CIRCUMFLEX   0xe6
#define C_UA_ACUTE        0xe7
#define C_UE_DIERESIS     0xe8
#define C_UE_GRAVE        0xe9
#define C_UI_ACUTE        0xea
#define C_UI_CIRCUMFLEX   0xeb
#define C_UI_DIERESIS     0xec
#define C_UI_GRAVE        0xed
#define C_UO_ACUTE        0xee
#define C_UO_CIRCUMFLEX   0xef
#define C_LOGO            0xf0
#define C_UO_GRAVE        0xf1
#define C_UU_ACUTE        0xf2
#define C_UU_CIRCUMFLEX   0xf3
#define C_UU_GRAVE        0xf4
#define C_LI_DOTLESS      0xf5
#define C_CIRCUMFLEX      0xf6
#define C_TILDE           0xf7
#define C_MACRON          0xf8
#define C_BREVE           0xf9
#define C_DOTACCENT       0xfa
#define C_RING            0xfb
#define C_CEDILLA         0xfc
#define C_HUNGARUMLAT     0xfd
#define C_OGONEK          0xfe
```

```
#define C_CARON             0xff
/*
 * common shortcuts for low 32 codes
 */
#define C_NUL               C_NULL
#define C_STX               C_CTRL_B
#define C_ETX               C_CTRL_C
#define C_BEL               C_CTRL_G
#define C_BS                C_CTRL_H
#define C_HT                C_CTRL_I
#define C_VT                C_CTRL_K
#define C_FF                C_CTRL_L
#define C_SO                C_CTRL_N
#define C_SI                C_CTRL_O
#define C_DC1               C_CTRL_Q
#define C_DC2               C_CTRL_R
#define C_DC3               C_CTRL_S
#define C_DC4               C_CTRL_T
#define C_CAN               C_CTRL_X
#define C_EM                C_CTRL_Y
#define C_ESC               C_ESCAPE
/*
 * Some alternative names
 */
#define C_CR                C_ENTER
#define C_CTRL_M            C_ENTER
#define C_CTRL_I            C_TAB
#define C_CTRL_J            C_LINEFEED
#define C_LF                C_LINEFEED
#define C_CTRL_N            C_SHIFT_OUT
#define C_CTRL_O            C_SHIFT_IN
#define C_FS                C_ENSPACE
#define C_FIELD_SEP         C_FS
#define C_HYPHEN            C_MINUS
#define C_GRAVE             C_BACKQUOTE
#define C_PARTIAL_DIFF      C_L_DELTA
#define C_SUM               C_U_SIGMA
#define C_PRODUCT           C_U_PI
#define C_RADICAL           C_ROOT
#define C_LOZENGE           C_DIAMONDBULLET
```

Text characters may be represented by the standard C type char or by the GEOS type Chars. The difference shows up in debugging. If printing the value of a string as char, then the debugger will output ASCII text. If the string is treated as Chars, then the debugger will print out the constant names.

Include: char.h

Routines

■ ChunkArrayHeader

```
typedef struct {
    word    CAH_count;         /* # of elements in chunk array */
    word    CAH_elementSize;   /* Size of each element (in bytes) */
    word    CAH_curOffset;     /* For internal use only */
    word    CAH_offset;        /* Offset from start of chunk to first element */
} ChunkArrayHeader;
```

Every chunk array begins with a **ChunkArrayHeader**. This structure contains information about the chunk array. Applications should never change the contents of the **ChunkArrayHeader**; only the chunk array routines should do this. However, applications can examine the header if they wish.

Contents: There are four word-length fields in the **ChunkArrayHeader**:

CAH_count This word contains the number of elements in the chunk array.

CAH_elementSize
This word contains the size of each element (in bytes). If the elements are variable-sized, *CAH_elementSize* will be zero.

CAH_curOffset
This word is used by **ChunkArrayEnum()** for bookkeeping.

CAH_offset This is the offset from the start of the chunk to the first element in the array.

■ ChunkHandle

```
typedef word ChunkHandle;
```

Chunk handles are offsets into a local memory heap. To find the current location of a chunk in an LMem heap, combine the segment address of the heap with the chunk handle. From this location you can read the current offset of the chunk itself.

See Also: optr, LMemDeref()

■ ChunkMapList

```
typedef struct {
    word    CML_source;
    word    CML_dest;
} ChunkMapList;
```

■ ClassFlags

```
typedef ByteFlags ClassFlags;
```

```
    #define CLASSF_HAS_DEFAULT       0x80
    #define CLASSF_MASTER_CLASS      0x40
    #define CLASSF_VARIANT_CLASS     0x20
    #define CLASSF_DISCARD_ON_SAVE   0x10
    #define CLASSF_NEVER_SAVED       0x08
    #define CLASSF_HAS_RELOC         0x04
    #define CLASSF_C_HANDLERS        0x02
```

This record is stored in the **ClassStruct** structure's *Class_flags* field. These flags are internal and may not be set or retrieved directly. See the entry on **@class** for more information about these flags.

ClassStruct

```
typedef  struct    _ClassStruct {
    struct _ClassStruct *Class_superClass;/* superclass pointer */
    word        Class_masterOffset;    /* offset to master offset in chunk */
    word        Class_methodCount;     /* number of methods in this class */
    word        Class_instanceSize;    /* size of entire master group */
    word        Class_vdRelocTable;    /* offset to vardata relocation table */
    word        Class_relocTable;      /* offset to relocation table */
    ClassFlags  Class_flags;           /* a record of ClassFlags */
    byte        Class_masterMessages;  /* internal flags for optimization */
} ClassStruct;
```

This is the structure that defines a class. It is internal and used only very rarely by anything other than the kernel and the UI.

ClipboardItemFlags

```
typedef WordFlags ClipboardItemFlags;
    #define CIF_QUICK         0x4000
    #define TIF_NORMAL        0x0000
```

ClipboardItemFormat

```
typedef enum /* word */ {
    CIF_TEXT,
    CIF_GRAPHICS_STRING,
    CIF_FILES,
    CIF_SPREADSHEET,
    CIF_INK,
    CIF_GROBJ,
    CIF_GEODEX,
    CIF_BITMAP,
    CIF_SOUND_SYNTH,
    CIF_SOUND_SAMPLE
} ClipboardItemFormat;
```

Routines

■ ClipboardItemFormatID

```
typedef dword ClipboardItemFormatID;
```

■ ClipboardItemFormatInfo

```
typedef struct {
    ClipboardItemFormatID     CIFI_format;
    word                      CIFI_extra1;
    word                      CIFI_extra2;
    VMChain                   CIFI_vmChain;
    GeodeToken                CIFI_renderer;
} ClipboardItemFormatInfo;
```

■ ClipboardItemHeader

```
typedef struct {
    optr                      CIH_owner;
    ClipboardItemFlags        CIH_flags;
    ClipboardItemNameBuffer   CIH_name;
    word                      CIH_formatCount;
    optr                      CIH_sourceID;
    FormatArray               CIH_formats;
    dword                     CIH_reserved;
} ClipboardItemHeader;
```

■ ClipboardItemNameBuffer

```
typedef char ClipboardItemNameBuffer[CLIPBOARD_ITEM_NAME_LENGTH+1];
```

■ ClipboardQueryArgs

See **ClipboardQueryItem()**.

■ ClipboardQuickNotifyFlags

```
typedef WordFlags ClipboardQuickNotifyFlags;
    #define CQNF_ERROR                0x8000
    #define CQNF_SOURCE_EQUAL_DEST    0x4000
    #define CQNF_MOVE                 0x2000
    #define CQNF_COPY                 0x1000
    #define CQNF_NO_OPERATION         0x0800
    #define CQNF_UNUSED               0x04ff
```

These flags give information about the success or failure of a quick transfer operation.

■ ClipboardQuickTransferFeedback

```
typedef enum {
    CQTF_SET_DEFAULT,
    CQTF_CLEAR_DEFAULT,
    CQTF_MOVE,
    CQTF_COPY,
    CQTF_CLEAR
} ClipboardQuickTransferFeedback;
```

■ ClipboardQuickTransferFlags

```
    typedef WordFlags ClipboardQuickTransferFlags;
        #define CQTF_IN_PROGRESS  0x8000
        #define CQTF_COPY_ONLY    0x4000
        #define CQTF_USE_REGION   0x2000
        #define CQTF_NOTIFICATION 0x1000
```

■ ClipboardQuickTransferRegionInfo

```
    typedef struct {
        word    CQTRI_paramAX;
        word    CQTRI_paramBX;
        word    CQTRI_paramCX;
        word    CQTRI_paramDX;
        Point   CQTRI_regionPos;
        dword   CQTRI_strategy;
        dword   CQTRI_region;
    } ClipboardQuickTransferRegionInfo;
```

■ ClipboardRequestArgs

See entry for **ClipboardRequestItemFormat()**.

■ CMYKTransfer

```
    typedef struct {
        byte    CMYKT_cyan[256];
        byte    CMYKT_magenta[256];
        byte    CMYKT_yellow[256];
        byte    CMYKT_black[256];
    } CMYKTransfer;
```

■ Color

```
    typedef ByteEnum Color;
        #define C_BLACK           0
        #define C_BLUE            1
        #define C_GREEN           2
        #define C_CYAN            3
        #define C_RED             4
        #define C_VIOLET          5
        #define C_BROWN           6
        #define C_LIGHT_GRAY      7
```

```
#define C_DARK_GRAY       8
#define C_LIGHT_BLUE      9
#define C_LIGHT_GREEN     10
#define C_LIGHT_CYAN      11
#define C_LIGHT_RED       12
#define C_LIGHT_VIOLET    13
#define C_YELLOW          14
#define C_WHITE           15

#define C_GRAY_0          0x10
#define C_GRAY_7          0x11
#define C_GRAY_13         0x12
#define C_GRAY_20         0x13
#define C_GRAY_27         0x14
#define C_GRAY_33         0x15
#define C_GRAY_40         0x16
#define C_GRAY_47         0x17
#define C_GRAY_53         0x18
#define C_GRAY_60         0x19
#define C_GRAY_68         0x1a
#define C_GRAY_73         0x1b
#define C_GRAY_80         0x1c
#define C_GRAY_88         0x1d
#define C_GRAY_93         0x1e
#define C_GRAY_100        0x1f

#define C_UNUSED_0        0x20
#define C_UNUSED_1        0x21
#define C_UNUSED_2        0x22
#define C_UNUSED_3        0x23
#define C_UNUSED_4        0x24
#define C_UNUSED_5        0x25
#define C_UNUSED_6        0x26
#define C_UNUSED_7        0x27

#define C_R0_G0_B0        0x28
#define C_R0_G0_B1        0x29
#define C_R0_G0_B2        0x2a
#define C_R0_G0_B3        0x2b
#define C_R0_G0_B4        0x2c
#define C_R0_G0_B5        0x2d
#define C_R0_G1_B0        0x2e
#define C_R0_G1_B1        0x2f
#define C_R0_G1_B2        0x30
#define C_R0_G1_B3        0x31
#define C_R0_G1_B4        0x32
#define C_R0_G1_B5        0x33
#define C_R0_G2_B0        0x34
#define C_R0_G2_B1        0x35
#define C_R0_G2_B2        0x36
```

```
#define C_R0_G2_B3        0x37
#define C_R0_G2_B4        0x38
#define C_R0_G2_B5        0x39

#define C_R0_G3_B0        0x3a
#define C_R0_G3_B1        0x3b
#define C_R0_G3_B2        0x3c
#define C_R0_G3_B3        0x3d
#define C_R0_G3_B4        0x3e
#define C_R0_G3_B5        0x3f
#define C_R0_G4_B0        0x40
#define C_R0_G4_B1        0x41
#define C_R0_G4_B2        0x42
#define C_R0_G4_B3        0x43
#define C_R0_G4_B4        0x44
#define C_R0_G4_B5        0x45
#define C_R0_G5_B0        0x46
#define C_R0_G5_B1        0x47
#define C_R0_G5_B2        0x48
#define C_R0_G5_B3        0x49
#define C_R0_G5_B4        0x4a
#define C_R0_G5_B5        0x4b

#define C_R1_G0_B0        0x4c
#define C_R1_G0_B1        0x4d
#define C_R1_G0_B2        0x4e
#define C_R1_G0_B3        0x4f
#define C_R1_G0_B4        0x50
#define C_R1_G0_B5        0x51
#define C_R1_G1_B0        0x52
#define C_R1_G1_B1        0x53
#define C_R1_G1_B2        0x54
#define C_R1_G1_B3        0x55
#define C_R1_G1_B4        0x56
#define C_R1_G1_B5        0x57
#define C_R1_G2_B0        0x58
#define C_R1_G2_B1        0x59
#define C_R1_G2_B2        0x5a
#define C_R1_G2_B3        0x5b
#define C_R1_G2_B4        0x5c
#define C_R1_G2_B5        0x5d

#define C_R1_G3_B0        0x5e
#define C_R1_G3_B1        0x5f
#define C_R1_G3_B2        0x60
#define C_R1_G3_B3        0x61
#define C_R1_G3_B4        0x62
#define C_R1_G3_B5        0x63
#define C_R1_G4_B0        0x64
#define C_R1_G4_B1        0x65
```

Routines

```
#define C_R1_G4_B2        0x66
#define C_R1_G4_B3        0x67
#define C_R1_G4_B4        0x68
#define C_R1_G4_B5        0x69
#define C_R1_G5_B0        0x6a
#define C_R1_G5_B1        0x6b
#define C_R1_G5_B2        0x6c
#define C_R1_G5_B3        0x6d
#define C_R1_G5_B4        0x6e
#define C_R1_G5_B5        0x6f

#define C_R2_G0_B0        0x70
#define C_R2_G0_B1        0x71
#define C_R2_G0_B2        0x72
#define C_R2_G0_B3        0x73
#define C_R2_G0_B4        0x74
#define C_R2_G0_B5        0x75
#define C_R2_G1_B0        0x76
#define C_R2_G1_B1        0x77
#define C_R2_G1_B2        0x78
#define C_R2_G1_B3        0x79
#define C_R2_G1_B4        0x7a
#define C_R2_G1_B5        0x7b
#define C_R2_G2_B0        0x7c
#define C_R2_G2_B1        0x7d
#define C_R2_G2_B2        0x7e
#define C_R2_G2_B3        0x7f
#define C_R2_G2_B4        0x80
#define C_R2_G2_B5        0x81

#define C_R2_G3_B0        0x82
#define C_R2_G3_B1        0x83
#define C_R2_G3_B2        0x84
#define C_R2_G3_B3        0x85
#define C_R2_G3_B4        0x86
#define C_R2_G3_B5        0x87
#define C_R2_G4_B0        0x88
#define C_R2_G4_B1        0x89
#define C_R2_G4_B2        0x8a
#define C_R2_G4_B3        0x8b
#define C_R2_G4_B4        0x8c
#define C_R2_G4_B5        0x8d
#define C_R2_G5_B0        0x8e
#define C_R2_G5_B1        0x8f
#define C_R2_G5_B2        0x90
#define C_R2_G5_B3        0x91
#define C_R2_G5_B4        0x92
#define C_R2_G5_B5        0x93

#define C_R3_G0_B0        0x94
```

```
#define C_R3_G0_B1        0x95
#define C_R3_G0_B2        0x96
#define C_R3_G0_B3        0x97
#define C_R3_G0_B4        0x98
#define C_R3_G0_B5        0x99
#define C_R3_G1_B0        0x9a
#define C_R3_G1_B1        0x9b
#define C_R3_G1_B2        0x9c
#define C_R3_G1_B3        0x9d
#define C_R3_G1_B4        0x9e
#define C_R3_G1_B5        0x9f
#define C_R3_G2_B0        0xa0
#define C_R3_G2_B1        0xa1
#define C_R3_G2_B2        0xa2
#define C_R3_G2_B3        0xa3
#define C_R3_G2_B4        0xa4
#define C_R3_G2_B5        0xa5

#define C_R3_G3_B0        0xa6
#define C_R3_G3_B1        0xa7
#define C_R3_G3_B2        0xa8
#define C_R3_G3_B3        0xa9
#define C_R3_G3_B4        0xaa
#define C_R3_G3_B5        0xab
#define C_R3_G4_B0        0xac
#define C_R3_G4_B1        0xad
#define C_R3_G4_B2        0xae
#define C_R3_G4_B3        0xaf
#define C_R3_G4_B4        0xb0
#define C_R3_G4_B5        0xb1
#define C_R3_G5_B0        0xb2
#define C_R3_G5_B1        0xb3
#define C_R3_G5_B2        0xb4
#define C_R3_G5_B3        0xb5
#define C_R3_G5_B4        0xb6
#define C_R3_G5_B5        0xb7

#define C_R4_G0_B0        0xb8
#define C_R4_G0_B1        0xb9
#define C_R4_G0_B2        0xba
#define C_R4_G0_B3        0xbb
#define C_R4_G0_B4        0xbc
#define C_R4_G0_B5        0xbd
#define C_R4_G1_B0        0xbe
#define C_R4_G1_B1        0xbf
#define C_R4_G1_B2        0xc0
#define C_R4_G1_B3        0xc1
#define C_R4_G1_B4        0xc2
#define C_R4_G1_B5        0xc3
#define C_R4_G2_B0        0xc4
```

```
#define C_R4_G2_B1        0xc5
#define C_R4_G2_B2        0xc6
#define C_R4_G2_B3        0xc7
#define C_R4_G2_B4        0xc8
#define C_R4_G2_B5        0xc9

#define C_R4_G3_B0        0xca
#define C_R4_G3_B1        0xcb
#define C_R4_G3_B2        0xcc
#define C_R4_G3_B3        0xcd
#define C_R4_G3_B4        0xce
#define C_R4_G3_B5        0xcf
#define C_R4_G4_B0        0xd0
#define C_R4_G4_B1        0xd1
#define C_R4_G4_B2        0xd2
#define C_R4_G4_B3        0xd3
#define C_R4_G4_B4        0xd4
#define C_R4_G4_B5        0xd5
#define C_R4_G5_B0        0xd6
#define C_R4_G5_B1        0xd7
#define C_R4_G5_B2        0xd8
#define C_R4_G5_B3        0xd9
#define C_R4_G5_B4        0xda
#define C_R4_G5_B5        0xdb

#define C_R5_G0_B0        0xdc
#define C_R5_G0_B1        0xdd
#define C_R5_G0_B2        0xde
#define C_R5_G0_B3        0xdf
#define C_R5_G0_B4        0xe0
#define C_R5_G0_B5        0xe1
#define C_R5_G1_B0        0xe2
#define C_R5_G1_B1        0xe3
#define C_R5_G1_B2        0xe4
#define C_R5_G1_B3        0xe5
#define C_R5_G1_B4        0xe6
#define C_R5_G1_B5        0xe7
#define C_R5_G2_B0        0xe8
#define C_R5_G2_B1        0xe9
#define C_R5_G2_B2        0xea
#define C_R5_G2_B3        0xeb
#define C_R5_G2_B4        0xec
#define C_R5_G2_B5        0xed
#define C_R5_G3_B0        0xee
#define C_R5_G3_B1        0xef
#define C_R5_G3_B2        0xf0
#define C_R5_G3_B3        0xf1
#define C_R5_G3_B4        0xf2
#define C_R5_G3_B5        0xf3
#define C_R5_G4_B0        0xf4
```

```
    #define C_R5_G4_B1        0xf5
    #define C_R5_G4_B2        0xf6
    #define C_R5_G4_B3        0xf7
    #define C_R5_G4_B4        0xf8
    #define C_R5_G4_B5        0xf9
    #define C_R5_G5_B0        0xfa
    #define C_R5_G5_B1        0xfb
    #define C_R5_G5_B2        0xfc
    #define C_R5_G5_B3        0xfd
    #define C_R5_G5_B4        0xfe
    #define C_R5_G5_B5        0xff

    #define C_LIGHT_GREY      C_LIGHT_GRAY
    #define C_DARK_GREY       C_DARK_GRAY
    #define C_BW_GREY         0x84
```

Include: color.h

■ ColorFlag

```
typedef ByteEnum ColorFlag;
    #define CF_INDEX 0
    #define CF_GRAY  1
    #define CF_SAME  2
    #define CF_RGB   0x80
```

Several color-related commands accept colors in a variety of formats. The **ColorFlag** enumerated type is used to specify how the color is being described. The **ColorFlag** is normally used as part of a **ColorQuad**. See **ColorQuad** for information about how to interpret color specifications using **ColorFlag**s.

■ ColorMapMode

```
typedef ByteFlags ColorMapMode;
    #define CMM_ON_BLACK 0x04 /* Set this bit if you’re drawing on black */
    #define CMM_MAP_TYPE 0x01 /* Either CMT_CLOSEST or CMT_DITHER) */
    #define LAST_MAP_MODE     (CMM_MAP_TYPE | CMM_ON_BLACK)
```

This structure defines how the system will try to simulate colors not in the palette. If the map type is CMT_CLOSEST, the closest available color will be used. If the map type is CMT_DITHER, the system will mix together two or more close colors in a dithered pattern. If you will be drawing against a black background, you may wish to set the CMM_ON_BLACK flag.

■ ColorQuad

```
typedef struct {
    byte        CQ_redOrIndex;
    ColorFlag   CQ_info;
    byte         CQ_green;
    byte         CQ_blue;
} ColorQuad;
```

This structure represents a color. The *CQ_info* field determines how the color is being described.

If the info field is CF_INDEX, then the color is being specified by its index, its place in the window's palette. The index is in the *CQ_redOrIndex* field; the the *CQ_green* and *CQ_blue* fields are meaningless for this specification.

If the info field is CF_RGB, then the color is specified by RGB (red, green, and blue) components. *CQ_redOrIndex* contains the color's red component, a number ranging from 0 to 255. The *CQ_green* and *CQ_blue* fields contain the color's green and blue components, respectively.

If the info field is CF_GRAY, then the color is being expressed as a grey scale. This is basically an optimized way of describing RGB colors where the red, green, and blue components are equal. The *CQ_redOrIndex* field contains the brightess, a number between 0 and 255. The *CQ_green* and *CQ_blue* fields are ignored.

When defining hatch patterns, it is possible have a CF_SAME info field. This means that the hatch lines should use the "same" color when drawing. That is, when hatching text, the text color will be used; when filling an area, the area color will be used. The *CQ_redOrIndex*, *CQ_green*, and *CQ_blue* fields are all ignored.

■ ColorQuadAsDWord

```
typedef dword ColorQuadAsDWord;
```

■ ColorTransfer

```
typedef struct {
    RGBDelta                 CT_data[125];
} ColorTransfer;
```

This structure consists of a 5x5x5 matrix of **RGBDelta** structures. This and be used to specify what sorts of adjustments to make to the color when displaying to a specific device. For instance, some color printers will wipe out certain colors if they try to use the amounts of ink suggested by the raw RGB values. The **ColorTransfer** structure thus serves to hold an array of "fudge factors" to tell the printer to use more or less ink than the raw RGB values would suggest.

■ ColorTransferData

```
typedef union {
    MonoTransfer CTD_mono;
    RGBTransfer  CTD_rgb;
    CMYKTransfer CTD_cmyk;
} ColorTransferData;
```

■ ColorTransferType

```
typedef ByteEnum ColorTransferType;
    #define CTT_MONO           0
    #define CTT_RGB            1
    #define CTT_CMYK           2
```

■ CommonParameters

```
typedef struct {
    word    CP_row;
    word    CP_column;
    word    CP_maxRow;
    word    CP_maxColumn;
    void    * CP_callback;
    void    * CP_cellParams; /* ptr to an instance of SpreadsheetClass */
} CommonParameters;
```

■ CompChildFlags

```
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
```

A record used when adding, moving, or removing children in an object tree. The record has one flag and a value, as follows:

CCF_MARK_DIRTY
: A flag indicating whether the object should be marked dirty at the end of the operation.

CCF_REFERENCE
: A child number; when adding or moving a child, this is the child number after which the new object should be inserted. It can be any number less than 32768, or it can be either of the two constants shown above (CCO_FIRST or CCO_LAST).

■ CountryType

```
typedef enum /* word */ {
    CT_UNITED_STATES=1,
    CT_CANADA,
    CT_UNITED_KINGDOM,
    CT_GERMANY,
    CT_FRANCE,
    CT_SPAIN,
    CT_ITALY,
    CT_DENMARK,
    CT_NETHERLANDS,
} CountryType;
```

■ CRangeEnumParams

```
typedef struct {
              RangeEnumParams         CREP_params;
              void                    *CREP_locals;
              PCB(RANGE_ENUM_CALLBACK_RETURN_TYPE, CREP_callback,
                 (RangeEnumCallbackParams));
} CRangeEnumParams;
```

The *CREP_callback* routine should be declared _pascal.

■ CurrencyFormatFlags

```
typedef ByteFlags CurrencyFormatFlags;
    #define CFF_LEADING_ZERO                   0x20
    #define CFF_SPACE_AROUND_SYMBOL            0x10
    #define CFF_USE_NEGATIVE_SIGN              0x08
    #define CFF_SYMBOL_BEFORE_NUMBER           0x04
    #define CFF_NEGATIVE_SIGN_BEFORE_NUMBER    0x02
    #define CFF_NEGATIVE_SIGN_BEFORE_SYMBOL    0x01
```

■ CustomDialogBoxFlags

```
typedef WordFlags CustomDialogBoxFlags;
    #define CDBF_SYSTEM_MODAL        0x8000
    #define CDBF_DIALOG_TYPE         0x6000
    #define CDBF_INTERACTION_TYPE    0x1e00
```

■ CustomDialogType

```
typedef ByteEnum CustomDialogType;
    #define CDT_QUESTION      0
    #define CDT_WARNING       1
    #define CDT_NOTIFICATION  2
    #define CDT_ERROR         3
    #define CDBF_DIALOG_TYPE_OFFSET     13
    #define CDBF_INTERACTION_TYPE_OFFSET 9
```

■ DACPlayFlags

```
typedef ByteFlags DACPlayFlags;
#define DACPF_CATENATE 0x80
```

■ DACReferenceByte

```
typedef enum {
    DACRB_NO_REFERENCE_BYTE,
    DACRB_WITH_REFERENCE_BYTE
} DACReferenceByte;
```

■ DACSampleFormat

```
typedef enum {
    DACSF_8_BIT_PCM,
    DACSF_2_TO_1_ADPCM,
    DACSF_3_TO_1_ADPCM,
    DACSF_4_TO_1_ADPCM
} DACSampleFormat;
```

This structure specifies what sort of sampling should be used when recording or playing a sampled sound.

■ DashPairArray

See: LineStyle

■ DateTimeFormat

```
typedef enum /* word */ {
   DTF_LONG,
   DTF_LONG_CONDENSED,
   DTF_LONG_NO_WEEKDAY,
   DTF_LONG_NO_WEEKDAY_CONDENSED,
   DTF_SHORT,
   DTF_ZERO_PADDED_SHORT,
   DTF_MD_LONG,
   DTF_MD_LONG_NO_WEEKDAY,
   DTF_MD_SHORT,
   DTF_MY_LONG,
   DTF_MY_SHORT,
   DTF_MONTH,
   DTF_WEEKDAY,
   DTF_HMS,
   DTF_HM,
   DTF_H,
   DTF_MS,
   DTF_HMS_24HOUR,
    DTF_HM_24HOUR,
} DateTimeFormat;
```

■ DayOfTheWeek

```
typedef enum {
    DOTW_SUNDAY,
    DOTW_MONDAY,
    DOTW_TUESAY,
    DOTW_WEDNESDAY,
    DOTW_THURSDAY,
    DOTW_FRIDAY,
    DOTW_SATURDAY
} DayOfTheWeek;
```

This enumerated type is used in the **TimerDateAndTime** structure.

■ DBGroup

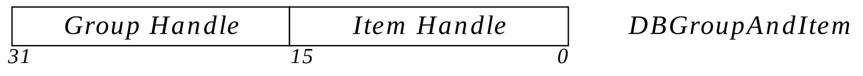
```
typedef word DBGroup;
```

This is the handle of a DB group. It is the VM handle of a DB group block. DB group handles do not change when a file is copied, or when it is closed and reopened.

■ DBGroupAndItem

```
typedef  dword DBGroupAndItem;
```

This is a dword which contains the group and item handles of a database item. The high word is the item's Group handle; the low word is the item's Item handle.



Macros are provided to create and parse the **DBGroupAndItem**:

DBCombineGroupAndItem()
Creates a **DBGroupAndItem** from given group and item handles.

```
DBCombineGroupAndItem(group, item);
```

DBExtractGroupFromGroupAndItem()
Extracts the **DBGroup** from a given **DBGroupAndItem**.

```
DBExtractGroupFromGroupAndItem(groupAndItem);
```

DBExtractItemFromGroupAndItem()
Extracts the **DBItem** from a given **DBGroupAndItem**.

```
DBExtractItemFromGroupAndItem(groupAndItem);
```

Include: geos.h

■ DBItem

```
typedef word DBItem;
```

This is the handle of a DB item. The **DBItem** and **DBGroup** together uniquely identify a DB item in a specified file.

■ DBReturn

```
typedef struct {
    word    DBR_group;
    word    DBR_item;
    word    unused1;
    word    unused2;
} DBReturn;
```

■ DefaultPrintSizes

```
typedef struct {
    word    paperWidth;
    word    paperHeight;
    word    documentWidth;
    word    documentHeight;
} DefaultPrintSizes;
```

■ DevicePresent

```
typedef enum /* word */ {
    DP_NOT_PRESENT=0xffff,
    DP_CANT_TELL=0,
    DP_PRESENT=1,
    DP_INVALID_DEVICE=0xfffe
} DevicePresent;
```

■ DirPathInfo

```
typedef word DirPathInfo;
    #define DPI_EXISTS_LOCALLY              0x8000
    #define DPI_ENTRY_NUMBER_IN_PATH        0x7f00
    #define DPI_ENTRY_NUMBER_IN_PATH_OFFSET 8
    #define DPI_STD_PATH                    0x00ff
    #define DPI_STD_PATH_OFFSET             0
```

■ DiskCopyCallback

```
typedef enum /* word */ {
    CALLBACK_GET_SOURCE_DISK,
    CALLBACK_REPORT_NUM_SWAPS,
    CALLBACK_GET_DEST_DISK,
    CALLBACK_VERIFY_DEST_DESTRUCTION,
    CALLBACK_REPORT_FORMAT_PCT,
    CALLBACK_REPORT_COPY_PCT
} DiskCopyCallback;
```

■ DiskCopyError

```
typedef enum /* word */ {
    ERR_DISKCOPY_INSUFFICIENT_MEM=0xd0,
    ERR_CANT_COPY_FIXED_DISKS,
    ERR_CANT_READ_FROM_SOURCE,
    ERR_CANT_WRITE_TO_DEST,
    ERR_INCOMPATIBLE_FORMATS,
    ERR_OPERATION_CANCELLED,
    ERR_CANT_FORMAT_DEST,
} DiskCopyError;
```

■ DiskFindResult

```
typedef enum /* word */ {
    DFR_UNIQUE,
    DFR_NOT_UNIQUE,
    DFR_NOT_FOUND,
} DiskFindResult;
```

■ DiskHandle

```
typedef Handle DiskHandle;
```

■ DiskInfoStruct

```
typedef struct {
    word        DIS_blockSize;
    sdword      DIS_freeSpace;
    sdword      DIS_totalSpace;
    VolumeName  DIS_name;
} DiskInfoStruct;
```

■ DiskRestoreError

```
typedef enum /* word */ {
    DRE_DISK_IN_DRIVE,
    DRE_DRIVE_NO_LONGER_EXISTS,
    DRE_REMOVABLE_DRIVE_DOESNT_HOLD_DISK,
    DRE_USER_CANCELED_RESTORE,
    DRE_COULDNT_CREATE_NEW_DISK_HANDLE,
    DRE_REMOVABLE_DRIVE_IS_BUSY,
} DiskRestoreError;
```

■ DisplayAspectRatio

```
typedef ByteEnum DisplayAspectRatio;
    #define DAR_NORMAL        0
    #define DAR_SQUISHED      1
    #define DAR_VERY_SQUISHED 2
```

■ DisplayClass

```
typedef ByteEnum DisplayClass;
    #define DC_TEXT           0
    #define DC_GRAY_1         1
    #define DC_GRAY_2         2
    #define DC_GRAY_4         3
    #define DC_GRAY_8         4
    #define DC_COLOR_2        5
    #define DC_COLOR_4        6
    #define DC_CF_RGB         7
```

■ DisplaySize

```
typedef ByteEnum DisplaySize;
    #define DS_TINY           0
    #define DS_STANDARD       1
    #define DS_LARGE          2
    #define DS_HUGE           3
```

■ DisplayType

```
typedef ByteFlags DisplayType;
    #define DT_DISP_SIZE         0xc0
    #define DT_DISP_ASPECT_RATIO 0x30
    #define DT_DISP_CLASS        0x0f
```

■ DistanceUnit

```
typedef ByteEnum DistanceUnit;
    #define DU_POINTS                     0
    #define DU_INCHES                     1
    #define DU_CENTIMETERS                2
    #define DU_MILLIMETERS                3
    #define DU_PICAS                      4
    #define DU_EUR_POINTS                 5
    #define DU_CICEROS                    6
    #define DU_POINTS_OR_MILLIMETERS      7
    #define DU_INCHES_OR_CENTIMETERS      8
    #define LOCAL_DISTANCE_BUFFER_SIZE   32
```

■ DocQuitStatus

```
typedef enum /* word */ {
     DQS_OK,
```

Routines

```
        DQS_CANCEL,
        DQS_DELAYED,
        DQS_SAVE_ERROR
} DocQuitStatus;
```

■ DocumentSize

```
typedef struct {
    int     leftMargin;
    int     topMargin;
    int     width;
    int     height;
} DocumentSize;
```

■ DosCodePage

```
typedef enum /* word */ {
    CODE_PAGE_US=437,
    CODE_PAGE_MULTILINGUAL=850,
    CODE_PAGE_PORTUGUESE=860,
    CODE_PAGE_CANADIAN_FRENCH=863,
    CODE_PAGE_NORDIC=865
} DosCodePage;
```

■ DosDotFileName

```
typedef char DosDotFileName[DOS_DOT_DOS_FILE_NAME_SIZE];
```

■ DosExecFlags

```
typedef ByteFlags DosExecFlags;
    #define DEF_PROMPT             0x80      /* prompt user to return to GEOS */
    #define DEF_FORCED_SHUTDOWN    0x40      /* force shutdown; no abort */
    #define DEF_INTERACTIVE        0x20      /* program is interactive shell */
```

Flags used with **DosExec()**. **DosExec()** executes a DOS program based on these flags.

■ DosFileInfoStruct

```
typedef struct {
     byte DFIS_attributes;
     dword DFIS_modTimeDate;
     dword DFIS_fileSize;
     char DFIS_name[DOS_DOT_FILE_NAME_LENGTH_ZT];
     word DFIS_pathInfo;
} DosFileInfoStruct;
```

■ DosNoDotFileName

```
typedef char DosNoDotFileName[DOS_NO_DOT_DOS_FILE_NAME_SIZE];
```

■ DrawMask

```
typedef byte DrawMask[8];
```

The graphics system uses this structure for defining custom draw masks.

■ DriveType

```
typedef ByteEnum DriveType;
    #define DRIVE_5_25        0
    #define DRIVE_3_5         1
    #define DRIVE_FIXED       2
    #define DRIVE_RAM         3
    #define DRIVE_CD_ROM      4
    #define DRIVE_8           5
    #define DRIVE_UNKNOWN     0xf
} DriveType;
```

Several routines (in particular, **DriveGetStatus()**) provide information about drives used by the computer running GEOS. These routines return a member of the **DriveTypes** enumerated type. Note that while the type is byte-length, all of the values are guaranteed to fit in four bits; thus, routines like **DriveGetStatus()** can return a **DriveTypes** value in the low four bits and other flags in the high four bits of a single byte.

■ DriverAttrs

```
typedef WordFlags DriverAttrs;
    #define DA_FILE_SYSTEM            0x8000
    #define DA_CHARACTER              0x4000
    #define DA_HAS_EXTENDED_INFO      0x2000
```

This record contains flags that indicate a given driver's attributes. This record is stored in the driver's **DriverInfoStruct** structure.

■ DriverExtendedInfoStruct

```
typedef struct {
    DriverInfoStruct DEIS_common;     /* The base driver info structure */
    MemHandle        DEIS_resource;   /* Handle of driver's DriverExtendedInfo
                                       * table. */
} DriverExtendedInfoStruct;
```

This structure is used by Preferences to locate the names of devices supported by a particular driver.

■ DriverExtendedInfoTable

```
typedef struct {
    LMemBlockHeader DEIT_common;
```

Routines

```
    word            DEIT_numDevices;
    ChunkHandle     DEIT_ChunkHandle;
    word            DEIT_infoTable;
} DriverExtendedInfoTable;
```

■ DriverInfoStruct

```
typedef struct {
     void (*DIS_strategy)();            /* Pointer to strategy routine */
     DriverAttrs DIS_driverAttributes;  /* driver’s attribute flags */
     DriverType  DIS_driverType;        /* driver’s type */
} DriverInfoStruct;
```

This structure defines the characteristics of a particular driver. In general, applications will not need to access this structure unless they use a driver directly.

■ DriverType

```
typedef enum {
    DRIVER_TYPE_VIDEO = 1,          /* Video drivers */
    DRIVER_TYPE_INPUT,              /* Input (keyboard, mouse) drivers */
    DRIVER_TYPE_MASS_STORAGE,       /* Disk/Drive drivers */
    DRIVER_TYPE_STREAM,             /* Stream and port drivers */
    DRIVER_TYPE_FONT,               /* Font drivers */
    DRIVER_TYPE_OUTPUT,             /* Output (not video and printer) drivers */
    DRIVER_TYPE_LOCALIZATION,       /* Localization drivers */
    DRIVER_TYPE_FILE_SYSTEM,        /* File system drivers */
    DRIVER_TYPE_PRINTER,            /* Printer drivers */
    DRIVER_TYPE_SWAP,               /* Swap drivers */
    DRIVER_TYPE_POWER_MANAGEMENT,   /* Power management drivers */
    DRIVER_TYPE_TASK_SWITCH,        /* Task switch drivers */
    DRIVER_TYPE_NETWORK             /* Network file system drivers */
} DriverType;
```

This enumerated type has one value for each type of driver in the system. It is used primarily with **GeodeUseDriver()** and its associated routines. Each driver stores its type in its **DriverInfoStruct** structure.

■ DWFixed

```
typedef struct {
    word WWF_frac;
    dword WWF_int;
} DWFixed;
```

■ dword

```
typedef unsigned long dword;
```

■ DWordFlags

```
typedef dword DWordFlags;
```

■ ElementArrayHeader

```
typedef struct {
    ChunkArrayHeaderEAH_meta;        /* chunk array header structure */
    word            EAH_freePtr;     /* First free element */
} ElementArrayHeader;
```

Every element array must begin with an **ElementArrayHeader**. Since element arrays are special kinds of chunk arrays, the **ElementArrayHeader** must itself begin with a **ChunkArrayHeader**. The structure contains one additional field, *EAH_freePtr*. This is used to keep track of the freed elements in the element array. Applications should not examine or change this field.

■ EndOfSongFlags

```
typedef ByteFlags EndOfSongFlags;
          #define EOSF_UNLOCK 0x0080 /* unlock block at EOS ? */
          #define EOSF_DESTROY 0x0040 /* destroy block at EOS ? */

          #define UNLOCK_ON_EOS EOSF_UNLOCK
          #define DESTROY_ON_EOS EOSF_DESTROY
```

■ EntryPointRelocation

```
typedef struct {
    char    EPR_geodeName[GEODE_NAME_SIZE];
    word    EPR_entryNumber;
} EntryPointRelocation;
```

■ EnvelopeOrientation

```
typedef ByteEnum EnvelopeOrientation;
    #define EO_PORTAIT_LEFT   0x00
    #define EO_PORTAIT_RIGHT  0x01
    #define EO_LANDSCAPE_UP   0x02
    #define EO_LANDSCAPE_DOWN 0x03
```

■ EnvelopePath

```
typedef ByteEnum EnvelopePath;
    #define EP_LEFT           0x00
    #define EP_CENTER         0x01
    #define EP_RIGHT          0x02
```

Routines

■ Errors

```
    #define ERROR_UNSUPPORTED_FUNCTION           1
    #define ERROR_FILE_NOT_FOUND                 2
    #define ERROR_PATH_NOT_FOUND                 3
    #define ERROR_TOO_MANY_OPEN_FILES            4
    #define ERROR_ACCESS_DENIED                  5
    #define ERROR_INSUFFICIENT_MEMORY            8
    #define ERROR_INVALID_VOLUME                 15
    #define ERROR_IS_CURRENT_DIRECTORY           16
    #define ERROR_DIFFERENT_DEVICE               17
    #define ERROR_NO_MORE_FILES                  18
    #define ERROR_WRITE_PROTECTED                19
    #define ERROR_UNKNOWN_VOLUME                 20
    #define ERROR_DRIVE_NOT_READY                21
    #define ERROR_CRC_ERROR                      23
    #define ERROR_SEEK_ERROR                     25
    #define ERROR_UNKNOWN_MEDIA                  26
    #define ERROR_SECTOR_NOT_FOUND               27
    #define ERROR_WRITE_FAULT                    29
    #define ERROR_READ_FAULT                     30
    #define ERROR_GENERAL_FAILURE                31
    #define ERROR_SHARING_VIOLATION              32
    #define ERROR_ALREADY_LOCKED                 33
    #define ERROR_SHARING_OVERFLOW               36
    #define ERROR_SHORT_READ_WRITE               128
    #define ERROR_INVALID_LONGNAME               129
    #define ERROR_FILE_EXISTS                    130
    #define ERROR_DOS_EXEC_IN_PROGRESS           131
    #define ERROR_FILE_IN_USE                    132
    #define ERROR_ARGS_TOO_LONG                  133
    #define ERROR_DISK_UNAVAILABLE               134
    #define ERROR_DISK_STALE                     135
    #define ERROR_FILE_FORMAT_MISMATCH           136
    #define ERROR_CANNOT_MAP_NAME                137
    #define ERROR_DIRECTORY_NOT_EMPTY            138
    #define ERROR_ATTR_NOT_SUPPORTED             139
    #define ERROR_ATTR_NOT_FOUND                 140
    #define ERROR_ATTR_SIZE_MISMATCH             141
    #define ERROR_ATTR_CANNOT_BE_SET             142
    #define ERROR_CANNOT_MOVE_DIRECTORY          143
    #define ERROR_PATH_TOO_LONG                  144
    #define ERROR_ARGS_INVALID                   145
    #define ERROR_CANNOT_FIND_COMMAND_INTERPRETER 146
    #define ERROR_NO_TASK_DRIVER_LOADED          147
```

■ ErrorCheckingFlags

```
typedef WordFlags ErrorCheckingFlags;
    #define ECF_REGION           0x8000
    #define ECF_HEAP_FREE_BLOCKS 0x4000
```

Routines

```
#define ECF_LMEM_INTERNAL     0x2000
#define ECF_LMEM_FREE_AREAS   0x1000
#define ECF_LMEM_OBJECT       0x0800
#define ECF_BLOCK_CHECKSUM    0x0400
#define ECF_GRAPHICS          0x0200
#define ECF_SEGMENT           0x0100
#define ECF_NORMAL            0x0080
#define ECF_VMEM              0x0040
#define ECF_APP               0x0020
#define ECF_LMEM_MOVE         0x0010
#define ECF_UNLOCK_MOVE       0x0008
#define ECF_VMEM_DISCARD      0x0004
```

Error checking flags are used when setting the system's error-checking level with **SysSetECLevel()**. The flags above may be individually set or cleared. It is important to use error checking when debugging; it can help catch obscure bugs that might otherwise go unnoticed until after a product ships.

■ EvalErrorData

```
typedef struct {
    byte        EED_errorCode;      /* ParserScannerEvaluatorError */
} EvalErrorData;
```

■ EvalFlags

```
typedef ByteFlags EvalFlags;
    #define EF_MAKE_DEPENDENCIES     0x80
    #define EF_ONLY_NAMES            0x40
    #define EF_KEEP_LAST_CELL        0x20
    #define EF_NO_NAMES              0x10
    #define EF_ERROR_PUSHED          0x08
    #define EVAL_MAX_NESTED_LEVELS 32
```

■ EvalFunctionData

```
typedef struct {
    FunctionID   EFD_functionID;
    word         EFD_nArgs;
} EvalFunctionData;
```

■ EvalNameData

```
typedef struct {
    word    END_name;
} EvalNameData;
```

■ EvalOperatorData

```
typedef struct {
    OperatorType EOD_opType;
} EvalOperatorData;
```

■ EvalStackArgumentData

```
typedef union {
    EvalStringData  ESAD_string;
    EvalRangeData   ESAD_range;
    EvalErrorData   ESAD_error;
} EvalStackArgumentData;
```

■ EvalParameters

```
typedef struct {
    CommonParameters          EP_common;
    EvalFlags                 EP_flags;
    word            EP_fpStack;
    word                      EP_depHandle;
    word                      EP_nestedLevel;
    dword                     EP_nestedAddresses[EVAL_MAX_NESTED_LEVELS];
} EvalParameters;
```

■ EvalRangeData

```
typedef struct {
    CellReference   ERD_firstCell;
    CellReference   ERD_lastCell;
} EvalRangeData;
```

■ EvalStackArgumentType

```
typedef ByteFlags EvalStackArgumentType;
    #define ESAT_EMPTY          0x80
    #define ESAT_ERROR          0x40
    #define ESAT_RANGE          0x20
    #define ESAT_STRING         0x10
    #define ESAT_NUMBER         0x08
    #define ESAT_NUM_TYPE       0x03
    #define ESAT_TOP_OF_STACK   0
    #define ESAT_NAME           (ESAT_RANGE | ESAT_STRING)
    #define ESAT_FUNCTION       (ESAT_NUMBER | ESAT_STRING)
```

■ EvalStackOperatorData

```
typedef union {
    EvalOperatorDataESOD_operator;
    EvalFunctionDataESOD_function;
} EvalStackOperatorData;
```

■ EvalStackOperatorType

```
typedef ByteEnum EvalStackOperatorType;
    #define ESOT_OPERATOR     0
    #define ESOT_FUNCTION     1
    #define ESOT_OPEN_PAREN   2
    #define ESOT_TOP_OF_STACK 3
```

■ EvalStringData

```
typedef struct {
    word    ESD_length;
} EvalStringData;
```

■ EventHandle

```
typedef Handle      EventHandle;
```

■ ExitFlags

```
typedef ByteFlags ExitFlags;
    #define EF_PANIC          0x80
    #define EF_RUN_DOS        0x40
    #define EF_OLD_EXIT       0x20
    #define EF_RESET          0x10
    #define EF_RESTART        0x08
```

■ ExportControlFeatures

```
typedef ByteFlags ExportControlFeatures;
    #define EXPORTCF_BASIC          0x01
```

■ ExportControlToolboxFeatures

```
typedef ByteFlags ExportControlToolboxFeatures;
    #define EXPORTCTF_DIALOG_BOX    0x01
```

■ Routines

■ FALSE

```
#define FALSE       0
#define TRUE       (~0)  /* use as return value, not for comparisons */
```

■ FFFieldMessageBlock

```
typedef struct {
    char    textBuffer[100];
    int     startOffset;
} FFFieldMessageBlock;
```

■ FileAccess

```
typedef ByteEnum FileAccess
    #define FA_READ_ONLY      0
    #define FA_WRITE_ONLY     1
    #define FA_READ_WRITE     2
```

■ FileAccessFlags

```
typedef ByteFlags FileAccessFlags;
    #define FILE_DENY_RW 0x10
    #define FILE_DENY_W 0x20
    #define FILE_DENY_R 0x30
    #define FILE_DENY_NONE 0x40
    #define FILE_ACCESS_R 0x00
    #define FILE_ACCESS_W 0x01
    #define FILE_ACCESS_RW 0x02
    #define FILE_NO_ERRORS 0x80
```

When you open a file for bytewise access, you must pass a record of **FileAccessFlags**. The **FileAccessFlags** record specifies two things: what kind of access the caller wants, and what type of access is permitted to other geodes. A set of **FileAccessFlags** is thus a bit-wise “or” of two different values. The first specifies what kind of access the calling geode wants and has the following values:

FILE_ACCESS_R
The geode will only be reading from the file.

FILE_ACCESS_W
The geode will write to the file but will not read from it.

FILE_ACCESS_RW
The geode will read from and write to the file.

The second part specifies what kind of access other geodes may have. Note that if you try to deny a permission which has already been given to another geode (e.g. you open a file with FILE_DENY_W when another geode has the file open for write-access), the call will fail. It has the following values:

FILE_DENY_RW
No geode may open the file for any kind of access, whether read, write, or read/write.

FILE_DENY_R
No geode may open the file for read or read/write access.

FILE_DENY_W
No geode may open the file for write or read/write access.

FILE_DENY_NONE
Other geodes may open the file for any kind of access.

Two flags, one from each of these sets of values, are combined to make up a proper **FileAccessFlags** value. For example, to open the file for read-only access while prohibiting other geodes from writing to the file, you would pass the flags “(FILE_ACCESS_R | FILE_DENY_W)”.

■ FileAccessRights

```
typedef char FileAccessRights[FILE_RIGHTS_SIZE];
```

■ FileAttrs

```
typedef ByteFlags FileAttrs;
    #define FA_ARCHIVE              0x20
    #define FA_SUBDIR               0x10
    #define FA_VOLUME               0x8
    #define FA_SYSTEM               0x4
    #define FA_HIDDEN               0x2
    #define FA_RDONLY               0x1
    #define FILE_ATTR_NORMAL        0
    #define FILE_ATTR_READ_ONLY     FA_RDONLY
    #define FILE_ATTR_HIDDEN        FA_HIDDEN
    #define FILE_ATTR_SYSTEM        FA_SYSTEM
    #define FILE_ATTR_VOLUME_LABEL  FA_VOLUME
```

Every DOS or GEOS file has certain attributes. These attributes mark such things as whether the file is read-only. With GEOS files, the attributes can be accessed by using the extended attribute FEA_FILE_ATTR. You can also access any file’s standard attributes with the routines **FileGetAttributes()** and **FileSetAttributes()**; these routines work for both GEOS files and plain DOS files.

The **FileAttrs** field contains the following bits:

FA_ARCHIVE	This flag is set if the file requires backup. Backup programs typically clear this bit.
FA_SUBDIR	This flag is set if the “file” is actually a directory. Geodes may not change this flag.
FA_VOLUME	This flag is set if the “file” is actually the volume label. This flag will be *off* for all files a geode will ever see. Geodes may not change this flag.
FA_SYSTEM	This flag is set if the file is a system file. Geodes should not change this bit.
FA_HIDDEN	This flag is set if the file is hidden.
FA_RDONLY	This flag is set if the file is read-only.

See Also: FileGetAttrs(), FileSetAttrs()

Include: file.h

■ FileChangeNotificationData

```
typedef struct {
    PathName        FCND_pathname;
    DiskHandle      FCND_diskHandle;
    FileChangeType  FCND_changeType;
} FileChangeNotificationData;
```

■ FileChangeType

```
typedef ByteEnum FileChangeType;
    #define FCT_CREATE        0
    #define FCT_DELETE        1
    #define FCT_RENAME        2
    #define FCT_CONTENTS      3
    #define FCT_DISK_FORMAT   4
```

■ FileCopyrightNotice

```
typedef char FileCopyrightNotice[GFH_NOTICE_SIZE];
```

■ FileCreateFlags

```
typedef WordFlags FileCreateFlags;
    #define FCF_NATIVE        0x8000
    #define FCF_MODE          0x0300 /* Filled with FILE_CREATE_* constant */
    #define FCF_ACCESS        0x00ff /* Filled with FileAccessFlags */
```

■ FileDateAndTime

```
typedef DWordFlags FileDateAndTime;
    #define FDAT_HOUR               0xf8000000
    #define FDAT_MINUTE             0x07e00000
    #define FDAT_2SECOND            0x001f0000
    #define FDAT_YEAR               0x0000fe00
    #define FDAT_MONTH              0x000001e0
    #define FDAT_DAY                0x0000001f
    #define FDAT_HOUR_OFFSET      27
    #define FDAT_MINUTE_OFFSET    21
    #define FDAT_2SECOND_OFFSET   16
    #define FDAT_YEAR_OFFSET      9
    #define FDAT_MONTH_OFFSET     5
    #define FDAT_DAY_OFFSET       0
    #define FDAT_BASE_YEAR        1980
```

Every GEOS file has two date and time stamps. One of them records the time the file was created, and one records the time the file was last modified. These stamps are recorded with the file's extended attributes; they are labeled FEA_CREATION and FEA_MODIFICATION, respectively. Non-GEOS files have a single date/time stamp, which records the time the file was last modified.

The date/time stamps are stored in a 32-bit bitfield. This field contains entries for the year, month, day, hour, minute, and second. Each field is identified by a mask and an offset. To access a field, simply clear all bits except those in the mask, then shift the bits to the right by the number of the offset. (Macros are provided to do this; they are described below.) **FileDateAndTime** contains the following fields, identified by their masks:

FDAT_YEAR
This field records the year, counting from a base year of 1980. (The constant FDAT_BASE_YEAR is defined as 1980.) This field is at an offset of FDAT_YEAR_OFFSET bits from the low end of the value.

FDAT_MONTH
This field records the month as an integer, with January being one. It is located at an offset of FDAT_MONTH_OFFSET.

FDAT_DAY
This field records the day of the month. It is located at an offset of FDAT_DAY_OFFSET.

FDAT_HOUR
This field records the hour on a 24-hour clock, with zero being the hour after midnight. It is located at an offset of FDAT_HOUR_OFFSET.

FDAT_MINUTE
This field records the minute. It is located at an offset of FDAT_MINUTE_OFFSET.

Routines

FDAT_2SECOND

This field records the second, divided by two; that is, a field value of 15 indicates the 30th second. (It is represented this way to let the second fit into 5 bits, thus letting the entire value fit into 32 bits.) It is located at an offset of FDAT_2SECOND_OFFSET.

Macros are provided to extract values from each of the fields of a **FileDateAndTime** structure. The macros are listed below:

```
byte FDATExtractYear( /* returns year field, counted from 1980*/
        FileDateAndTime fdat);
word FDATExtractYearAD( /* returns year field + base year */
        FileDateAndTime fdat);
byte FDATExtractMonth( /* returns month field (1 = January, etc.) */
        FileDateAndTime fdat);
byte FDATExtractDay( /* returns day field */
        FileDateAndTime fdat);
byte FDATExtractHour( /* returns hour field */
        FileDateAndTime fdat);
byte FDATExtractMinute( /* returns minute field */
        FileDateAndTime fdat);
byte FDATExtract2Second( /* returns 2Second field */
        FileDateAndTime fdat);
byte FDATExtractSecond( /* returns number of seconds (2 * 2Second) */
        FileDateAndTime fdat);
```

Include: file.h

■ FileDesktopInfo

```
typedef char FileDesktopInfo[FILE_DESKTOP_INFO_SIZE];
```

■ FileDirID

```
typedef dword FileDirID;
```

■ FileFileID

```
typedef dword FileFileID;
```

■ FileExclude

```
typedef ByteEnum FileExclude;
    #define FE_EXCLUSIVE      1
    #define FE_DENY_WRITE     2
    #define FE_DENY_READ      3
    #define FE_NONE           4
```

■ FileExtAttrDesc

```
typedef struct {
    FileExtendedAttribute     FEAD_attr;/* Attribute to get or set */
    void            *FEAD_value;        /* Pointer to buffer/new value */
    word            FEAD_size;          /* length of buffer/new value */
    chr             *FEAD_name;         /* If FEAD_attr == FEA_CUSTOM,
                                         * this points to null-
                                         * terminated ASCII string with
                                         * attribute's name; otherwise,
                                         * this is ignored. */
} FileExtendedAttrDesc;
```

The routines to get and set extended attributes can be passed the attribute FEA_MULTIPLE. In this case, they will also be passed the address of an array of **FileExtAttrDesc** structures and the number of elements of the array. They will go through the array and read or write the appropriate information.

FileEnum() can also be passed arrays of **FileExtAttrDesc** structures. In this case, the number of elements in the array is not passed. Instead, each array ends with a **FileExtAttrDesc** with a *FEAD_attr* field set to FEA_END_OF_LIST.

See Also: FileExtendedAttribute

Include: file.h

■ FileExtendedAttribute

```
typedef enum /* word */ {
    FEA_MODIFICATION,
    FEA_FILE_ATTR,
    FEA_SIZE,
    FEA_FILE_TYPE,
    FEA_FLAGS,
    FEA_RELEASE,
    FEA_PROTOCOL,
    FEA_TOKEN,
    FEA_CREATOR,
    FEA_USER_NOTES,
    FEA_NOTICE,
    FEA_CREATION,
    FEA_PASSWORD,
    FEA_CUSTOM,
    FEA_NAME,
    FEA_GEODE_ATTR,
    FEA_PATH_INFO,
    FEA_FILE_ID,
    FEA_DESKTOP_INFO,
    FEA_DRIVE_STATUS,
    FEA_DOS_NAME,
    FEA_OWNER,
    FEA_RIGHTS,
    FEA_MULTIPLE = 0xfffe,
    FEA_END_OF_LIST = 0xffff,
} FileExtendedAttribute;
```

Every GEOS file has a set of extended attributes. These attributes can be recovered with **FileGetPathExtAttributes()** or **FileGetHandleExtAttributes()**. You can also use **FileEnum()** to search a directory for files with specified extended attributes.

The above extended attributes have been implemented. More may be added with future releases of GEOS. The attributes are discussed at length in Section 17.5.3 of the Concepts book.

See Also: FileExtAttrDesc

Include: file.h

■ FileHandle

```
typedef Handle FileHandle;
```

■ FileLongName

```
typedef char FileLongName[FILE_LONGNAME_BUFFER_SIZE];
```

■ FileOwnerName

```
typedef char FileOwnerName[FILE_OWNER_NAME_SIZE];
```

■ FilePassword

```
typedef char FilePassword[FILE_PASSWORD_SIZE];
```

■ FilePosMode

```
typedef ByteEnum FilePosMode;
    #define FILE_POS_START    0
    #define FILE_POS_RELATIVE 1
    #define FILE_POS_END      2
```

■ FileUserNotes

```
typedef char FileUserNotes[GFH_USER_NOTES_BUFFER_SIZE];
```

■ FindNoteHeader

```
typedef struct {
    word    FNH_count;         /* The number of matching notes we’ve found */
} FindNoteHeader;
```

■ FloatExponent

```
typedef WordFlags FloatExponent;
    #define FE_SIGN         0x8000
    #define FE_EXPONENT     0x7fff
```

■ FloatNum

```
typedef struct {
            word                F_mantissa_wd0;
            word                F_mantissa_wd1;
            word                F_mantissa_wd2;
            word                F_mantissa_wd3;
            FloatExponent       F_exponent;
} FloatNum;
```

■ FontAttrs

```
typedef ByteFlags FontAttrs;
    #define FA_FIXED_WIDTH    0x40
    #define FA_ORIENT         0x20
    #define FA_OUTLINE        0x10
    #define FA_FAMILY         0x0f
    #define FA_FAMILY_OFFSET  0
```

Include: font.h

■ FontEnumFlags

```
typedef ByteFlags FontEnumFlags;
    #define FEF_ALPHABETIZE   0x80   /* Alphabetize returned list of fonts */
    #define FEF_FIXED_WIDTH   0x20   /* Return only fixed-width fonts */
    #define FEF_FAMILY        0x10
    #define FEF_STRING        0x08
    #define FEF_DOWNCASE      0x04   /* Returned font names will be lowercase */
    #define FEF_BITMAPS       0x02   /* Interested in bitmap fonts */
    #define FEF_OUTLINES      0x01   /* Interested in outline fonts */
```

Include: font.h

Routines

■ FontEnumStruct

```
typedef struct {
     FontIDs     FES_ID;
     char        FES_name[FID_NAME_LEN];
} FontEnumStruct;
```

Include: font.h

■ FontFamily

```
typedef byte FontFamily;
    #define FF_NON_PORTABLE       0x0007
    #define FF_SPECIAL            0x0006
    #define FF_MONO               0x0005
    #define FF_SYMBOL             0x0004
    #define FF_ORNAMENT           0x0003
    #define FF_SCRIPT             0x0002
    #define FF_SANS_SERIF         0x0001
    #define FF_SERIF              0x0000
```

Include: fontID.h

■ FontGroup

```
typedef enum /* word */ {
    #define FG_NON_PORTABLE       0x0e00
    #define FG_SPECIAL            0x0c00
    #define FG_MONO               0x0a00
    #define FG_SYMBOL             0x0800
    #define FG_ORNAMENT           0x0600
    #define FG_SCRIPT             0x0400
    #define FG_SANS_SERIF         0x0200
    #define FG_SERIF              0x0000
} FontGroup;
```

Include: fontID.h

■ FontIDRecord

```
typedef WordFlags FontIDRecord;
    #define FIDR_maker        0xf000
    #define FIDR_ID           0x0fff
    #define FIDR_maker_OFFSET     12
    #define FIDR_ID_OFFSET         0
```

Include: font.h

■ FontID

```
typedef word FontID;
    #define FID_PRINTER_20CPI                     0xfa05
    #define FID_PRINTER_17CPI                     0xfa04
    #define FID_PRINTER_16CPI                     0xfa03
    #define FID_PRINTER_15CPI                     0xfa02
    #define FID_PRINTER_12CPI                     0xfa01
    #define FID_PRINTER_10CPI                     0xfa00
    #define FID_PRINTER_PROP_SANS                 0xf200
    #define FID_PRINTER_PROP_SERIF                0xf000
    #define FID_BITSTREAM_LETTER_GOTHIC           0x3a03
    #define FID_PS_LETTER_GOTHIC                  0x2a03
    #define FID_DTC_LETTER_GOTHIC                 0x1a03
    #define FID_BITSTREAM_PRESTIGE_ELITE          0x3a02
    #define FID_PS_PRESTIGE_ELITE                 0x2a02
    #define FID_DTC_PRESTIGE_ELITE                0x1a02
    #define FID_BITSTREAM_AMERICAN_TYPEWRITER     0x3a01
    #define FID_PS_AMERICAN_TYPEWRITER            0x2a01
    #define FID_DTC_AMERICAN_TYPEWRITER           0x1a01
    #define FID_BITSTREAM_URW_MONO                0x3a00
    #define FID_PS_COURIER                        0x2a00
    #define FID_DTC_URW_MONO                      0x1a00
    #define FID_BITSTREAM_FUN_DINGBATS            0x380d
    #define FID_PS_FUN_DINGBATS                   0x280d
    #define FID_DTC_FUN_DINGBATS                  0x180d
    #define FID_BITSTREAM_CHEQ                    0x380c
    #define FID_PS_CHEQ                           0x280c
    #define FID_DTC_CHEQ                          0x180c
    #define FID_BITSTREAM_BUNDESBAHN_PI_3         0x380b
    #define FID_PS_BUNDESBAHN_PI_3                0x280b
    #define FID_DTC_BUNDESBAHN_PI_3               0x180b
    #define FID_BITSTREAM_BUNDESBAHN_PI_2         0x380a
    #define FID_PS_BUNDESBAHN_PI_2                0x280a
    #define FID_DTC_BUNDESBAHN_PI_2               0x180a
    #define FID_BITSTREAM_BUNDESBAHN_PI_1         0x3809
    #define FID_PS_BUNDESBAHN_PI_1                0x2809
    #define FID_DTC_BUNDESBAHN_PI_1               0x1809
    #define FID_BITSTREAM_U_GREEK_MATH_PI         0x3808
    #define FID_PS_U_GREEK_MATH_PI                0x2808
    #define FID_DTC_U_GREEK_MATH_PI               0x1808
    #define FID_BITSTREAM_U_NEWS_COMM_PI          0x3807
    #define FID_PS_U_NEWS_COMM_PI                 0x2807
    #define FID_DTC_U_NEWS_COMM_PI                0x1807
    #define FID_BITSTREAM_ACE_I                   0x3806
    #define FID_PS_ACE_I                          0x2806
    #define FID_DTC_ACE_I                         0x1806
    #define FID_BITSTREAM_SONATA                  0x3805
    #define FID_PS_SONATA                         0x2805
    #define FID_DTC_SONATA                        0x1805
```

Routines

```
#define FID_BITSTREAM_CARTA                  0x3804
#define FID_PS_CARTA                         0x2804
#define FID_DTC_CARTA                        0x1804
#define FID_BITSTREAM_MICR                   0x3803
#define FID_PS_MICR                          0x2803
#define FID_DTC_MICR                         0x1803
#define FID_BITSTREAM_ZAPF_DINGBATS          0x3802
#define FID_PS_ZAPF_DINGBATS                 0x2802
#define FID_DTC_ZAPF_DINGBATS                0x1802
#define FID_BITSTREAM_DINGBATS               0x3801
#define FID_PS_DINGBATS                      0x2801
#define FID_DTC_DINGBATS                     0x1801
#define FID_BITSTREAM_URW_SYMBOLPS           0x3800
#define FID_PS_SYMBOL                        0x2800
#define FID_DTC_URW_SYMBOLPS                 0x1800
#define FID_BITSTREAM_JUNIPER                0x367f
#define FID_PS_JUNIPER                       0x267f
#define FID_DTC_JUNIPER                      0x167f
#define FID_BITSTREAM_COTTONWOOD             0x367e
#define FID_PS_COTTONWOOD                    0x267e
#define FID_DTC_COTTONWOOD                   0x167e
#define FID_BITSTREAM_BANCO                  0x367d
#define FID_PS_BANCO                         0x267d
#define FID_DTC_BANCO                        0x167d
#define FID_BITSTREAM_ARCADIA                0x367c
#define FID_PS_ARCADIA                       0x267c
#define FID_DTC_ARCADIA                      0x167c
#define FID_BITSTREAM_ZIPPER                 0x367b
#define FID_PS_ZIPPER                        0x267b
#define FID_DTC_ZIPPER                       0x167b
#define FID_BITSTREAM_WEIFZ_RUNDGOTIFCH      0x367a
#define FID_PS_WEIFZ_RUNDGOTIFCH             0x267a
#define FID_DTC_WEIFZ_RUNDGOTIFCH            0x167a
#define FID_BITSTREAM_WASHINGTON             0x3679
#define FID_PS_WASHINGTON                    0x2679
#define FID_DTC_WASHINGTON                   0x1679
#define FID_BITSTREAM_VICTORIAN              0x3678
#define FID_PS_VICTORIAN                     0x2678
#define FID_DTC_VICTORIAN                    0x1678
#define FID_BITSTREAM_VEGAS                  0x3677
#define FID_PS_VEGAS                         0x2677
#define FID_DTC_VEGAS                        0x1677
#define FID_BITSTREAM_VARIO                  0x3676
#define FID_PS_VARIO                         0x2676
#define FID_DTC_VARIO                        0x1676
#define FID_BITSTREAM_VAG_RUNDSCHRIFT        0x3675
#define FID_PS_VAG_RUNDSCHRIFT               0x2675
#define FID_DTC_VAG_RUNDSCHRIFT              0x1675
#define FID_BITSTREAM_TRAJANUS               0x3674
#define FID_PS_TRAJANUS                      0x2674
```

```
#define FID_DTC_TRAJANUS                      0x1674
#define FID_BITSTREAM_TITUS                   0x3673
#define FID_PS_TITUS                          0x2673
#define FID_DTC_TITUS                         0x1673
#define FID_BITSTREAM_TIME_SCRIPT             0x3672
#define FID_PS_TIME_SCRIPT                    0x2672
#define FID_DTC_TIME_SCRIPT                   0x1672
#define FID_BITSTREAM_THUNDERBIRD             0x3671
#define FID_PS_THUNDERBIRD                    0x2671
#define FID_DTC_THUNDERBIRD                   0x1671
#define FID_BITSTREAM_THOROWGOOD              0x3670
#define FID_PS_THOROWGOOD                     0x2670
#define FID_DTC_THOROWGOOD                    0x1670
#define FID_BITSTREAM_TARRAGON                0x366f
#define FID_PS_TARRAGON                       0x266f
#define FID_DTC_TARRAGON                      0x166f
#define FID_BITSTREAM_TANGO                   0x366e
#define FID_PS_TANGO                          0x266e
#define FID_DTC_TANGO                         0x166e
#define FID_BITSTREAM_SYNCHRO                 0x366d
#define FID_PS_SYNCHRO                        0x266d
#define FID_DTC_SYNCHRO                       0x166d
#define FID_BITSTREAM_SUPERSTAR               0x366c
#define FID_PS_SUPERSTAR                      0x266c
#define FID_DTC_SUPERSTAR                     0x166c
#define FID_BITSTREAM_STOP                    0x366b
#define FID_PS_STOP                           0x266b
#define FID_DTC_STOP                          0x166b
#define FID_BITSTREAM_STILLA_CAPS             0x366a
#define FID_PS_STILLA_CAPS                    0x266a
#define FID_DTC_STILLA_CAPS                   0x166a
#define FID_BITSTREAM_STILLA                  0x3669
#define FID_PS_STILLA                         0x2669
#define FID_DTC_STILLA                        0x1669
#define FID_BITSTREAM_STENTOR                 0x3668
#define FID_PS_STENTOR                        0x2668
#define FID_DTC_STENTOR                       0x1668
#define FID_BITSTREAM_SQUIRE                  0x3667
#define FID_PS_SQUIRE                         0x2667
#define FID_DTC_SQUIRE                        0x1667
#define FID_BITSTREAM_SPRINGFIELD             0x3666
#define FID_PS_SPRINGFIELD                    0x2666
#define FID_DTC_SPRINGFIELD                   0x1666
#define FID_BITSTREAM_SLIPSTREAM              0x3665
#define FID_PS_SLIPSTREAM                     0x2665
#define FID_DTC_SLIPSTREAM                    0x1665
#define FID_BITSTREAM_SINALOA                 0x3664
#define FID_PS_SINALOA                        0x2664
#define FID_DTC_SINALOA                       0x1664
#define FID_BITSTREAM_SHELLEY                 0x3663
```

Routines

```
#define FID_PS_SHELLEY                         0x2663
#define FID_DTC_SHELLEY                        0x1663
#define FID_BITSTREAM_SERPENTINE               0x3662
#define FID_PS_SERPENTINE                      0x2662
#define FID_DTC_SERPENTINE                     0x1662
#define FID_BITSTREAM_RUBBER_STAMP             0x3661
#define FID_PS_RUBBER_STAMP                    0x2661
#define FID_DTC_RUBBER_STAMP                   0x1661
#define FID_BITSTREAM_ROMIC                    0x3660
#define FID_PS_ROMIC                           0x2660
#define FID_DTC_ROMIC                          0x1660
#define FID_BITSTREAM_RIALTO                   0x365f
#define FID_PS_RIALTO                          0x265f
#define FID_DTC_RIALTO                         0x165f
#define FID_BITSTREAM_REVUE                    0x365e
#define FID_PS_REVUE                           0x265e
#define FID_DTC_REVUE                          0x165e
#define FID_BITSTREAM_QUENTIN                  0x365d
#define FID_PS_QUENTIN                         0x265d
#define FID_DTC_QUENTIN                        0x165d
#define FID_BITSTREAM_PRO_ARTE                 0x365c
#define FID_PS_PRO_ARTE                        0x265c
#define FID_DTC_PRO_ARTE                       0x165c
#define FID_BITSTREAM_PRINCETOWN               0x365b
#define FID_PS_PRINCETOWN                      0x265b
#define FID_DTC_PRINCETOWN                     0x165b
#define FID_BITSTREAM_PRESIDENT                0x365a
#define FID_PS_PRESIDENT                       0x265a
#define FID_DTC_PRESIDENT                      0x165a
#define FID_BITSTREAM_PREMIER                  0x3659
#define FID_PS_PREMIER                         0x2659
#define FID_DTC_PREMIER                        0x1659
#define FID_BITSTREAM_POST_ANTIQUA             0x3658
#define FID_PS_POST_ANTIQUA                    0x2658
#define FID_DTC_POST_ANTIQUA                   0x1658
#define FID_BITSTREAM_PLAZA                    0x3657
#define FID_PS_PLAZA                           0x2657
#define FID_DTC_PLAZA                          0x1657
#define FID_BITSTREAM_PLAYBILL                 0x3656
#define FID_PS_PLAYBILL                        0x2656
#define FID_DTC_PLAYBILL                       0x1656
#define FID_BITSTREAM_PICCADILLY               0x3655
#define FID_PS_PICCADILLY                      0x2655
#define FID_DTC_PICCADILLY                     0x1655
#define FID_BITSTREAM_PEIGNOT                  0x3654
#define FID_PS_PEIGNOT                         0x2654
#define FID_DTC_PEIGNOT                        0x1654
#define FID_BITSTREAM_PAPYRUS                  0x3653
#define FID_PS_PAPYRUS                         0x2653
#define FID_DTC_PAPYRUS                        0x1653
```

```
#define FID_BITSTREAM_PADDINGTION          0x3652
#define FID_PS_PADDINGTION                 0x2652
#define FID_DTC_PADDINGTION                0x1652
#define FID_BITSTREAM_OKAY                 0x3651
#define FID_PS_OKAY                        0x2651
#define FID_DTC_OKAY                       0x1651
#define FID_BITSTREAM_ODIN                 0x3650
#define FID_PS_ODIN                        0x2650
#define FID_DTC_ODIN                       0x1650
#define FID_BITSTREAM_OCTOPUSS             0x364f
#define FID_PS_OCTOPUSS                    0x264f
#define FID_DTC_OCTOPUSS                   0x164f
#define FID_BITSTREAM_MOTTER_FEMINA        0x364e
#define FID_PS_MOTTER_FEMINA               0x264e
#define FID_DTC_MOTTER_FEMINA              0x164e
#define FID_BITSTREAM_MICROGRAMMA          0x364d
#define FID_PS_MICROGRAMMA                 0x264d
#define FID_DTC_MICROGRAMMA                0x164d
#define FID_BITSTREAM_MACHINE              0x364c
#define FID_PS_MACHINE                     0x264c
#define FID_DTC_MACHINE                    0x164c
#define FID_BITSTREAM_LINOTEXT             0x364b
#define FID_PS_LINOTEXT                    0x264b
#define FID_DTC_LINOTEXT                   0x164b
#define FID_BITSTREAM_LIBERTY              0x364a
#define FID_PS_LIBERTY                     0x264a
#define FID_DTC_LIBERTY                    0x164a
#define FID_BITSTREAM_LAZYBONES            0x3649
#define FID_PS_LAZYBONES                   0x2649
#define FID_DTC_LAZYBONES                  0x1649
#define FID_BITSTREAM_LATIN_WIDE           0x3648
#define FID_PS_LATIN_WIDE                  0x2648
#define FID_DTC_LATIN_WIDE                 0x1648
#define FID_BITSTREAM_KNIGHTSBRIDGE        0x3647
#define FID_PS_KNIGHTSBRIDGE               0x2647
#define FID_DTC_KNIGHTSBRIDGE              0x1647
#define FID_BITSTREAM_KAPITELLIA           0x3646
#define FID_PS_KAPITELLIA                  0x2646
#define FID_DTC_KAPITELLIA                 0x1646
#define FID_BITSTREAM_KALLIGRAPHIA         0x3645
#define FID_PS_KALLIGRAPHIA                0x2645
#define FID_DTC_KALLIGRAPHIA               0x1645
#define FID_BITSTREAM_ICE_AGE              0x3644
#define FID_PS_ICE_AGE                     0x2644
#define FID_DTC_ICE_AGE                    0x1644
#define FID_BITSTREAM_ICONE                0x3643
#define FID_PS_ICONE                       0x2643
#define FID_DTC_ICONE                      0x1643
#define FID_BITSTREAM_HORNDON              0x3642
#define FID_PS_HORNDON                     0x2642
```

Routines

```
#define FID_DTC_HORNDON                   0x1642
#define FID_BITSTREAM_HORATIO             0x3641
#define FID_PS_HORATIO                    0x2641
#define FID_DTC_HORATIO                   0x1641
#define FID_BITSTREAM_HIGHLIGHT           0x3640
#define FID_PS_HIGHLIGHT                  0x2640
#define FID_DTC_HIGHLIGHT                 0x1640
#define FID_BITSTREAM_HADFIELD            0x363f
#define FID_PS_HADFIELD                   0x263f
#define FID_DTC_HADFIELD                  0x163f
#define FID_BITSTREAM_GLASER_STENCIL      0x363e
#define FID_PS_GLASER_STENCIL             0x263e
#define FID_DTC_GLASER_STENCIL            0x163e
#define FID_BITSTREAM_GILL_KAYO           0x363d
#define FID_PS_GILL_KAYO                  0x263d
#define FID_DTC_GILL_KAYO                 0x163d
#define FID_BITSTREAM_GALADRIEL           0x363c
#define FID_PS_GALADRIEL                  0x263c
#define FID_DTC_GALADRIEL                 0x163c
#define FID_BITSTREAM_FUTURA_DISPLAY      0x363b
#define FID_PS_FUTURA_DISPLAY             0x263b
#define FID_DTC_FUTURA_DISPLAY            0x163b
#define FID_BITSTREAM_FUTURA_C_BLACK      0x363a
#define FID_PS_FUTURA_C_BLACK             0x263a
#define FID_DTC_FUTURA_C_BLACK            0x163a
#define FID_BITSTREAM_FRANKFURTER         0x3639
#define FID_PS_FRANKFURTER                0x2639
#define FID_DTC_FRANKFURTER               0x1639
#define FID_BITSTREAM_FLORA               0x3638
#define FID_PS_FLORA                      0x2638
#define FID_DTC_FLORA                     0x1638
#define FID_BITSTREAM_FLANGE              0x3637
#define FID_PS_FLANGE                     0x2637
#define FID_DTC_FLANGE                    0x1637
#define FID_BITSTREAM_FLASH               0x3636
#define FID_PS_FLASH                      0x2636
#define FID_DTC_FLASH                     0x1636
#define FID_BITSTREAM_FLAMENCO            0x3635
#define FID_PS_FLAMENCO                   0x2635
#define FID_DTC_FLAMENCO                  0x1635
#define FID_BITSTREAM_FETTE_GOTILCH       0x3634
#define FID_PS_FETTE_GOTILCH              0x2634
#define FID_DTC_FETTE_GOTILCH             0x1634
#define FID_BITSTREAM_FETTE_FRAKTUR       0x3633
#define FID_PS_FETTE_FRAKTUR              0x2633
#define FID_DTC_FETTE_FRAKTUR             0x1633
#define FID_BITSTREAM_ENVIRO              0x3632
#define FID_PS_ENVIRO                     0x2632
#define FID_DTC_ENVIRO                    0x1632
#define FID_BITSTREAM_EINHORN             0x3631
```

```
#define FID_PS_EINHORN                          0x2631
#define FID_DTC_EINHORN                         0x1631
#define FID_BITSTREAM_ECKMANN                   0x3630
#define FID_PS_ECKMANN                          0x2630
#define FID_DTC_ECKMANN                         0x1630
#define FID_BITSTREAM_DYNAMO                    0x362f
#define FID_PS_DYNAMO                           0x262f
#define FID_DTC_DYNAMO                          0x162f
#define FID_BITSTREAM_DOM_CASUAL                0x362e
#define FID_PS_DOM_CASUAL                       0x262e
#define FID_DTC_DOM_CASUAL                      0x162e
#define FID_BITSTREAM_DAVIDA                    0x362d
#define FID_PS_DAVIDA                           0x262d
#define FID_DTC_DAVIDA                          0x162d
#define FID_BITSTREAM_CROISSANT                 0x362c
#define FID_PS_CROISSANT                        0x262c
#define FID_DTC_CROISSANT                       0x162c
#define FID_BITSTREAM_CRILLEE                   0x362b
#define FID_PS_CRILLEE                          0x262b
#define FID_DTC_CRILLEE                         0x162b
#define FID_BITSTREAM_COUNTDOWN                 0x362a
#define FID_PS_COUNTDOWN                        0x262a
#define FID_DTC_COUNTDOWN                       0x162a
#define FID_BITSTREAM_CORTEZ                    0x3629
#define FID_PS_CORTEZ                           0x2629
#define FID_DTC_CORTEZ                          0x1629
#define FID_BITSTREAM_CONFERENCE                0x3628
#define FID_PS_CONFERENCE                       0x2628
#define FID_DTC_CONFERENCE                      0x1628
#define FID_BITSTREAM_COMPANY                   0x3627
#define FID_PS_COMPANY                          0x2627
#define FID_DTC_COMPANY                         0x1627
#define FID_BITSTREAM_COLUMNA_SOLID             0x3626
#define FID_PS_COLUMNA_SOLID                    0x2626
#define FID_DTC_COLUMNA_SOLID                   0x1626
#define FID_BITSTREAM_CITY                      0x3625
#define FID_PS_CITY                             0x2625
#define FID_DTC_CITY                            0x1625
#define FID_BITSTREAM_CIRKULUS                  0x3624
#define FID_PS_CIRKULUS                         0x2624
#define FID_DTC_CIRKULUS                        0x1624
#define FID_BITSTREAM_CHURCHWARD_BRUSH          0x3623
#define FID_PS_CHURCHWARD_BRUSH                 0x2623
#define FID_DTC_CHURCHWARD_BRUSH                0x1623
#define FID_BITSTREAM_CHROMIUM_ONE              0x3622
#define FID_PS_CHROMIUM_ONE                     0x2622
#define FID_DTC_CHROMIUM_ONE                    0x1622
#define FID_BITSTREAM_CHOC                      0x3621
#define FID_PS_CHOC                             0x2621
#define FID_DTC_CHOC                            0x1621
```

Routines

```
#define FID_BITSTREAM_CHISEL                  0x3620
#define FID_PS_CHISEL                         0x2620
#define FID_DTC_CHISEL                        0x1620
#define FID_BITSTREAM_CHESTERFIELD            0x361f
#define FID_PS_CHESTERFIELD                   0x261f
#define FID_DTC_CHESTERFIELD                  0x161f
#define FID_BITSTREAM_CAROUSEL                0x361e
#define FID_PS_CAROUSEL                       0x261e
#define FID_DTC_CAROUSEL                      0x161e
#define FID_BITSTREAM_CAMELLIA                0x361d
#define FID_PS_CAMELLIA                       0x261d
#define FID_DTC_CAMELLIA                      0x161d
#define FID_BITSTREAM_CABARET                 0x361c
#define FID_PS_CABARET                        0x261c
#define FID_DTC_CABARET                       0x161c
#define FID_BITSTREAM_BUXOM                   0x361b
#define FID_PS_BUXOM                          0x261b
#define FID_DTC_BUXOM                         0x161b
#define FID_BITSTREAM_BUSTER                  0x361a
#define FID_PS_BUSTER                         0x261a
#define FID_DTC_BUSTER                        0x161a
#define FID_BITSTREAM_BOTTLENECK              0x3619
#define FID_PS_BOTTLENECK                     0x2619
#define FID_DTC_BOTTLENECK                    0x1619
#define FID_BITSTREAM_BLOCK                   0x3618
#define FID_PS_BLOCK                          0x2618
#define FID_DTC_BLOCK                         0x1618
#define FID_BITSTREAM_BINNER                  0x3617
#define FID_PS_BINNER                         0x2617
#define FID_DTC_BINNER                        0x1617
#define FID_BITSTREAM_BERNHARD_ANTIQUE        0x3616
#define FID_PS_BERNHARD_ANTIQUE               0x2616
#define FID_DTC_BERNHARD_ANTIQUE              0x1616
#define FID_BITSTREAM_BELSHAW                 0x3615
#define FID_PS_BELSHAW                        0x2615
#define FID_DTC_BELSHAW                       0x1615
#define FID_BITSTREAM_BARCELONA               0x3614
#define FID_PS_BARCELONA                      0x2614
#define FID_DTC_BARCELONA                     0x1614
#define FID_BITSTREAM_BAUHAUS                 0x3613
#define FID_PS_BAUHAUS                        0x2613
#define FID_DTC_BAUHAUS                       0x1613
#define FID_BITSTREAM_AUGUSTEA_OPEN           0x3612
#define FID_PS_AUGUSTEA_OPEN                  0x2612
#define FID_DTC_AUGUSTEA_OPEN                 0x1612
#define FID_BITSTREAM_AMERICAN_UNCIAL         0x3611
#define FID_PS_AMERICAN_UNCIAL                0x2611
#define FID_DTC_AMERICAN_UNCIAL               0x1611
#define FID_BITSTREAM_ULTE_SCHWABACHER        0x3610
#define FID_PS_ULTE_SCHWABACHER               0x2610
```

```
#define FID_DTC_ULTE_SCHWABACHER              0x1610
#define FID_BITSTREAM_ARNOLD_BOCKLIN          0x360f
#define FID_PS_ARNOLD_BOCKLIN                 0x260f
#define FID_DTC_ARNOLD_BOCKLIN                0x160f
#define FID_BITSTREAM_ALGERIAN                0x360e
#define FID_PS_ALGERIAN                       0x260e
#define FID_DTC_ALGERIAN                      0x160e
#define FID_BITSTREAM_PUMP                    0x360d
#define FID_PS_PUMP                           0x260d
#define FID_DTC_PUMP                          0x160d
#define FID_BITSTREAM_MARIAGE                 0x360c
#define FID_PS_MARIAGE                        0x260c
#define FID_DTC_MARIAGE                       0x160c
#define FID_BITSTREAM_OLD_TOWN                0x360b
#define FID_PS_OLD_TOWN                       0x260b
#define FID_DTC_OLD_TOWN                      0x160b
#define FID_BITSTREAM_HOBO                    0x360a
#define FID_PS_HOBO                           0x260a
#define FID_DTC_HOBO                          0x160a
#define FID_BITSTREAM_GOUDY_HEAVYFACE         0x3609
#define FID_PS_GOUDY_HEAVYFACE                0x2609
#define FID_DTC_GOUDY_HEAVYFACE               0x1609
#define FID_BITSTREAM_DATA_70                 0x3608
#define FID_PS_DATA_70                        0x2608
#define FID_DTC_DATA_70                       0x1608
#define FID_BITSTREAM_LCD                     0x3607
#define FID_PS_LCD                            0x2607
#define FID_DTC_LCD                           0x1607
#define FID_BITSTREAM_BALLOON                 0x3606
#define FID_PS_BALLOON                        0x2606
#define FID_DTC_BALLOON                       0x1606
#define FID_BITSTREAM_BLIPPO_C_BLACK          0x3605
#define FID_PS_BLIPPO_C_BLACK                 0x2605
#define FID_DTC_BLIPPO_C_BLACK                0x1605
#define FID_BITSTREAM_COOPER_C_BLACK          0x3604
#define FID_PS_COOPER_C_BLACK                 0x2604
#define FID_DTC_COOPER_C_BLACK                0x1604
#define FID_BITSTREAM_COPPERPLATE             0x3603
#define FID_PS_COPPERPLATE                    0x2603
#define FID_DTC_COPPERPLATE                   0x1603
#define FID_BITSTREAM_STENCIL                 0x3602
#define FID_PS_STENCIL                        0x2602
#define FID_DTC_STENCIL                       0x1602
#define FID_BITSTREAM_OLD_ENGLISH             0x3601
#define FID_PS_OLD_ENGLISH                    0x2601
#define FID_DTC_OLD_ENGLISH                   0x1601
#define FID_BITSTREAM_BROADWAY                0x3600
#define FID_PS_BROADWAY                       0x2600
#define FID_DTC_BROADWAY                      0x1600
#define FID_BITSTREAM_NUPITAL_SCRIPT          0x3430
```

Routines

```
#define FID_PS_NUPITAL_SCRIPT                 0x2430
#define FID_DTC_NUPITAL_SCRIPT                0x1430
#define FID_BITSTREAM_MEDICI_SCRIPT           0x342f
#define FID_PS_MEDICI_SCRIPT                  0x242f
#define FID_DTC_MEDICI_SCRIPT                 0x142f
#define FID_BITSTREAM_CHARME                  0x342e
#define FID_PS_CHARME                         0x242e
#define FID_DTC_CHARME                        0x142e
#define FID_BITSTREAM_CASCADE_SCRIPT          0x342d
#define FID_PS_CASCADE_SCRIPT                 0x242d
#define FID_DTC_CASCADE_SCRIPT                0x142d
#define FID_BITSTREAM_LITHOS                  0x342c
#define FID_PS_LITHOS                         0x242c
#define FID_DTC_LITHOS                        0x142c
#define FID_BITSTREAM_TEKTON                  0x342b
#define FID_PS_TEKTON                         0x242b
#define FID_DTC_TEKTON                        0x142b
#define FID_BITSTREAM_VLADIMIR_SCRIPT         0x342a
#define FID_PS_VLADIMIR_SCRIPT                0x242a
#define FID_DTC_VLADIMIR_SCRIPT               0x142a
#define FID_BITSTREAM_VAN_DIJK                0x3429
#define FID_PS_VAN_DIJK                       0x2429
#define FID_DTC_VAN_DIJK                      0x1429
#define FID_BITSTREAM_SLOGAN                  0x3428
#define FID_PS_SLOGAN                         0x2428
#define FID_DTC_SLOGAN                        0x1428
#define FID_BITSTREAM_SHAMROCK                0x3427
#define FID_PS_SHAMROCK                       0x2427
#define FID_DTC_SHAMROCK                      0x1427
#define FID_BITSTREAM_ROMAN_SCRIPT            0x3426
#define FID_PS_ROMAN_SCRIPT                   0x2426
#define FID_DTC_ROMAN_SCRIPT                  0x1426
#define FID_BITSTREAM_RAGE                    0x3425
#define FID_PS_RAGE                           0x2425
#define FID_DTC_RAGE                          0x1425
#define FID_BITSTREAM_PRESENT_SCRIPT          0x3424
#define FID_PS_PRESENT_SCRIPT                 0x2424
#define FID_DTC_PRESENT_SCRIPT                0x1424
#define FID_BITSTREAM_PHYLLIS_INITIALS        0x3423
#define FID_PS_PHYLLIS_INITIALS               0x2423
#define FID_DTC_PHYLLIS_INITIALS              0x1423
#define FID_BITSTREAM_PHYLLIS                 0x3422
#define FID_PS_PHYLLIS                        0x2422
#define FID_DTC_PHYLLIS                       0x1422
#define FID_BITSTREAM_PEPITA                  0x3421
#define FID_PS_PEPITA                         0x2421
#define FID_DTC_PEPITA                        0x1421
#define FID_BITSTREAM_PENDRY_SCRIPT           0x3420
#define FID_PS_PENDRY_SCRIPT                  0x2420
#define FID_DTC_PENDRY_SCRIPT                 0x1420
```

```
#define FID_BITSTREAM_PALETTE                  0x341f
#define FID_PS_PALETTE                         0x241f
#define FID_DTC_PALETTE                        0x141f
#define FID_BITSTREAM_PALACE_SCRIPT            0x341e
#define FID_PS_PALACE_SCRIPT                   0x241e
#define FID_DTC_PALACE_SCRIPT                  0x141e
#define FID_BITSTREAM_NEVISON_CASUAL           0x341d
#define FID_PS_NEVISON_CASUAL                  0x241d
#define FID_DTC_NEVISON_CASUAL                 0x141d
#define FID_BITSTREAM_HILL                     0x341c
#define FID_PS_HILL                            0x241c
#define FID_DTC_HILL                           0x141c
#define FID_BITSTREAM_LINOSCRIPT               0x341b
#define FID_PS_LINOSCRIPT                      0x241b
#define FID_DTC_LINOSCRIPT                     0x141b
#define FID_BITSTREAM_LINDSAY                  0x341a
#define FID_PS_LINDSAY                         0x241a
#define FID_DTC_LINDSAY                        0x141a
#define FID_BITSTREAM_LE_GRIFFE                0x3419
#define FID_PS_LE_GRIFFE                       0x2419
#define FID_DTC_LE_GRIFFE                      0x1419
#define FID_BITSTREAM_KUNSTLERSCHREIBSCHRIFT   0x3418
#define FID_PS_KUNSTLERSCHREIBSCHRIFT          0x2418
#define FID_DTC_KUNSTLERSCHREIBSCHRIFT         0x1418
#define FID_BITSTREAM_JULIA_SCRIPT             0x3417
#define FID_PS_JULIA_SCRIPT                    0x2417
#define FID_DTC_JULIA_SCRIPT                   0x1417
#define FID_BITSTREAM_ISBELL                   0x3416
#define FID_PS_ISBELL                          0x2416
#define FID_DTC_ISBELL                         0x1416
#define FID_BITSTREAM_ISADORA                  0x3415
#define FID_PS_ISADORA                         0x2415
#define FID_DTC_ISADORA                        0x1415
#define FID_BITSTREAM_HOGARTH_SCRIPT           0x3414
#define FID_PS_HOGARTH_SCRIPT                  0x2414
#define FID_DTC_HOGARTH_SCRIPT                 0x1414
#define FID_BITSTREAM_HARLOW                   0x3413
#define FID_PS_HARLOW                          0x2413
#define FID_DTC_HARLOW                         0x1413
#define FID_BITSTREAM_GLASTONBURY              0x3412
#define FID_PS_GLASTONBURY                     0x2412
#define FID_DTC_GLASTONBURY                    0x1412
#define FID_BITSTREAM_GILLIES_GOTHIC           0x3411
#define FID_PS_GILLIES_GOTHIC                  0x2411
#define FID_DTC_GILLIES_GOTHIC                 0x1411
#define FID_BITSTREAM_FREESTYLE_SCRIPT         0x3410
#define FID_PS_FREESTYLE_SCRIPT                0x2410
#define FID_DTC_FREESTYLE_SCRIPT               0x1410
#define FID_BITSTREAM_ENGLISCHE_SCHREIBSCHRIFT0x340f
#define FID_PS_ENGLISCHE_SCHREIBSCHRIFT        0x240f
```

Routines

```
#define FID_DTC_ENGLISCHE_SCHREIBSCHRIFT      0x140f
#define FID_BITSTREAM_DEMIAN                  0x340e
#define FID_PS_DEMIAN                         0x240e
#define FID_DTC_DEMIAN                        0x140e
#define FID_BITSTREAM_CANDICE                 0x340d
#define FID_PS_CANDICE                        0x240d
#define FID_DTC_CANDICE                       0x140d
#define FID_BITSTREAM_BRONX                   0x340c
#define FID_PS_BRONX                          0x240c
#define FID_DTC_BRONX                         0x140x
#define FID_BITSTREAM_BRODY                   0x340b
#define FID_PS_BRODY                          0x240b
#define FID_DTC_BRODY                         0x140b
#define FID_BITSTREAM_BIBLE_SCRIPT            0x340a
#define FID_PS_BIBLE_SCRIPT                   0x240a
#define FID_DTC_BIBLE_SCRIPT                  0x140a
#define FID_BITSTREAM_ARISTON                 0x3409
#define FID_PS_ARISTON                        0x2409
#define FID_DTC_ARISTON                       0x1409
#define FID_BITSTREAM_ANGLIA                  0x3408
#define FID_PS_ANGLIA                         0x2408
#define FID_DTC_ANGLIA                        0x1408
#define FID_BITSTREAM_MISTRAL                 0x3407
#define FID_PS_MISTRAL                        0x2407
#define FID_DTC_MISTRAL                       0x1407
#define FID_BITSTREAM_BALMORAL                0x3406
#define FID_PS_BALMORAL                       0x2406
#define FID_DTC_BALMORAL                      0x1406
#define FID_BITSTREAM_COMMERCIAL_SCRIPT       0x3405
#define FID_PS_COMMERCIAL_SCRIPT              0x2405
#define FID_DTC_COMMERCIAL_SCRIPT             0x1405
#define FID_BITSTREAM_KAUFMANN                0x3404
#define FID_PS_KAUFMANN                       0x2404
#define FID_DTC_KAUFMANN                      0x1404
#define FID_BITSTREAM_PARK_AVENUE             0x3403
#define FID_PS_PARK_AVENUE                    0x2403
#define FID_DTC_PARK_AVENUE                   0x1403
#define FID_BITSTREAM_BRUSH_SCRIPT            0x3402
#define FID_PS_BRUSH_SCRIPT                   0x2402
#define FID_DTC_BRUSH_SCRIPT                  0x1402
#define FID_BITSTREAM_VIVALDI                 0x3401
#define FID_PS_VIVALDI                        0x2401
#define FID_DTC_VIVALDI                       0x1401
#define FID_BITSTREAM_ZAPF_CHANCERY           0x3400
#define FID_PS_ZAPF_CHANCERY                  0x2400
#define FID_DTC_ZAPF_CHANCERY                 0x1400
#define FID_BITSTREAM_AVANTE_GARDE_CONDENSED  0x323d
#define FID_PS_AVANTE_GARDE_CONDENSED         0x223d
#define FID_DTC_AVANTE_GARDE_CONDENSED        0x123d
#define FID_BITSTREAM_INSIGNIA                0x323c
```

```
#define FID_PS_INSIGNIA                        0x223c
#define FID_DTC_INSIGNIA                       0x123c
#define FID_BITSTREAM_INDUSTRIA                0x323b
#define FID_PS_INDUSTRIA                       0x223b
#define FID_DTC_INDUSTRIA                      0x123b
#define FID_BITSTREAM_DORIC_BOLD               0x323a
#define FID_PS_DORIC_BOLD                      0x223a
#define FID_DTC_DORIC_BOLD                     0x123a
#define FID_BITSTREAM_AKZINDENZ_GROTESK        0x3239
#define FID_PS_AKZINDENZ_GROTESK               0x2239
#define FID_DTC_AKZINDENZ_GROTESK              0x1239
#define FID_BITSTREAM_GROTESK                  0x3238
#define FID_PS_GROTESK                         0x2238
#define FID_DTC_GROTESK                        0x1238
#define FID_BITSTREAM_TEMPO                    0x3237
#define FID_PS_TEMPO                           0x2237
#define FID_DTC_TEMPO                          0x1237
#define FID_BITSTREAM_SYNTAX                   0x3236
#define FID_PS_SYNTAX                          0x2236
#define FID_DTC_SYNTAX                         0x1236
#define FID_BITSTREAM_STONE_SANS               0x3235
#define FID_PS_STONE_SANS                      0x2235
#define FID_DTC_STONE_SANS                     0x1235
#define FID_BITSTREAM_SERIF_GOTHIC             0x3234
#define FID_PS_SERIF_GOTHIC                    0x2234
#define FID_DTC_SERIF_GOTHIC                   0x1234
#define FID_BITSTREAM_PRIMUS_ANTIQUA           0x3233
#define FID_PS_PRIMUS_ANTIQUA                  0x2233
#define FID_DTC_PRIMUS_ANTIQUA                 0x1233
#define FID_BITSTREAM_PRIMUS                   0x3232
#define FID_PS_PRIMUS                          0x2232
#define FID_DTC_PRIMUS                         0x1232
#define FID_BITSTREAM_PRAXIS                   0x3231
#define FID_PS_PRAXIS                          0x2231
#define FID_DTC_PRAXIS                         0x1231
#define FID_BITSTREAM_PANACHE                  0x3230
#define FID_PS_PANACHE                         0x2230
#define FID_DTC_PANACHE                        0x1230
#define FID_BITSTREAM_OCR_B                    0x322f
#define FID_PS_OCR_B                           0x222f
#define FID_DTC_OCR_B                          0x122f
#define FID_BITSTREAM_OCR_A                    0x322e
#define FID_PS_OCR_A                           0x222e
#define FID_DTC_OCR_A                          0x122e
#define FID_BITSTREAM_NEWTEXT                  0x322d
#define FID_PS_NEWTEXT                         0x222d
#define FID_DTC_NEWTEXT                        0x122d
#define FID_BITSTREAM_NEWS_GOTHIC              0x322c
#define FID_PS_NEWS_GOTHIC                     0x222c
#define FID_DTC_NEWS_GOTHIC                    0x122c
```

Routines

```
#define FID_BITSTREAM_NEUZEIT_GROTESK        0x322b
#define FID_PS_NEUZEIT_GROTESK               0x222b
#define FID_DTC_NEUZEIT_GROTESK              0x122b
#define FID_BITSTREAM_MIXAGE                 0x322a
#define FID_PS_MIXAGE                        0x222a
#define FID_DTC_MIXAGE                       0x122a
#define FID_BITSTREAM_MAXIMA                 0x3229
#define FID_PS_MAXIMA                        0x2229
#define FID_DTC_MAXIMA                       0x1229
#define FID_BITSTREAM_LUCIDA_SANS            0x3228
#define FID_PS_LUCIDA_SANS                   0x2228
#define FID_DTC_LUCIDA_SANS                  0x1228
#define FID_BITSTREAM_LITERA                 0x3227
#define FID_PS_LITERA                        0x2227
#define FID_DTC_LITERA                       0x1227
#define FID_BITSTREAM_KABEL                  0x3226
#define FID_PS_KABEL                         0x2226
#define FID_DTC_KABEL                        0x1226
#define FID_BITSTREAM_HOLSATIA               0x3225
#define FID_PS_HOLSATIA                      0x2225
#define FID_DTC_HOLSATIA                     0x1225
#define FID_BITSTREAM_HELVETICA_INSERAT      0x3224
#define FID_PS_HELVETICA_INSERAT             0x2224
#define FID_DTC_HELVETICA_INSERAT            0x1224
#define FID_BITSTREAM_NEUE_HELVETICA         0x3223
#define FID_PS_NEUE_HELVETICA                0x2223
#define FID_DTC_NEUE_HELVETICA               0x1223
#define FID_BITSTREAM_HELVETICA              0x3222
#define FID_PS_HELVETICA                     0x2222
#define FID_DTC_HELVETICA                    0x1222
#define FID_BITSTREAM_HAAS_UNICA             0x3221
#define FID_PS_HAAS_UNICA                    0x2221
#define FID_DTC_HAAS_UNICA                   0x1221
#define FID_BITSTREAM_GOUDY_SANS             0x3220
#define FID_PS_GOUDY_SANS                    0x2220
#define FID_DTC_GOUDY_SANS                   0x1220
#define FID_BITSTREAM_GOTHIC                 0x321f
#define FID_PS_GOTHIC                        0x221f
#define FID_DTC_GOTHIC                       0x121f
#define FID_BITSTREAM_GILL_SANS              0x321e
#define FID_PS_GILL_SANS                     0x221e
#define FID_DTC_GILL_SANS                    0x121e
#define FID_BITSTREAM_GILL                   0x321d
#define FID_PS_GILL                          0x221d
#define FID_DTC_GILL                         0x121d
#define FID_BITSTREAM_FUTURA                 0x321c
#define FID_PS_FUTURA                        0x221c
#define FID_DTC_FUTURA                       0x121c
#define FID_BITSTREAM_FOLIO                  0x321b
#define FID_PS_FOLIO                         0x221b
```

```
#define FID_DTC_FOLIO                         0x121b
#define FID_BITSTREAM_FLYER                   0x321a
#define FID_PS_FLYER                          0x221a
#define FID_DTC_FLYER                         0x121a
#define FID_BITSTREAM_FETTE_MIDSCHRIFT        0x3219
#define FID_PS_FETTE_MIDSCHRIFT               0x2219
#define FID_DTC_FETTE_MIDSCHRIFT              0x1219
#define FID_BITSTREAM_FETTE_ENGSCHRIFT        0x3218
#define FID_PS_FETTE_ENGSCHRIFT               0x2218
#define FID_DTC_FETTE_ENGSCHRIFT              0x1218
#define FID_BITSTREAM_ERAS                    0x3217
#define FID_PS_ERAS                           0x2217
#define FID_DTC_ERAS                          0x1217
#define FID_BITSTREAM_DIGI_GROTESK            0x3216
#define FID_PS_DIGI_GROTESK                   0x2216
#define FID_DTC_DIGI_GROTESK                  0x1216
#define FID_BITSTREAM_CORINTHIAN              0x3215
#define FID_PS_CORINTHIAN                     0x2215
#define FID_DTC_CORINTHIAN                    0x1215
#define FID_BITSTREAM_COMPACTA                0x3214
#define FID_PS_COMPACTA                       0x2214
#define FID_DTC_COMPACTA                      0x1214
#define FID_BITSTREAM_CLEARFACE_GOTHIC        0x3213
#define FID_PS_CLEARFACE_GOTHIC               0x2213
#define FID_DTC_CLEARFACE_GOTHIC              0x1213
#define FID_BITSTREAM_OPTIMA                  0x3212
#define FID_PS_OPTIMA                         0x2212
#define FID_DTC_OPTIMA                        0x1212
#define FID_BITSTREAM_CHELMSFORD              0x3211
#define FID_PS_CHELMSFORD                     0x2211
#define FID_DTC_CHELMSFORD                    0x1211
#define FID_BITSTREAM_CASTLE                  0x3210
#define FID_PS_CASTLE                         0x2210
#define FID_DTC_CASTLE                        0x1210
#define FID_BITSTREAM_BRITANNIC               0x320f
#define FID_PS_BRITANNIC                      0x220f
#define FID_DTC_BRITANNIC                     0x120f
#define FID_BITSTREAM_BERLINER_GROTESK        0x320e
#define FID_PS_BERLINER_GROTESK               0x220e
#define FID_DTC_BERLINER_GROTESK              0x120e
#define FID_BITSTREAM_BENGUIAT_GOTHIC         0x320d
#define FID_PS_BENGUIAT_GOTHIC                0x220d
#define FID_DTC_BENGUIAT_GOTHIC               0x120d
#define FID_BITSTREAM_AVANTE_GARDE            0x320c
#define FID_PS_AVANTE_GARDE                   0x220c
#define FID_DTC_AVANTE_GARDE                  0x120c
#define FID_BITSTREAM_ANZEIGEN_GROTESK        0x320b
#define FID_PS_ANZEIGEN_GROTESK               0x220b
#define FID_DTC_ANZEIGEN_GROTESK              0x120b
#define FID_BITSTREAM_ANTIQUE_OLIVE           0x320a
```

Routines

```
#define FID_PS_ANTIQUE_OLIVE                   0x220a
#define FID_DTC_ANTIQUE_OLIVE                  0x120a
#define FID_BITSTREAM_ALTERNATE_GOTHIC         0x3209
#define FID_PS_ALTERNATE_GOTHIC                0x2209
#define FID_DTC_ALTERNATE_GOTHIC               0x1209
#define FID_BITSTREAM_AKZIDENZ_GROTESK_BUCH    0x3208
#define FID_PS_AKZIDENZ_GROTESK_BUCH           0x2208
#define FID_DTC_AKZIDENZ_GROTESK_BUCH          0x1208
#define FID_BITSTREAM_AKZIDENZ_GROTESK         0x3207
#define FID_PS_AKZIDENZ_GROTESK                0x2207
#define FID_DTC_AKZIDENZ_GROTESK               0x1207
#define FID_BITSTREAM_AVENIR                   0x3206
#define FID_PS_AVENIR                          0x2206
#define FID_DTC_AVENIR                         0x1206
#define FID_BITSTREAM_UNIVERS                  0x3205
#define FID_PS_UNIVERS                         0x2205
#define FID_DTC_UNIVERS                        0x1205
#define FID_BITSTREAM_FRANKLIN_GOTHIC          0x3204
#define FID_PS_FRANKLIN_GOTHIC                 0x2204
#define FID_DTC_FRANKLIN_GOTHIC                0x1204
#define FID_BITSTREAM_ANGRO                    0x3203
#define FID_PS_ANGRO                           0x2203
#define FID_DTC_ANGRO                          0x1203
#define FID_BITSTREAM_EUROSTILE                0x3202
#define FID_PS_EUROSTILE                       0x2202
#define FID_DTC_EUROSTILE                      0x1202
#define FID_BITSTREAM_FRUTIGER                 0x3201
#define FID_PS_FRUTIGER                        0x2201
#define FID_DTC_FRUTIGER                       0x1201
#define FID_BITSTREAM_URW_SANS                 0x3200
#define FID_PS_URW_SANS                        0x2200
#define FID_DTC_URW_SANS                       0x1200
#define FID_BITSTREAM_GALLIARD_ROMAN_ITALIC    0x307e
#define FID_PS_GALLIARD_ROMAN_ITALIC           0x207e
#define FID_DTC_GALLIARD_ROMAN_ITALIC          0x107e
#define FID_BITSTREAM_GRANJON                  0x307d
#define FID_PS_GRANJON                         0x207d
#define FID_DTC_GRANJON                        0x107d
#define FID_BITSTREAM_GARTH_GRAPHIC            0x307c
#define FID_PS_GARTH_GRAPHIC                   0x207c
#define FID_DTC_GARTH_GRAPHIC                  0x107c
#define FID_BITSTREAM_BAUER_BODONI             0x307b
#define FID_PS_BAUER_BODONI                    0x207b
#define FID_DTC_BAUER_BODONI                   0x107b
#define FID_BITSTREAM_BELWE                    0x307a
#define FID_PS_BELWE                           0x207a
#define FID_DTC_BELWE                          0x107a
#define FID_BITSTREAM_CHARLEMAGNE              0x3079
#define FID_PS_CHARLEMAGNE                     0x2079
#define FID_DTC_CHARLEMAGNE                    0x1079
```

```
#define FID_BITSTREAM_TRAJAN                    0x3078
#define FID_PS_TRAJAN                           0x2078
#define FID_DTC_TRAJAN                          0x1078
#define FID_BITSTREAM_ADOBE_GARAMOND            0x3077
#define FID_PS_ADOBE_GARAMOND                   0x2077
#define FID_DTC_ADOBE_GARAMOND                  0x1077
#define FID_BITSTREAM_ZAPF_INTERNATIONAL        0x3076
#define FID_PS_ZAPF_INTERNATIONAL               0x2076
#define FID_DTC_ZAPF_INTERNATIONAL              0x1076
#define FID_BITSTREAM_ZAPF_BOOK                 0x3075
#define FID_PS_ZAPF_BOOK                        0x2075
#define FID_DTC_ZAPF_BOOK                       0x1075
#define FID_BITSTREAM_WORCESTER_ROUND           0x3074
#define FID_PS_WORCESTER_ROUND                  0x2074
#define FID_DTC_WORCESTER_ROUND                 0x1074
#define FID_BITSTREAM_WINDSOR                   0x3073
#define FID_PS_WINDSOR                          0x2073
#define FID_DTC_WINDSOR                         0x1073
#define FID_BITSTREAM_WEISS                     0x3072
#define FID_PS_WEISS                            0x2072
#define FID_DTC_WEISS                           0x1072
#define FID_BITSTREAM_WEIDEMANN                 0x3071
#define FID_PS_WEIDEMANN                        0x2071
#define FID_DTC_WEIDEMANN                       0x1071
#define FID_BITSTREAM_WALBAUM                   0x3070
#define FID_PS_WALBAUM                          0x2070
#define FID_DTC_WALBAUM                         0x1070
#define FID_BITSTREAM_VOLTA                     0x306f
#define FID_PS_VOLTA                            0x206f
#define FID_DTC_VOLTA                           0x106f
#define FID_BITSTREAM_VENDOME                   0x306e
#define FID_PS_VENDOME                          0x206e
#define FID_DTC_VENDOME                         0x106e
#define FID_BITSTREAM_VELJOVIC                  0x306d
#define FID_PS_VELJOVIC                         0x206d
#define FID_DTC_VELJOVIC                        0x106d
#define FID_BITSTREAM_ADOBE_UTOPIA              0x306c
#define FID_PS_ADOBE_UTOPIA                     0x206c
#define FID_DTC_ADOBE_UTOPIA                    0x106c
#define FID_BITSTREAM_USHERWOOD                 0x306b
#define FID_PS_USHERWOOD                        0x206b
#define FID_DTC_USHERWOOD                       0x106b
#define FID_BITSTREAM_URW_ANTIQUA               0x306a
#define FID_PS_URW_ANTIQUA                      0x206a
#define FID_DTC_URW_ANTIQUA                     0x106a
#define FID_BITSTREAM_TIMES_NEW_ROMAN           0x3069
#define FID_PS_TIMES_NEW_ROMAN                  0x2069
#define FID_DTC_TIMES_NEW_ROMAN                 0x1069
#define FID_BITSTREAM_TIMELESS                  0x3068
#define FID_PS_TIMELESS                         0x2068
```

Routines

```
#define FID_DTC_TIMELESS                       0x1068
#define FID_BITSTREAM_TIFFANY                  0x3067
#define FID_PS_TIFFANY                         0x2067
#define FID_DTC_TIFFANY                        0x1067
#define FID_BITSTREAM_TIEPOLO                  0x3066
#define FID_PS_TIEPOLO                         0x2066
#define FID_DTC_TIEPOLO                        0x1066
#define FID_BITSTREAM_SWIFT                    0x3065
#define FID_PS_SWIFT                           0x2065
#define FID_DTC_SWIFT                          0x1065
#define FID_BITSTREAM_STYMIE                   0x3064
#define FID_PS_STYMIE                          0x2064
#define FID_DTC_STYMIE                         0x1064
#define FID_BITSTREAM_STRATFORD                0x3063
#define FID_PS_STRATFORD                       0x2063
#define FID_DTC_STRATFORD                      0x1063
#define FID_BITSTREAM_STONE_SERIF              0x3062
#define FID_PS_STONE_SERIF                     0x2062
#define FID_DTC_STONE_SERIF                    0x1062
#define FID_BITSTREAM_STONE_INFORMAL           0x3061
#define FID_PS_STONE_INFORMAL                  0x2061
#define FID_DTC_STONE_INFORMAL                 0x1061
#define FID_BITSTREAM_STEMPEL_SCHNEIDLER       0x3060
#define FID_PS_STEMPEL_SCHNEIDLER              0x2060
#define FID_DTC_STEMPEL_SCHNEIDLER             0x1060
#define FID_BITSTREAM_SOUVENIR                 0x305f
#define FID_PS_SOUVENIR                        0x205f
#define FID_DTC_SOUVENIR                       0x105f
#define FID_BITSTREAM_SLIMBACH                 0x305e
#define FID_PS_SLIMBACH                        0x205e
#define FID_DTC_SLIMBACH                       0x105e
#define FID_BITSTREAM_SERIFA                   0x305d
#define FID_PS_SERIFA                          0x205d
#define FID_DTC_SERIFA                         0x105d
#define FID_BITSTREAM_SABON_ANTIQUA            0x305c
#define FID_PS_SABON_ANTIQUA                   0x205c
#define FID_DTC_SABON_ANTIQUA                  0x105c
#define FID_BITSTREAM_SABON                    0x305b
#define FID_PS_SABON                           0x205b
#define FID_DTC_SABON                          0x105b
#define FID_BITSTREAM_ROMANA                   0x305a
#define FID_PS_ROMANA                          0x205a
#define FID_DTC_ROMANA                         0x105a
#define FID_BITSTREAM_ROCKWELL                 0x3059
#define FID_PS_ROCKWELL                        0x2059
#define FID_DTC_ROCKWELL                       0x1059
#define FID_BITSTREAM_RENAULT                  0x3058
#define FID_PS_RENAULT                         0x2058
#define FID_DTC_RENAULT                        0x1058
#define FID_BITSTREAM_RALEIGH                  0x3057
```

```
#define FID_PS_RALEIGH                          0x2057
#define FID_DTC_RALEIGH                         0x1057
#define FID_BITSTREAM_QUORUM                    0x3056
#define FID_PS_QUORUM                           0x2056
#define FID_DTC_QUORUM                          0x1056
#define FID_BITSTREAM_PROTEUS                   0x3055
#define FID_PS_PROTEUS                          0x2055
#define FID_DTC_PROTEUS                         0x1055
#define FID_BITSTREAM_PLANTIN                   0x3054
#define FID_PS_PLANTIN                          0x2054
#define FID_DTC_PLANTIN                         0x1054
#define FID_BITSTREAM_PERPETUA                  0x3053
#define FID_PS_PERPETUA                         0x2053
#define FID_DTC_PERPETUA                        0x1053
#define FID_BITSTREAM_PACELLA                   0x3052
#define FID_PS_PACELLA                          0x2052
#define FID_DTC_PACELLA                         0x1052
#define FID_BITSTREAM_NOVARESE                  0x3051
#define FID_PS_NOVARESE                         0x2051
#define FID_DTC_NOVARESE                        0x1051
#define FID_BITSTREAM_NIMROD                    0x3050
#define FID_PS_NIMROD                           0x2050
#define FID_DTC_NIMROD                          0x1050
#define FID_BITSTREAM_NIKIS                     0x304f
#define FID_PS_NIKIS                            0x204f
#define FID_DTC_NIKIS                           0x104f
#define FID_BITSTREAM_NAPOLEAN                  0x304e
#define FID_PS_NAPOLEAN                         0x204e
#define FID_DTC_NAPOLEAN                        0x104e
#define FID_BITSTREAM_MODERN_NO_216             0x304d
#define FID_PS_MODERN_NO_216                    0x204d
#define FID_DTC_MODERN_NO_216                   0x104d
#define FID_BITSTREAM_MODERN                    0x304c
#define FID_PS_MODERN                           0x204c
#define FID_DTC_MODERN                          0x104c
#define FID_BITSTREAM_MINISTER                  0x304b
#define FID_PS_MINISTER                         0x204b
#define FID_DTC_MINISTER                        0x104b
#define FID_BITSTREAM_MESSIDOR                  0x304a
#define FID_PS_MESSIDOR                         0x204a
#define FID_DTC_MESSIDOR                        0x104a
#define FID_BITSTREAM_MERIDIEN                  0x3049
#define FID_PS_MERIDIEN                         0x2049
#define FID_DTC_MERIDIEN                        0x1049
#define FID_BITSTREAM_MEMPHIS                   0x3048
#define FID_PS_MEMPHIS                          0x2048
#define FID_DTC_MEMPHIS                         0x1048
#define FID_BITSTREAM_MELIOR                    0x3047
#define FID_PS_MELIOR                           0x2047
#define FID_DTC_MELIOR                          0x1047
```

Routines

```
#define FID_BITSTREAM_MARCONI                   0x3046
#define FID_PS_MARCONI                          0x2046
#define FID_DTC_MARCONI                         0x1046
#define FID_BITSTREAM_MAGNUS                    0x3045
#define FID_PS_MAGNUS                           0x2045
#define FID_DTC_MAGNUS                          0x1045
#define FID_BITSTREAM_MAGNA                     0x3044
#define FID_PS_MAGNA                            0x2044
#define FID_DTC_MAGNA                           0x1044
#define FID_BITSTREAM_MADISON                   0x3043
#define FID_PS_MADISON                          0x2043
#define FID_DTC_MADISON                         0x1043
#define FID_BITSTREAM_LUCIDA                    0x3042
#define FID_PS_LUCIDA                           0x2042
#define FID_DTC_LUCIDA                          0x1042
#define FID_BITSTREAM_LUBALIN_GRAPH             0x3041
#define FID_PS_LUBALIN_GRAPH                    0x2041
#define FID_DTC_LUBALIN_GRAPH                   0x1041
#define FID_BITSTREAM_LIFE                      0x3040
#define FID_PS_LIFE                             0x2040
#define FID_DTC_LIFE                            0x1040
#define FID_BITSTREAM_LEAWOOD                   0x303f
#define FID_PS_LEAWOOD                          0x203f
#define FID_DTC_LEAWOOD                         0x103f
#define FID_BITSTREAM_KORINNA                   0x303e
#define FID_PS_KORINNA                          0x203e
#define FID_DTC_KORINNA                         0x103e
#define FID_BITSTREAM_JENSON_OLD_STYLE          0x303d
#define FID_PS_JENSON_OLD_STYLE                 0x203d
#define FID_DTC_JENSON_OLD_STYLE                0x103d
#define FID_BITSTREAM_JANSON                    0x303c
#define FID_PS_JANSON                           0x203c
#define FID_DTC_JANSON                          0x103c
#define FID_BITSTREAM_JAMILLE                   0x303b
#define FID_PS_JAMILLE                          0x203b
#define FID_DTC_JAMILLE                         0x103b
#define FID_BITSTREAM_ITALIA                    0x303a
#define FID_PS_ITALIA                           0x203a
#define FID_DTC_ITALIA                          0x103a
#define FID_BITSTREAM_IMPRESSUM                 0x3039
#define FID_PS_IMPRESSUM                        0x2039
#define FID_DTC_IMPRESSUM                       0x1039
#define FID_BITSTREAM_HOLLANDER                 0x3038
#define FID_PS_HOLLANDER                        0x2038
#define FID_DTC_HOLLANDER                       0x1038
#define FID_BITSTREAM_HIROSHIGE                 0x3037
#define FID_PS_HIROSHIGE                        0x2037
#define FID_DTC_HIROSHIGE                       0x1037
#define FID_BITSTREAM_HAWTHORN                  0x3036
#define FID_PS_HAWTHORN                         0x2036
```

```
#define FID_DTC_HAWTHORN                      0x1036
#define FID_BITSTREAM_GOUDY                   0x3035
#define FID_PS_GOUDY                          0x2035
#define FID_DTC_GOUDY                         0x1035
#define FID_BITSTREAM_GAMMA                   0x3034
#define FID_PS_GAMMA                          0x2034
#define FID_DTC_GAMMA                         0x1034
#define FID_BITSTREAM_GALLIARD                0x3033
#define FID_PS_GALLIARD                       0x2033
#define FID_DTC_GALLIARD                      0x1033
#define FID_BITSTREAM_FRIZ_QUADRATA           0x3032
#define FID_PS_FRIZ_QUADRATA                  0x2032
#define FID_DTC_FRIZ_QUADRATA                 0x1032
#define FID_BITSTREAM_FENICE                  0x3031
#define FID_PS_FENICE                         0x2031
#define FID_DTC_FENICE                        0x1031
#define FID_BITSTREAM_EXCELSIOR               0x3030
#define FID_PS_EXCELSIOR                      0x2030
#define FID_DTC_EXCELSIOR                     0x1030
#define FID_BITSTREAM_ESPRIT                  0x302f
#define FID_PS_ESPRIT                         0x202f
#define FID_DTC_ESPRIT                        0x102f
#define FID_BITSTREAM_ELAN                    0x302e
#define FID_PS_ELAN                           0x202e
#define FID_DTC_ELAN                          0x102e
#define FID_BITSTREAM_EGYPTIENNE              0x302d
#define FID_PS_EGYPTIENNE                     0x202d
#define FID_DTC_EGYPTIENNE                    0x102d
#define FID_BITSTREAM_EGIZIO                  0x302c
#define FID_PS_EGIZIO                         0x202c
#define FID_DTC_EGIZIO                        0x102c
#define FID_BITSTREAM_EDWARDIAN               0x302b
#define FID_PS_EDWARDIAN                      0x202b
#define FID_DTC_EDWARDIAN                     0x102b
#define FID_BITSTREAM_EDISON                  0x302a
#define FID_PS_EDISON                         0x202a
#define FID_DTC_EDISON                        0x102a
#define FID_BITSTREAM_DIGI_ANTIQUA            0x3029
#define FID_PS_DIGI_ANTIQUA                   0x2029
#define FID_DTC_DIGI_ANTIQUA                  0x1029
#define FID_BITSTREAM_DEMOS                   0x3028
#define FID_PS_DEMOS                          0x2028
#define FID_DTC_DEMOS                         0x1028
#define FID_BITSTREAM_CUSHING                 0x3027
#define FID_PS_CUSHING                        0x2027
#define FID_DTC_CUSHING                       0x1027
#define FID_BITSTREAM_CORONA                  0x3026
#define FID_PS_CORONA                         0x2026
#define FID_DTC_CORONA                        0x1026
#define FID_BITSTREAM_CONGRESS                0x3025
```

Routines

```
#define FID_PS_CONGRESS                        0x2025
#define FID_DTC_CONGRESS                       0x1025
#define FID_BITSTREAM_CONCORDE_NOVA            0x3024
#define FID_PS_CONCORDE_NOVA                   0x2024
#define FID_DTC_CONCORDE_NOVA                  0x1024
#define FID_BITSTREAM_CONCORDE                 0x3023
#define FID_PS_CONCORDE                        0x2023
#define FID_DTC_CONCORDE                       0x1023
#define FID_BITSTREAM_CLEARFACE                0x3022
#define FID_PS_CLEARFACE                       0x2022
#define FID_DTC_CLEARFACE                      0x1022
#define FID_BITSTREAM_CLARENDON                0x3021
#define FID_PS_CLARENDON                       0x2021
#define FID_DTC_CLARENDON                      0x1021
#define FID_BITSTREAM_CHELTENHAM               0x3020
#define FID_PS_CHELTENHAM                      0x2020
#define FID_DTC_CHELTENHAM                     0x1020
#define FID_BITSTREAM_CENTURY_OLD_STYLE        0x301f
#define FID_PS_CENTURY_OLD_STYLE               0x201f
#define FID_DTC_CENTURY_OLD_STYLE              0x101f
#define FID_BITSTREAM_CENTURY                  0x301e
#define FID_PS_CENTURY                         0x201e
#define FID_DTC_CENTURY                        0x101e
#define FID_BITSTREAM_CENTENNIAL               0x301d
#define FID_PS_CENTENNIAL                      0x201d
#define FID_DTC_CENTENNIAL                     0x101d
#define FID_BITSTREAM_CAXTON                   0x301c
#define FID_PS_CAXTON                          0x201c
#define FID_DTC_CAXTON                         0x101c
#define FID_BITSTREAM_ADOBE_CASLON             0x301b
#define FID_PS_ADOBE_CASLON                    0x201b
#define FID_DTC_ADOBE_CASLON                   0x101b
#define FID_BITSTREAM_CASLON                   0x301a
#define FID_PS_CASLON                          0x201a
#define FID_DTC_CASLON                         0x101a
#define FID_BITSTREAM_CANDIDA                  0x3019
#define FID_PS_CANDIDA                         0x2019
#define FID_DTC_CANDIDA                        0x1019
#define FID_BITSTREAM_BOOKMAN                  0x3018
#define FID_PS_BOOKMAN                         0x2018
#define FID_DTC_BOOKMAN                        0x1018
#define FID_BITSTREAM_BASKERVILLE_HANDCUT      0x3017
#define FID_PS_BASKERVILLE_HANDCUT             0x2017
#define FID_DTC_BASKERVILLE_HANDCUT            0x1017
#define FID_BITSTREAM_BASKERVILLE              0x3016
#define FID_PS_BASKERVILLE                     0x2016
#define FID_DTC_BASKERVILLE                    0x1016
#define FID_BITSTREAM_BASILIA                  0x3015
#define FID_PS_BASILIA                         0x2015
#define FID_DTC_BASILIA                        0x1015
```

```
#define FID_BITSTREAM_BARBEDOR               0x3014
#define FID_PS_BARBEDOR                      0x2014
#define FID_DTC_BARBEDOR                     0x1014
#define FID_BITSTREAM_AUREALIA               0x3013
#define FID_PS_AUREALIA                      0x2013
#define FID_DTC_AUREALIA                     0x1013
#define FID_BITSTREAM_NEW_ASTER              0x3012
#define FID_PS_NEW_ASTER                     0x2012
#define FID_DTC_NEW_ASTER                    0x1012
#define FID_BITSTREAM_ASTER                  0x3011
#define FID_PS_ASTER                         0x2011
#define FID_DTC_ASTER                        0x1011
#define FID_BITSTREAM_AMERICANA              0x3010
#define FID_PS_AMERICANA                     0x2010
#define FID_DTC_AMERICANA                    0x1010
#define FID_BITSTREAM_AACHEN                 0x300f
#define FID_PS_AACHEN                        0x200f
#define FID_DTC_AACHEN                       0x100f
#define FID_BITSTREAM_NICOLAS_COCHIN         0x300e
#define FID_PS_NICOLAS_COCHIN                0x200e
#define FID_DTC_NICOLAS_COCHIN               0x100e
#define FID_BITSTREAM_COCHIN                 0x300d
#define FID_PS_COCHIN                        0x200d
#define FID_DTC_COCHIN                       0x100d
#define FID_BITSTREAM_ALBERTUS               0x300c
#define FID_PS_ALBERTUS                      0x200c
#define FID_DTC_ALBERTUS                     0x100c
#define FID_BITSTREAM_ACCOLADE               0x300b
#define FID_PS_ACCOLADE                      0x200b
#define FID_DTC_ACCOLADE                     0x100b
#define FID_BITSTREAM_PALATINO               0x300a
#define FID_PS_PALATINO                      0x200a
#define FID_DTC_PALATINO                     0x100a
#define FID_BITSTREAM_GOUDY_OLD_STYLE        0x3009
#define FID_PS_GOUDY_OLD_STYLE               0x2009
#define FID_DTC_GOUDY_OLD_STYLE              0x1009
#define FID_BITSTREAM_BERKELEY_OLD_STYLE     0x3008
#define FID_PS_BERKELEY_OLD_STYLE            0x2008
#define FID_DTC_BERKELEY_OLD_STYLE           0x1008
#define FID_BITSTREAM_ARSIS                  0x3007
#define FID_PS_ARSIS                         0x2007
#define FID_DTC_ARSIS                        0x1007
#define FID_BITSTREAM_UNIVERSITY_ROMAN       0x3006
#define FID_PS_UNIVERSITY_ROMAN              0x2006
#define FID_DTC_UNIVERSITY_ROMAN             0x1006
#define FID_BITSTREAM_BEMBO                  0x3005
#define FID_PS_BEMBO                         0x2005
#define FID_DTC_BEMBO                        0x1005
#define FID_BITSTREAM_GARAMOND               0x3004
#define FID_PS_GARAMOND                      0x2004
```

Routines

```
    #define FID_DTC_GARAMOND                      0x1004
    #define FID_BITSTREAM_GLYPHA                  0x3003
    #define FID_PS_GLYPHA                         0x2003
    #define FID_DTC_GLYPHA                        0x1003
    #define FID_BITSTREAM_BODONI                  0x3002
    #define FID_PS_BODONI                         0x2002
    #define FID_DTC_BODONI                        0x1002
    #define FID_BITSTREAM_CENTURY_SCHOOLBOOK      0x3001
    #define FID_PS_CENTURY_SCHOOLBOOK             0x2001
    #define FID_DTC_CENTURY_SCHOOLBOOK            0x1001
    #define FID_BITSTREAM_URW_ROMAN               0x3000
    #define FID_PS_TIMES_ROMAN                    0x2000
    #define FID_DTC_URW_ROMAN                     0x1000
    #define FID_WINDOWS                           0x0a01
    #define FID_BISON                             0x0a00
    #define FID_LED                               0x0600
    #define FID_PMSYSTEM                          0x0203
    #define FID_BERKELEY                          0x0202
    #define FID_UNIVERSITY                        0x0201
    #define FID_CHICAGO                           0x0200
    #define FID_ROMA                              0x0001
    #define FID_INVALID                           0x0000
```

Fonts are normally referenced by FontID.

Include: fontID.h

■ FontMaker

```
typedef word FontMaker;
    #define FM_PRINTER        0xf000
    #define FM_MICROLOGIC     0xe000
    #define FM_ATECH          0xd000
    #define FM_PUBLIC         0xc000
    #define FM_AGFA           0x4000
    #define FM_BITSTREAM      0x3000
    #define FM_ADOBE          0x2000
    #define FM_NIMBUSQ        0x1000
    #define FM_BITMAP         0x0000
```

Include: fontID.h

■ FontMap

```
typedef byte FontMap;
    #define FM_DONT_USE       0x00ff
    #define FM_EXACT          0x0000
```

Include: fontID.h

■ FontWeight

```
typedef ByteEnum FontWeight;
    #define FW_ULTRA_LIGHT       0
    #define FW_EXTRA_LIGHT       1
    #define FW_LIGHT             2
    #define FW_BOOK              3
    #define FW_NORMAL            4
    #define FW_DEMI              5
    #define FW_BOLD              6
    #define FW_EXTRA_BOLD        7
    #define FW_ULTRA_BOLD        8
    #define FW_BLACK             9
```

Include: font.h

■ FontWidth

```
typedef  ByteEnum FontWidth;
    #define FWI_NARROW           0
    #define FWI_CONDENSED        1
    #define FWI_MEDIUM           2
    #define FWI_WIDE             3
    #define FWI_EXPANDED         4
```

Include: font.h

■ FormatArray

```
typedef ClipboardItemFormatInfo FormatArray[CLIPBOARD_MAX_FORMATS];
```

■ FormatError

```
typedef ByteEnum FormatError;
    #define FMT_DONE                                        0
    #define FMT_READY                                       1
    #define FMT_RUNNING                                     2
    #define FMT_DRIVE_NOT_READY                             3
    #define FMT_ERR_WRITING_BOOT                            4
    #define FMT_ERR_WRITING_ROOT_DIR                        5
    #define FMT_ERR_WRITING_FAT                             6
    #define FMT_ABORTED                                     7
    #define FMT_SET_VOLUME_NAME_ERR                         8
    #define FMT_CANNOT_FORMAT_FIXED_DISKS_IN_CUR_RELEASE 9
    #define FMT_BAD_PARTITION_TABLE                         10
    #define FMT_ERR_READING_PARTITION_TABLE                 11
    #define FMT_ERR_NO_PARTITION_FOUND                      12
    #define FMT_ERR_MULTIPLE_PRIMARY_PARTITIONS             13
    #define FMT_ERR_NO_EXTENDED_PARTITION_FOUND             14
    #define FMT_ERR_CANNOT_ALLOC_SECTOR_BUFFER              15
    #define FMT_ERR_DISK_IS_IN_USE                          16
```

```
#define FMT_ERR_WRITE_PROTECTED                         17
#define FMT_ERR_DRIVE_CANNOT_SUPPORT_GIVEN_FORMAT       18
#define FMT_ERR_INVALID_DRIVE_SPECIFIED                 19
#define FMT_ERR_DRIVE_CANNOT_BE_FORMATTED               20
#define FMT_ERR_DISK_UNAVAILABLE                        21
```

■ FunctionID

```
typedef enum /* word */ {
    FUNCTION_ID_ABS,
    FUNCTION_ID_ACOS,
    FUNCTION_ID_ACOSH,
    FUNCTION_ID_AND,
    FUNCTION_ID_ASIN,
    FUNCTION_ID_ASINH,
    FUNCTION_ID_ATAN,
    FUNCTION_ID_ATAN2,
    FUNCTION_ID_ATANH,
    FUNCTION_ID_AVG,
    FUNCTION_ID_CHAR,
    FUNCTION_ID_CHOOSE,
    FUNCTION_ID_CLEAN,
    FUNCTION_ID_CODE,
    FUNCTION_ID_COLS,
    FUNCTION_ID_COS,
    FUNCTION_ID_COSH,
    FUNCTION_ID_COUNT,
    FUNCTION_ID_CTERM,
    FUNCTION_ID_DATE,
    FUNCTION_ID_DATEVALUE,
    FUNCTION_ID_DAY,
    FUNCTION_ID_DDB,
    FUNCTION_ID_ERR,
    FUNCTION_ID_EXACT,
    FUNCTION_ID_EXP,
    FUNCTION_ID_FACT,
    FUNCTION_ID_FALSE,
    FUNCTION_ID_FIND,
    FUNCTION_ID_FV,
    FUNCTION_ID_HLOOKUP,
    FUNCTION_ID_HOUR,
    FUNCTION_ID_IF,
    FUNCTION_ID_INDEX,
    FUNCTION_ID_INT,
    FUNCTION_ID_IRR,
    FUNCTION_ID_ISERR,
    FUNCTION_ID_ISNUMBER,
    FUNCTION_ID_ISSTRING,
    FUNCTION_ID_LEFT,
    FUNCTION_ID_LENGTH,
    FUNCTION_ID_LN,
    FUNCTION_ID_LOG,
    FUNCTION_ID_LOWER,
    FUNCTION_ID_MAX,
    FUNCTION_ID_MID,
    FUNCTION_ID_MIN,
    FUNCTION_ID_MINUTE,
    FUNCTION_ID_MOD,
    FUNCTION_ID_MONTH,
    FUNCTION_ID_N,
    FUNCTION_ID_NA,
    FUNCTION_ID_NOW,
    FUNCTION_ID_NPV,
    FUNCTION_ID_OR,
    FUNCTION_ID_PI,
    FUNCTION_ID_PMT,
    FUNCTION_ID_PRODUCT,
    FUNCTION_ID_PROPER,
    FUNCTION_ID_PV,
    FUNCTION_ID_RANDOM_N,
    FUNCTION_ID_RANDOM,
    FUNCTION_ID_RATE,
    FUNCTION_ID_REPEAT,
    FUNCTION_ID_REPLACE,
    FUNCTION_ID_RIGHT,
    FUNCTION_ID_ROUND,
    FUNCTION_ID_ROWS,
    FUNCTION_ID_SECOND,
    FUNCTION_ID_SIN,
    FUNCTION_ID_SINH,
    FUNCTION_ID_SLN,
    FUNCTION_ID_SQRT,
    FUNCTION_ID_STD,
    FUNCTION_ID_STDP,
    FUNCTION_ID_STRING,
    FUNCTION_ID_SUM,
    FUNCTION_ID_SYD,
    FUNCTION_ID_TAN,
    FUNCTION_ID_TANH,
    FUNCTION_ID_TERM,
    FUNCTION_ID_TIME,
    FUNCTION_ID_TIMEVALUE,
    FUNCTION_ID_TODAY,
    FUNCTION_ID_TRIM,
    FUNCTION_ID_TRUE,
    FUNCTION_ID_TRUNC,
    FUNCTION_ID_UPPER,
    FUNCTION_ID_VALUE,
    FUNCTION_ID_VAR,
    FUNCTION_ID_VARP,
```

Routines

```
    FUNCTION_ID_VLOOKUP,
    FUNCTION_ID_WEEKDAY,
    FUNCTION_ID_YEAR,
    FUNCTION_ID_FILENAME,
    FUNCTION_ID_PAGE,
    FUNCTION_ID_PAGES,
    FUNCTION_ID_FIRST_EXTERNAL_FUNCTION=FUNCTION_ID_FIRST_EXTERNAL_FUNCTION_BASE
} FunctionID;
```

■ GCM_info

```
typedef enum /* word */ {
    GCMI_MIN_X,
    GCMI_MIN_X_ROUNDED,
    GCMI_MIN_Y,
    GCMI_MIN_Y_ROUNDED,
    GCMI_MAX_X,
    GCMI_MAX_X_ROUNDED,
    GCMI_MAX_Y,
    GCMI_MAX_Y_ROUNDED,
} GCM_info;
```

■ GCNDriveChangeNotificationType

```
typedef enum {
    GCNDCNT_CREATED,
    GCNDCNT_DESTROYED
} GCNDriveChangeNotificationType;
```

■ GCNExpressMenuNotificationType

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

■ GCNListBlockHeader

```
typedef struct {
    LMemBlockHeader GCNLBH_lmemHeader;
    ChunkHandle     GCNLBH_listOfLists;
} GCNListBlockHeader;
```

■ GCNListElement

```
typedef struct {
    optr    GCNLE_item;
} GCNListElement;
```

■ GCNListHeader

```
typedef struct {
    ChunkArrayHeader          GCNLH_meta;
    word                      GCNLH_statusEvent;
    MemHandle                 GCNLH_statusData;
    word                      GCNLH_statusCount;
    /* Start of GCNListOfListElements */
} GCNListHeader;
```

■ GCNListOfListsElement

```
typedef struct {
    GCNListType     GCNLOLE_ID;
    ChunkHandle     GCNLOLE_list;
} GCNListOfListsElement;
```

■ GCNListOfListsHeader

```
typedef struct {
    ChunkArrayHeader          GCNLOL_meta;
    /* Start of GCNListOfListsElement’s */
} GCNListOfListsHeader;
```

■ GCNListParams

```
typedef struct {
    GCNListType  GCNLP_ID;
    optr         GCNLP_optr;
} GCNListParams;
```

■ GCNListSendFlags

```
typedef WordFlags GCNListSendFlags;
    #define GCNLSF_SET_STATUS                          0x8000
    #define GCNLSF_IGNORE_IF_STATUS_TRANSITIONING  0x4000
```

■ GCNListType

```
typedef struct {
    word    GCNLT_manuf;
```

Routines

```
    word    GCNLT_type;
} GCNListType;
```

■ GCNListTypeFlags

```
typedef WordFlags GCNListTypeFlags;
    #define GCNLTF_SAVE_TO_STATE  0x8000
```

■ GCNShutdownControlType

```
typedef enum {
    GCNSCT_SUSPEND,
    GCNSCT_SHUTDOWN,
    GCNSCT_UNSUSPEND
} GCNShutdownControlType;
```

■ GCNStandardListType

```
typedef enum {
    GCNSLT_FILE_SYSTEM,
    GCNSLT_APPLICATION,
    GCNSLT_DATE_TIME,
    GCNSLT_DICTIONARY,
    GCNSLT_EXPRESS_MENU,
    GCNSLT_SHUTDOWN_CONTROL
} GCNStandardListType;
```

■ GenAppGCNListTypes

```
typedef enum /* word */ {
    GAGCNLT_GEN_CONTROL_OBJECTS,
    GAGCNLT_GEN_CONTROL_NOTIFY_STATUS_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SELECT_STATE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_SHEET_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_CHAR_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_PARA_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_TYPE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_SELECTION_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_COUNT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_STYLE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FONT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_POINT_SIZE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FONT_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_JUSTIFICATION_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_FG_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_BG_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_TYPE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_GROUP_FLAGS,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_AXIS_ATTRIBUTES,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_MARKER_SHAPE,
        GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_EXPRESSION_BUILDER_STATUS_CHANGE,
        GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_PROPERTIES_STATUS_CHANGE,
        GAGCNLT_APP_NOTIFY_DOC_SIZE_CHANGE,
        GAGCNLT_APP_NOTIFY_PAPER_SIZE_CHANGE,
        GAGCNLT_APP_TARGET_NOTIFY_VIEW_STATE_CHANGE,
        GAGCNLT_CONTROLLED_GEN_VIEW_OBJECTS
} GenAppGCNListTypes;
```

■ GeneralEvent

```
typedef enum {
        GE_NO_EVENT=0,            /* dummy event (NOP) */
        GE_END_OF_SONG=2,         /* marks end of song */
        GE_SET_PRIORITY=4,        /* changes sound priority */
        GE_SET_TEMPO=6,           /* changes sound tempo */
        GE_SEND_NOTIFICATION=8,   /* sends encoded message */
        GE_V_SEMAPHORE=10         /* V’s a specified semaphore*/
} GeneralEvent;
```

These represent some of the miscellaneous events which can make up a music buffer.

■ GenTravelOption

The **GenClass** defines some values meant to be used in the place of a **TravelOption** enumerated value. See **TravelOption**.

■ GeodeAttrs

```
typedef WordFlags GeodeAttrs;
        #define GA_PROCESS                      0x8000
        #define GA_LIBRARY                      0x4000
        #define GA_DRIVER                       0x2000
        #define GA_KEEP_FILE_OPEN               0x1000
        #define GA_SYSTEM                       0x0800
        #define GA_MULTI_LAUNCHABLE             0x0400
        #define GA_APPLICATION                  0x0200
        #define GA_DRIVER_INITIALIZED           0x0100
        #define GA_LIBRARY_INITIALIZED          0x0080
        #define GA_GEODE_INITIALIZED            0x0040
        #define GA_USES_COPROC                  0x0020
        #define GA_REQUIRES_COPROC              0x0010
        #define GA_HAS_GENERAL_CONSUMER_MODE    0x0008
        #define GA_ENTRY_POINTS_IN_C            0x0004
```

■ GeodeDefaultDriverType

```
typedef enum {
        GDDT_FILE_SYSTEM = 0,        /* File system driver */
        GDDT_KEYBOARD = 2,           /* Keyboard driver */
```

Routines

```
    GDDT_MOUSE = 4,              /* Mouse driver */
    GDDT_VIDEO = 6,              /* Video driver */
    GDDT_MEMORY_VIDEO = 8,       /* Vidmem driver */
    GDDT_POWER_MANAGEMENT = 10   /* Power management driver */
    GDDT_TASK = 12               /* Task driver */
} GeodeDefaultDriverType;
```

The default driver type has one value for each default driver type in GEOS. This type is used with **GeodeGetDefaultDriver()** and **GeodeSetDefaultDriver()**.

■ GeodeGetInfoType

```
typedef enum /* word */ {
    GGIT_ATTRIBUTES=0,
    GGIT_TYPE=2,
    GGIT_GEODE_RELEASE=4,
    GGIT_GEODE_PROTOCOL=6,
    GGIT_TOKEN_ID=8,
    GGIT_PERM_NAME_AND_EXT=10,
    GGIT_PERM_NAME_ONLY=12,
} GeodeGetInfoType;
```

■ GeodeHandle

```
typedef Handle GeodeHandle;
```

A standard handle that contains information about a loaded geode. When a geode has been loaded, it is referred to by its handle.

■ GeodeHeapVars

```
typedef struct {
     word       GHV_heapSpace;
    } GeodeHeapVars;
```

■ GeodeLoadError

```
typedef enum {
    GLE_PROTOCOL_IMPORTER_TOO_RECENT,
    GLE_PROTOCOL_IMPORTER_TOO_OLD,
    GLE_FILE_NOT_FOUND,
    GLE_LIBRARY_NOT_FOUND,
    GLE_FILE_READ_ERROR,
    GLE_NOT_GEOS_FILE,
    GLE_NOT_GEOS_EXECUTABLE_FILE,
    GLE_ATTRIBUTE_MISMATCH,
    GLE_MEMORY_ALLOCATION_ERROR,
    GLE_NOT_MULTI_LAUNCHABLE,
    GLE_LIBRARY_PROTOCOL_ERROR,
    GLE_LIBRARY_LOAD_ERROR,
    GLE_DRIVER_INIT_ERROR,
    GLE_LIBRARY_INIT_ERROR,
    GLE_DISK_TOO_FULL,
    GLE_FIELD_DETACHING,
} GeodeLoadError;
```

These errors may be returned by routines that load geodes, including **UserLoadApplication()**, **GeodeUseLibrary()**, **GeodeUseDriver()**, and **GeodeLoad()**.

■ GeodeToken

```
typedef struct {
    TokenChars      GT_chars;
    ManufacturerID  GT_manufID;
} GeodeToken;
```

Defines a token identifier. The *GT_chars* field is four characters that identify the token; *GT_manufID* is the identifying number of the manufacturer of the item being referenced.

■ GeosFileHeaderFlags

```
typedef WordFlags GeosFileHeaderFlags;
    #define GFHF_TEMPLATE             0x8000
    #define GFHF_SHARED_MULTIPLE      0x4000
    #define GFHF_SHARED_SINGLE        0x2000
```

■ GeosFileType

```
typedef enum /* word */ {
    GFT_NOT_GEOS_FILE,
    GFT_EXECUTABLE,
    GFT_VM,
    GFT_DATA,
    GFT_DIRECTORY,
    GFT_LINK
} GeosFileType;
```

GEOS files are divided into several broad categories. You can find out a file's category by getting its FEA_FILE_TYPE extended attribute. This attribute is a member of the **GeosFileType** enumerated type. This type has the following values:

GFT_NOT_GEOS_FILE
: The file is not a GEOS file. This constant is guaranteed to be equal to zero.

GFT_EXECUTABLE
: The file is executable; in other words, it is some kind of geode.

Routines

GFT_VM	The file is a VM file.
GFT_DATA	The file is a GEOS byte file (see below).
GFT_DIRECTORY	The file is a GEOS directory (not yet implemented).
GFT_LINK	The file is a symbolic link (not yet implemented).

■ GeoWorksGenAppGCNListType

```
typedef enum /* word */ {
    GAGCNLT_SELF_LOAD_OPTIONS = 0x6800,
    GAGCNLT_GEN_CONTROL_NOTIFY_STATUS_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SELECT_STATE_CHANGE,
    GAGCNLT_EDIT_CONTROL_NOTIFY_UNDO_STATE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_CHAR_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_PARA_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_TYPE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_SELECTION_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_COUNT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_TEXT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_SHEET_TEXT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_STYLE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FONT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_POINT_SIZE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FONT_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_JUSTIFICATION_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_FG_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_BG_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_PARA_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_BORDER_COLOR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SEARCH_SPELL_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SEARCH_REPLACE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_TYPE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_GROUP_FLAGS,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_AXIS_ATTRIBUTES,
    GAGCNLT_APP_TARGET_NOTIFY_CHART_MARKER_SHAPE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_CURRENT_TOOL_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_BODY_SELECTION_STATE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_AREA_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_LINE_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_TEXT_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_GROBJ_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_STYLE_SHEET_GROBJ_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_BODY_INSTRUCTION_FLAGS_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_GROBJ_GRADIENT_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_RULER_TYPE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_RULER_GRID_CHANGE,
    GAGCNLT_TEXT_RULER_OBJECTS,
    GAGCNLT_APP_TARGET_NOTIFY_BITMAP_CURRENT_TOOL_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_BITMAP_CURRENT_FORMAT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_PROPERTIES_STATUS_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_LIST_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_RCP_STATUS_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_APPEARANCE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_DUMMY_CHANGE_2,
    GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_DUMMY_CHANGE_3,
    GAGCNLT_APP_NOTIFY_DOC_SIZE_CHANGE,
    GAGCNLT_APP_NOTIFY_PAPER_SIZE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_VIEW_STATE_CHANGE,
    GAGCNLT_CONTROLLED_GEN_VIEW_OBJECTS,
    GAGCNLT_APP_TARGET_NOTIFY_INK_STATE_CHANGE,
    GAGCNLT_CONTROLLED_INK_OBJECTS,
    GAGCNLT_APP_TARGET_NOTIFY_PAGE_STATE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_DOCUMENT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_DISPLAY_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_DISPLAY_LIST_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPLINE_MARKER_SHAPE,
    GAGCNLT_APP_TARGET_NOTIFY_SPLINE_POINT,
    GAGCNLT_APP_TARGET_NOTIFY_SPLINE_POLYLINE,
    GAGCNLT_APP_TARGET_NOTIFY_SPLINE_SMOOTHNESS,
    GAGCNLT_APP_TARGET_NOTIFY_SPLINE_OPEN_CLOSE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_ACTIVE_CELL_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_EDIT_BAR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_SELECTION_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_WIDTH_HEIGHT_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_DOC_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_ATTR_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_NOTES_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_DATA_RANGE_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_TEXT_NAME_CHANGE,
    GAGCNLT_FLOAT_FORMAT_CHANGE,
    GAGCNLT_DISPLAY_OBJECTS_WITH_RULERS,
    GAGCNLT_APP_TARGET_NOTIFY_APP_CHANGE,
    GAGCNLT_APP_TARGET_NOTIFY_LIBRARY_CHANGE,
    GAGCNLT_WINDOWS,
    GAGCNLT_STARTUP_LOAD_OPTIONS
} GeoWorksGenAppGCNListType;
```

■ GeoWorksMetaGCNListType

```
    typedef enum /* word */ {
        MGCNLT_ACTIVE_LIST = 0x00,
        MGCNLT_APP_STARTUP = 0x02
    } GeoWorksMetaGCNListType;
```

■ GeoWorksNotificationType

```
    typedef enum {
        GWNT_INK,
        GWNT_GEN_CONTROL_NOTIFY_STATUS_CHANGE,
        GWNT_SELECT_STATE_CHANGE,
        GWNT_UNDO_STATE_CHANGE,
        GWNT_STYLE_CHANGE,
        GWNT_STYLE_SHEET_CHANGE,
        GWNT_TEXT_CHAR_ATTR_CHANGE,
        GWNT_TEXT_PARA_ATTR_CHANGE,
        GWNT_TEXT_TYPE_CHANGE,
        GWNT_TEXT_SELECTION_CHANGE,
        GWNT_TEXT_COUNT_CHANGE,
        GWNT_TEXT_STYLE_CHANGE,
        GWNT_FONT_CHANGE,
        GWNT_POINT_SIZE_CHANGE,
        GWNT_FONT_ATTR_CHANGE,
        GWNT_JUSTIFICATION_CHANGE,
        GWNT_TEXT_FG_COLOR_CHANGE,
        GWNT_TEXT_BG_COLOR_CHANGE,
        GWNT_TEXT_PARA_COLOR_CHANGE,
        GWNT_TEXT_BORDER_COLOR_CHANGE,
        GWNT_SEARCH_REPLACE_ENABLE_CHANGE,
        GWNT_SPELL_ENABLE_CHANGE,
        GWNT_CHART_TYPE_CHANGE,
        GWNT_CHART_GROUP_FLAGS,
        GWNT_CHART_AXIS_ATTRIBUTES,
        GWNT_GROBJ_CURRENT_TOOL_CHANGE,
        GWNT_GROBJ_BODY_SELECTION_STATE_CHANGE,
        GWNT_GROBJ_AREA_ATTR_CHANGE,
        GWNT_GROBJ_LINE_ATTR_CHANGE,
        GWNT_GROBJ_TEXT_ATTR_CHANGE,
        GWNT_GROBJ_BODY_INSTRUCTION_FLAGS_CHANGE,
        GWNT_GROBJ_GRADIENT_ATTR_CHANGE,
        GWNT_RULER_TYPE_CHANGE,
        GWNT_RULER_GRID_CHANGE,
        GWNT_RULER_GUIDE_CHANGE,
        GWNT_BITMAP_CURRENT_TOOL_CHANGE,
        GWNT_BITMAP_CURRENT_FORMAT_CHANGE,
        GWNT_FLAT_FILE_FIELD_PROPERTIES_STATUS_CHANGE,
        GWNT_FLAT_FILE_FIELD_LIST_CHANGE,
        GWNT_FLAT_FILE_RCP_STATUS_CHANGE,
        GWNT_FLAT_FIELD_APPEARANCE_CHANGE,
        GWNT_FLAT_FILE_DUMMY_CHANGE_2,
        GWNT_FLAT_FILE_DUMMY_CHANGE_3,
        GWNT_SPOOL_DOC_OR_PAPER_SIZE,
        GWNT_VIEW_STATE_CHANGE,
        GWNT_INK_HAS_TARGET,
        GWNT_PAGE_STATE_CHANGE,
        GWNT_DOCUMENT_CHANGE,
        GWNT_DISPLAY_CHANGE,
        GWNT_DISPLAY_LIST_CHANGE,
        GWNT_SPLINE_MARKER_SHAPE,
        GWNT_SPLINE_POINT,
        GWNT_SPLINE_POLYLINE,
        GWNT_SPLINE_SMOOTHNESS,
        GWNT_SPLINE_OPEN_CLOSE_CHANGE,
        GWNT_UNUSED_1,
        GWNT_SPREADSHEET_ACTIVE_CELL_CHANGE,
        GWNT_SPREADSHEET_EDIT_BAR_CHANGE,
        GWNT_SPREADSHEET_SELECTION_CHANGE,
        GWNT_SPREADSHEET_CELL_WIDTH_HEIGHT_CHANGE,
        GWNT_SPREADSHEET_DOC_ATTR_CHANGE,
        GWNT_SPREADSHEET_CELL_ATTR_CHANGE,
        GWNT_SPREADSHEET_CELL_NOTES_CHANGE,
        GWNT_SPREADSHEET_DATA_RANGE_CHANGE,
        GWNT_FLOAT_FORMAT_CHANGE,
        GWNT_MAP_APP_CHANGE,
        GWNT_MAP_LIBRARY_CHANGE,
        GWNT_TEXT_NAME_CHANGE,
        GWNT_CARD_BACK_CHANGE,
        GWNT_TEXT_OBJECT_HAS_FOCUS,
        GWNT_TEXT_CONTEXT,
        GWNT_TEXT_REPLACE_WITH_HWR,
        GWNT_HELP_CONTEXT_CHANGE,
        GWNT_FLOAT_FORMAT_INIT,
        GWNT_HARD_ICON_BAR_FUNCTION,
        GWNT_STARTUP_INDEXED_APP,
        GWNT_SPOOL_PRINTING_COMPLETE,
        GWNT_MODAL_WIN_CHANGE,
        GWNT_SPREADSHEET_NAME_CHANGE,
        GWNT_DOCUMENT_OPEN_COMPLETE,
        GWNT_EMAIL_SCAN_INBOX,
        GWNT_FOCUS_WINDOW_KBD_STATUS,
        GWNT_TAB_DOUBLE_CLICK,
        GWNT_PAGE_INFO_STATE_CHANGE,
        GWNT_CURSOR_POSITION_CHANGE,
        GWNT_FAX_NEW_JOB_CREATED,
        GWNT_FAX_NEW_JOB_COMPLETED,
        GWNT_EMAIL_DATABASE_CHANGE,
        GWNT_EMAIL_STATUS_CHANGE,
        GWNT_EMAIL_PAGE_PANEL_UPDATE,
        GWNT_PCCOM_DISPLAY_CHAR,
        GWNT_PCCOM_DISPLAY_STRING,
        GWNT_PCCOM_EXIT
} GeoWorksNotificationType;
```

■ GeoWorksVisContentGCNListType

```
typedef enum {
     VCGCNLT_TARGET_NOTIFY_TEXT_PARA_ATTR_CHANGE = 0x4a00,
     PADDING_VCGCNLT_INVALID_ITEM_000
    } GeoWorksVisContentGCNListType;
```

Routines

■ GetMaskType

```
typedef ByteEnum GetMaskType;
    #define GMT_ENUM          0
    #define GMT_BUFFER        1
```

■ GetPalType

```
typedef ByteEnum GetPalType;
    #define GPT_ACTIVE        0
    #define GPT_CUSTOM        1
    #define GPT_DEFAULT       2
```

■ GFM_info

```
 GFMI_HEIGHT=0, /* 0 */
     GFMI_HEIGHT_ROUNDED=1,
     GFMI_MEAN=2,
     GFMI_MEAN_ROUNDED=3,
     GFMI_DESCENT=4,
     GFMI_DESCENT_ROUNDED=5,
     GFMI_BASELINE=6,
     GFMI_BASELINE_ROUNDED=7,
     GFMI_LEADING=8,
     GFMI_LEADING_ROUNDED=9,
     GFMI_AVERAGE_WIDTH=10, /* 10 */
     GFMI_AVERAGE_WIDTH_ROUNDED=11,
     GFMI_ASCENT=12,
     GFMI_ASCENT_ROUNDED=13,
     GFMI_MAX_WIDTH=14,
     GFMI_MAX_WIDTH_ROUNDED=15,
     GFMI_MAX_ADJUSTED_HEIGHT=16,
     GFMI_MAX_ADJUSTED_HEIGHT_ROUNDED=17,
     GFMI_UNDER_POS=18,
     GFMI_UNDER_POS_ROUNDED=19,
     GFMI_UNDER_THICKNESS=20, /* 20 */
     GFMI_UNDER_THICKNESS_ROUNDED=21,
     GFMI_ABOVE_BOX=22,
     GFMI_ABOVE_BOX_ROUNDED=23,
     GFMI_ACCENT=24,
     GFMI_ACCENT_ROUNDED=25,
     GFMI_MANUFACTURER=26, /* 26 */
     GFMI_KERN_COUNT=28, /* 28 */
     GFMI_FIRST_CHAR=30, /* 30 */
     GFMI_LAST_CHAR=32, /* 32 */
     GFMI_DEFAULT_CHAR=34, /* 34 */
     GFMI_STRIKE_POS=36, /* 36 */
     GFMI_STRIKE_POS_ROUNDED=37,
     GFMI_BELOW_BOX=38,
    GFMI_BELOW_BOX_ROUNDED=39,
} GFM_info;
```

■ GraphicPattern

```
typedef struct {
    PatternType HP_type;
    byte HP_data;
} GraphicPattern;
```

■ GSControl

```
typedef WordFlags GSControl;
    #define GSC_PARTIAL       0x0200
    #define GSC_ONE           0x0100
    #define GSC_MISC          0x0080
    #define GSC_LABEL         0x0040
    #define GSC_ESCAPE        0x0020
    #define GSC_NEW_PAGE      0x0010
    #define GSC_XFORM         0x0008
    #define GSC_OUTPUT        0x0004
    #define GSC_ATTR          0x0002
    #define GSC_PATH          0x0001
```

■ GSRetType

```
typedef ByteEnum GSRetType;
    #define GSRT_COMPLETE     0
    #define GSRT_FORM_FEED    1
    #define GSRT_ONE          2
    #define GSRT_ESCAPE       3
    #define GSRT_OUTPUT       4
    #define GSRT_ELEMENT      5
    #define GSRT_FAULT        0xff
```

■ GState

GStates are always referenced by means of GStateHandles, and are documented there.

■ GStateHandle

```
typedef Handle GStateHandle;
```

GStates, or graphics states, are used to interpret graphics commands. Any graphics command that draws anything takes a GStateHandle as an argument. Each GState is associated with a window, and the graphics system uses the GState to determine which window the command should affect.

Routines

The GState also holds considerable information determining how drawing commands will be carried out. For instance, it holds the line color. To draw a green line, first one routine set's the GState's line color to green. From then on (or until the line color is changed again), all lines drawn using that GState will be green. Thus, all commands that set color, pattern, or other drawing attributes take a GStateHandle argument.

GStateHandles are also used when creating bitmaps and graphics strings. In this case, the associated window is fake; all drawing commands passed a GStateHandle representing a bitmap or graphics string will affect the data structure instead of being drawn to screen.

■ GString

```
typedef void GString;
```

A GString (short for “Graphics Strings”) represents a string of graphics commands. Each GString is made up of one or more GString elements, each of which corresponds to some standard graphics command.

GStrings may be created by means of drawing to a GStateHandle returned by **GrCreateState()**, but quite often GStrings are declared explicitly. The GString's data is often set up using macros like GSDrawLine(). These macros will output an opcode (of type **GStringElement**) and format their macro arguments into data expected with the opcode.

For instance,

```
GSDrawLine(72, 144, 216, 288);
```

Would expand to the data:

```
(GStringElement)    GR_DRAW_LINE
(sword)             72, 144, 216, 288
```

Thus, these macros just represent data, though they look like normal kernel graphics commands.

■ GStringElement

```
typedef ByteEnum GStringElement;
    /* The following elements are defined :
            (Miscellaneous GString opcodes:)
    GR_END_STRING,
    GR_COMMENT,                  (data: variable (word (length of code), code))
    GR_NULL_OP,
    GR_SET_GSTRING_BOUNDS,       (data: 8 bytes (4 swords))
    GR_LABEL,                    (data: 2 bytes (word))
    GR_ESCAPE,                   (data: variable (word (size of code), code))
    GR_NEW_PAGE,
            (Coordinate Transform opcodes:)
    GR_APPLY_ROTATION,            (data: 4 bytes (WWFixed))
    GR_APPLY_SCALE,               (data: 8 bytes (2 WWFixed))
    GR_APPLY_TRANSLATION,         (data: 8 bytes (2 WWFixed))
    GR_APPLY_TRANSFORM,           (data: 26 bytes(4 WWFixed, 2 DWFixed))
    GR_APPLY_TRANSLATION_DWORD,   (data: 8 bytes (2 sdwords))
    GR_SET_TRANSFORM,             (data: 26 bytes (4 WWFixed, 2 DWFixed))
    GR_SET_NULL_TRANSFORM,
    GR_SET_DEFAULT_TRANSFORM,
    GR_INIT_DEFAULT_TRANSFORM,
    GR_SAVE_TRANSFORM,
    GR_RESTORE_TRANSFORM,
            (Output opcodes:)
    GR_DRAW_LINE,                 (data: 8 bytes (4 swords))
    GR_DRAW_LINE_TO,              (data: 4 bytes (2 swords))
    GR_DRAW_REL_LINE_TO           (data: 8 bytes (2 WWFixed))
    GR_DRAW_HLINE,                (data: 6 bytes (3 swords))
    GR_DRAW_HLINE_TO,             (data: 2 bytes (sword))
    GR_DRAW_VLINE,                (data: 6 bytes (3 swords))
    GR_DRAW_VLINE_TO,             (data: 2 bytes (sword))
    GR_DRAW_POLYLINE,             (data: variable (word (# of points), points)
    GR_DRAW_ARC,                  (data: 14 bytes(ArcCloseType, 6 swords))
    GR_DRAW_ARC_3POINT,           (data: 14 bytes(ArcCloseType, 6 swords))
    GR_DRAW_ARC_3POINT_TO,        (data: 10 bytes(ArcCloseType, 4 swords))
    GR_DRAW_REL_ARC_3POINT_TO,    (data: 18 bytes(ArcCloseType, 4 WWFixed))
    GR_DRAW_RECT,                 (data: 8 bytes (4 swords))
    GR_DRAW_RECT_TO,              (data: 4 bytes (2 swords))
    GR_DRAW_ROUND_RECT,           (data: 10 bytes(word, 4 swords))
    GR_DRAW_ROUND_RECT_TO,        (data: 6 bytes (word, 2 swords))
    GR_DRAW_SPLINE,               (data: variable (word (# of points), points))
    GR_DRAW_SPLINE_TO,            (data: variable (word (# of points), points))
    GR_DRAW_CURVE,                (data: 16 bytes(8 swords))
    GR_DRAW_CURVE_TO,             (data: 12 bytes(6 swords))
    GR_DRAW_REL_CURVE_TO,         (data: 24 bytes(6 WWFixed))
    GR_DRAW_ELLIPSE,              (data: 8 bytes (4 swords))
    GR_DRAW_POLYGON,              (data: variable (word (# of points), points))
    GR_DRAW_POINT,                (data: 4 bytes (2 words))
    GR_DRAW_POINT_CP,
    GR_BRUSH_POLYLINE,            (data: variable (word (# of points), 2 bytes,
                                        points))
    GR_DRAW_CHAR,                 (data: 5 bytes) (Chars, 2 swords))
    GR_DRAW_CHAR_CP,              (data: 1 byte) (Chars))
    GR_DRAW_TEXT,                 (data: variable (sword, sword,
                                   word (length of string),
                                        string (not null terminated)))
    GR_DRAW_TEXT_CP,              (data: variable (word (length of string),
                                        string (not null terminated)))
    GR_DRAW_TEXT_PTR,             (data: 6 bytes (2 swords, (char *)))
    GR_DRAW_TEXT_OPTR,            (data: 6 bytes (2 swords, optr))
    GR_DRAW_PATH,
```

Routines

```
GR_FILL_RECT,                (data: 8 bytes (4 swords))
GR_FILL_RECT_TO,             (data: 4 bytes (2 swords))
GR_FILL_ROUND_RECT,          (data: 10 bytes(4 swords, word))
GR_FILL_ROUND_RECT_TO,       (data: 6 bytes (2 swords, word))
GR_FILL_ARC,                 (data: 14 bytes (ArcCloseType, 6 swords))
GR_FILL_POLYGON,             (data: variable (word (# of points),
                                    RegionFillRule, points))
GR_FILL_ELLIPSE,             (data: 8 bytes (2 swords))
GR_FILL_PATH,                (data: 1 byte  (RegionFillRule))
GR_FILL_ARC_3POINT,          (data: 14 bytes(ArcCloseType, 6 swords))
GR_FILL_ARC_3POINT_TO        (data: 10 bytes(ArcCloseType, 4 swords))
GR_FILL_BITMAP,              (data: 6 bytes (2 swords, word))
GR_FILL_BITMAP_CP,           (data: 2 bytes (word))
GR_FILL_BITMAP_OPTR,
GR_DRAW_BITMAP,              (data: 6 bytes (2 swords, word))
GR_DRAW_BITMAP_CP,           (data: 2 bytes (word))
GR_DRAW_BITMAP_OPTR,         (data: 6 bytes (2 swords, optr))
GR_DRAW_BITMAP_PTR,          (data: 6 bytes (2 swords, *))
GSE_BITMAP_SLICE,            (data: variable)
        (Drawing Attribute opcodes:)
GR_SAVE_STATE,
GR_RESTORE_STATE,
GR_SET_MIX_MODE,             (data: 1 byte  (MixMode))
GR_MOVE_TO,                  (data: 4 bytes (2 swords))
GR_REL_MOVE_TO,              (data: 8 bytes (2 WWFixed))
GR_CREATE_PALETTE,
GR_DESTROY_PALETTE,
GR_SET_PALETTE_ENTRY,        (data: 4 bytes (Color, 3 bytes))
GR_SET_PALETTE,              (data: variable (word (# of entries),
                                    entries (3 bytes each)))
GR_SET_LINE_COLOR,           (data: 3 bytes (3 bytes))
GR_SET_LINE_MASK,            (data: 1 byte  (SysDrawMask))
GR_SET_LINE_COLOR_MAP,       (data: 1 byte  (ColorMapMode))
GR_SET_LINE_WIDTH,           (data: 4 bytes (WWFixed))
GR_SET_LINE_JOIN,            (data: 1 byte  (LineJoin))
GR_SET_LINE_END,             (data: 1 byte  (LineEnd))
GR_SET_LINE_ATTR,            (data: 9 bytes (CF_RGB, 3 bytes, SysDrawMask,
                                ColorMapMode, LineEnd, LineJoin, LineStyle)
GR_SET_MITER_LIMIT,          (data: 4 bytes (WWFixed))
GR_SET_LINE_STYLE,           (data: 2 bytes (LineStyle, index))
GR_SET_LINE_COLOR_INDEX,     (data: 1 byte  (Color))
GR_SET_CUSTOM_LINE_MASK,     (data: 8 bytes (8 bytes))
GR_SET_CUSTOM_LINE_STYLE,    (data: variable (word (index),
                                    word (# of on-off dash pairs),
                                    pairs (each pair is 2 bytes)))
GR_SET_AREA_COLOR,           (data: 3 bytes (3 bytes)
GR_SET_AREA_MASK,            (data: 1 byte  (SysDrawMask))
GR_SET_AREA_COLOR_MAP,       (data: 1 byte  (ColorMapMode))
GR_SET_AREA_ATTR,            (data: 6 bytes (CF_RGB, 3 bytes, SysDrawMask,
                                    ColorMapMode))
```

```
    GR_SET_AREA_COLOR_INDEX,      (data: 1 byte  (Color))
    GR_SET_CUSTOM_AREA_MASK,      (data: 8 bytes (8 bytes))
    GR_SET_AREA_PATTERN,          (data: 2 bytes (GraphicPattern))
    GR_SET_CUSTOM_AREA_PATTERN,   (data: variable (GraphicPattern,
                                         word (size of data)
                                         pattern data))
    GR_SET_TEXT_COLOR,            (data: 3 bytes (3 bytes))
    GR_SET_TEXT_MASK,             (data: 1 byte  (SysDrawMask))
    GR_SET_TEXT_COLOR_MAP,        (data: 1 byte  (ColorMapMode))
    GR_SET_TEXT_STYLE,            (data: 2 bytes (2 TextStyles))
    GR_SET_TEXT_MODE,             (data: 2 bytes (2 TextModes))
    GR_SET_TEXT_SPACE_PAD,        (data: 3 bytes (WBFixed))
    GR_SET_TEXT_ATTR,             (data: 20 bytes(CF_RGB, 3 bytes, SysDrawMask,
                                         ColorMapMode, 2 TextStyles,
                                         2 TextModes, WBFixed, FontID, word))
    GR_SET_FONT,                  (data: 5 bytes (WBFixed, FontID))
    GR_SET_TEXT_COLOR_INDEX,      (data: 1 byte  (Color))
    GR_SET_CUSTOM_TEXT_MASK,      (data: 8 bytes ()
    GR_SET_TRACK_KERN,            (data: 2 bytes (sword))
    GR_SET_FONT_WEIGHT,           (data: 2 bytes (FontWeight))
    GR_SET_FONT_WIDTH,            (data: 2 bytes (FontWidth))
    GR_SET_SUPERSCRIPT_ATTR,      (data: 2 bytes (position, scale))
    GR_SET_SUBSCRIPT_ATTR,        (data: 2 bytes (position, scale))
    GR_SET_TEXT_PATTERN,          (data: 2 bytes (GraphicPattern))
    GR_SET_CUSTOM_TEXT_PATTERN,   (data: variable (GraphicPattern,
                                         word (size of data),
                                         pattern data))
            (Path opcodes:)
    GR_BEGIN_PATH,                (data: 1 byte  (PathCombineParam))
    GR_END_PATH,
    GR_SET_CLIP_RECT,             (data: 8 bytes (4 swords))
    GR_SET_WIN_CLIP_RECT,         (data: 8 bytes (4 swords))
    GR_CLOSE_SUB_PATH,
    GR_SET_CLIP_PATH,             (data: 1 byte  (flags))
    GR_SET_WIN_CLIP_PATH,         (data: 1 byte  (flags))
    GR_SET_STROKE_PATH                                      */
```

■ GStringErrorType

```
typedef enum /* word */ {
    GSET_NO_ERROR,
    GSET_DISK_FULL
} GStringErrorType;
```

■ GStringKillType

```
typedef ByteEnum GStringKillType;
    #define GSKT_KILL_DATA    0
    #define GSKT_LEAVE_DATA   1
```

Routines

■ GStringSetPosType

```
typedef ByteEnum GStringSetPosType;
    #define GSSPT_SKIP_1      0
    #define GSSPT_RELATIVE    1
    #define GSSPT_BEGINNING   2
    #define GSSPT_END         3
```

■ GStringType

```
typedef ByteEnum GStringType;
    #define GST_CHUNK         0
    #define GST_STREAM        1
    #define GST_VMEM          2
    #define GST_PTR           3
    #define GST_PATH          4
```

■ Handle

```
typedef word Handle;
```

■ HatchDash

```
typedef struct {
    WWFixed      HD_on;
    WWFixed      HD_off;
} HatchDash;
```

■ HatchLine

```
typedef struct {
    PointWWFixed HL_origin;
    WWFixed      HL_deltaX;
    WWFixed      HL_deltaY;
    WWFixed      HL_angle;
    ColorQuad    HL_color;
    word         HL_numDashes;
            /* array of HatchDash structures follows here */
} HatchLine;
```

■ HatchPattern

```
typedef struct {
    word HP_numLines;
    /* array of HatchLine structures follows here */
} HatchPattern;
```

■ HeapAllocFlags

```
typedef ByteFlags HeapAllocFlags;
    #define HAF_ZERO_INIT        0x80
    #define HAF_LOCK             0x40
    #define HAF_NO_ERR           0x20
    #define HAF_UI               0x10
    #define HAF_READ_ONLY        0x08
    #define HAF_OBJECT_RESOURCE  0x04
    #define HAF_CODE             0x02
    #define HAF_CONFORMING       0x01
    #define HAF_STANDARD         (0)
    #define HAF_STANDARD_NO_ERR  (HAF_NO_ERR)
    #define HAF_STANDARD_LOCK    (HAF_LOCK)
    #define HAF_STANDARD_NO_ERR_LOCK (HAF_NO_ERR | HAF_LOCK)
```

■ HeapCongestion

```
typedef enum /* word */ {
    HC_SCRUBBING,
    HC_CONGESTED,
    HC_DESPERATE
} HeapCongestion;
```

■ HeapFlags

```
typedef ByteFlags HeapFlags;
    #define HF_FIXED          0x80
    #define HF_SHARABLE       0x40
    #define HF_DISCARDABLE    0x20
    #define HF_SWAPABLE       0x10
    #define HF_LMEM           0x08
    #define HF_DISCARDED      0x02
    #define HF_SWAPPED        0x01
    #define HF_STATIC         (HF_DISCARDABLE | HF_SWAPABLE)
    #define HF_DYNAMIC        HF_SWAPABLE
```

■ HugeArrayDirectory

```
typedef struct {
    LMemBlockHeader HAD_header;
    VMBlockHandle   HAD_data;
    ChunkHandle     HAD_dir;
    VMBlockHandle   HAD_xdir;
    VMBlockHandle   HAD_self;
    word            HAD_size;
} HugeArrayDirectory;
```

■ IACPConnectFlags

```
typedef WordFlags IACPConnectFlags;
    #define IACPCF_OBEY_LAUNCH_MODEL    0x0020
    #define IACPCF_CLIENT_OD_SPECIFIED  0x0010
    #define IACPCF_FIRST_ONLY           0x0008
    #define IACPCF_SERVER_MODE          0x0007
```

Routines

Include: iacp.goh

■ IACPServerFlags

```
typedef ByteFlags IACPServerFlags;
    #define IACPSF_MULTIPLE_INSTANCES 0x80
```

Include: iacp.goh

■ IACPServerMode

```
typedef ByteEnum IACPServerMode;
    #define IACPSM_NOT_USER_INTERACTIBLE 0
    #define IACPSM_IN_FLUX               1
    #define IACPSM_USER_INTERACTIBLE     2
```

Include: iacp.goh

■ IACPSide

```
typedef enum {
    IACPS_CLIENT,
    IACPS_SERVER
} IACPSide;
```

Include: iacp.goh

■ ImageFlags

```
typedef ByteFlags ImageFlags;
    #define IF_IGNORE_MASK    0x10
    #define IF_BORDER         0x08
    #define IF_BITSIZE        0x07 /* Should hold an ImageBitSize */

    #define IBS_1   0
    #define IBS_2   1
    #define IBS_4   2
    #define IBS_8   3
    #define IBS_16  4
```

■ IMCFeatures

```
typedef ByteFlags IMCFeatures;
    #define IMCF_MAP          0x01
    #define IMC_DEFAULT_FEATURES        IMCF_MAP
    #define IMC_DEFAULT_TOOLBOX_FEATURES0
    #define IMC_MAP_MONIKER_SIZE        1024
```

■ ImpexDataClasses

```
typedef WordFlags ImpexDataClasses;
    #define IDC_TEXT          0x8000
    #define IDC_GRAPHICS      0x4000
    #define IDC_SPREADSHEET   0x2000
    #define IDC_FONT          0x1000
```

■ ImpexFileSelectionData

```
typedef struct {
    FileLongName              IFSD_selection;
    PathName                  IFSD_path;
    word                      IFSD_disk;
    GenFileSelectorEntryFlags IFSD_type;
} ImpexFileSelectionData;
```

■ ImpexMapFlags

```
typedef ByteFlags ImpexMapFlags;
    #define IMF_IMPORT        0x80
    #define IMF_EXPORT        0x40
```

■ ImpexMapFileInfoHeader

```
typedef struct {
    LMemBlockHeader           IMFIH_base;
    word                      IMFIH_fieldChunk;
    word                      IMFIH_numFields;
} ImpexMapFileInfoHeader;
```

■ ImpexTranslationParams

```
typedef struct {
    optr         ITP_impexOD;
    Message      ITP_returnMsg;
    word         ITP_dataClass;
    FileHandle   ITP_transferVMFile;
    VMChain      ITP_transferVMChain;
    dword        ITP_internal;
} ImpexTranslationParams;
```

■ ImportControlAttrs

```
typedef WordFlags ImportControlAttrs;
    #define ICA_IGNORE_INPUT 0x8000  /* ignore input while import occurs */
```

■ ImportControlToolboxFeatures

```
typedef ByteFlags ImportControlToolboxFeatures;
    #define IMPORTCTF_DIALOG_BOX  0x01
```

■ InitFileCharConvert

```
typedef ByteEnum InitFileCharConvert;
    #define IFCC_INTACT       0      /* Leave all characters unchanged. */
```

Routines

```
    #define IFCC_UPCASE       1      /* Make all characters upper case. */
    #define IFCC_DOWNCASE     2      /* Make all characters lower case. */
```

This enumerated type describes how **InitFileRead...()** routines should handle incoming strings.

■ InitFileReadFlags

```
typedef WordFlags InitFileReadFlags;
    #define IFRF_CHAR_CONVERT 0xc000    /* 2 bits: InitFileCharConvert type */
    #define IFRF_READ_ALL     0x2000
    #define IFRF_FIRST_ONLY   0x1000
    #define IFRF_SIZE         0x0fff
```

This record is used with the **InitFileRead...()** routines. The IFRF_CHAR_CONVERT field is used to indicate whether strings being read should be upcased, downcased, or left unaltered—the type is designated by a value of **InitFileCharConvert**. The IFRF_SIZE field is used by routines that take a passed buffer; this field indicates the size of the buffer (the maximum number of bytes that can be returned by the routine).

When setting this record, make sure you shift the IFRF_CHAR_CONVERT value left an offset of IFRF_CHAR_CONVERT_OFFSET.

■ InkBackgroundType

```
typedef enum {
    IBT_NO_BACKGROUND = 0,
    IBT_NARROW_LINED_PAPER = 2,
    IBT_MEDIUM_LINED_PAPER = 4,
    IBT_WIDE_LINED_PAPER = 6,
    IBT_NARROW_STENO_PAPER = 8,
    IBT_MEDIUM_STENO_PAPER = 10,
    IBT_WIDE_STENO_PAPER = 12,
    IBT_SMALL_GRID = 14,
    IBT_MEDIUM_GRID = 16,
    IBT_LARGE_GRID = 18,
    IBT_SMALL_CROSS_SECTION = 20,
    IBT_MEDIUM_CROSS_SECTION = 22,
    IBT_LARGE_CROSS_SECTION = 24,
    IBT_TO_DO_LIST = 26,
    IBT_PHONE_MESSAGE = 28,
    IBT_CUSTOM_BACKGROUND = 30
} InkBackgrountType;
```

This enumerated type is a set of standard background pictures for use with the Ink Database routines.

■ InkControlFeatures

```
typedef ByteFlags InkControlFeatures;
    #define ICF_PENCIL_TOOL   0x02
    #define ICF_ERASER_TOOL   0x01
```

■ InkControlToolboxFeatures

```
typedef ByteFlags InkControlToolboxFeatures;
    #define ICTF_PENCIL_TOOL  0x02
    #define ICTF_ERASER_TOOL  0x01
```

■ InkDBDisplayInfo

```
typedef struct {
    dword   IDBDI_dword1;
    dword   IDBDI_dword2;
    word    IDBDI_word1;
} InkDBDisplayInfo;
```

■ InkDBFrame

```
typedef struct {
    Rectangle IDBF_bounds;              /* bounds of data to save or coord at
                                         * which to load data */
    VMFileHandle IDBF_VMFile;           /* VM File to write to/read from */
    DBGroupAndItem IDBF_DBGroupAndItem; /* DB item to save to/load from */
    word IDBF_DBExtra;                  /* space to skip at start of block */
} InkDBFrame;
```

■ InkFlags

```
typedef ByteFlags InkFlags;
    #define IF_HAS_TARGET             0x20
    #define IF_DIRTY                  0x10
    #define IF_ONLY_CHILD_OF_CONTENT  0x08
    #define IF_CONTROLLED             0x04
    #define IF_INVALIDATE_ERASURES    0x02
    #define IF_HAS_UNDO               0x01
```

■ InkReturnValue

```
typedef enum {
    IRV_NO_REPLY,
    /* VisComp objects use VisCallChildUnderPoint to send
     * MSG_META_QUERY_IF_PRESS_IS_INK to its children, and
     * VisCallChildUnderPoint returns this value (0) if there was not child
     * under the point. No object should actually return this value. */
    IRV_NO_INK,
    /* Return this if the object wants to treat incoming event as mouse data. */
```

Routines

```
    IRV_INK_WITH_STANDARD_OVERRIDE,
    /* Return this if the object normally wants ink (the text object does this),
     * but the user can force mouse events instead by pressing the pen and
     * holding for some user-adjustable amount of time. */
    IRV_WAIT
    /* Return this value if the object under the point is run by a different
     * thread and you want to hold up input (don't do anything with the incoming
     * MSG_META_START_SELECT) 'til obj sends MSG_GEN_APPLICATION_INK_QUERY_REPLY
     * to the applicaiton object. */
} InkReturnValue;
```

This enumerated type is used by objects to let the system know whether incoming pointer events should be interpreted as mouse or pen data.

■ InsertChildFlags

```
typedef WordFlags InsertChildFlags
    #define ICF_MARK_DIRTY    0x8000
    #define ICF_OPTIONS       0x0003
```

This record specifies how children are to be added to an object tree.

■ InsertChildOption

```
typedef ByteEnum InsertChildOption
    #define ICO_FIRST            0
    #define ICO_LAST             1
    #define ICO_BEFORE_REFERENCE  2
    #define ICO_AFTER_REFERENCE   3
```

This enumerated type determines how a child is added and is used with the **InsertChildFlags** record. It has four enumerations, as shown above.

■ InstrumentPatch

```
typedef enum {
    #define IP_ACOUSTIC_GRAND_PIANO 0
    #define IP_BRIGHT_ACOUSTIC_PIANO 1
    #define IP_ELECTRIC_GRAND_PIANO 2
    #define IP_HONKY_TONK_PIANO 3
    #define IP_ELECTRIC_PIANO_1 4
    #define IP_ELECTRIC_PIANO_2 5
    #define IP_HARPSICORD     6
    #define IP_CLAVICORD      7
    #define IP_CELESTA        8

    #define IP_GLOCKENSPIEL   9
    #define IP_MUSIC_BOC      10
    #define IP_VIBRAPHONE     11
    #define IP_MARIMBA        12
    #define IP_XYLOPHONE      13
```

```
#define IP_TUBULAR_BELLS  14
#define IP_DULCIMER       15

#define IP_DRAWBAR_ORGAN  16
#define IP_PERCUSSIVE_ORGAN 17
#define IP_ROCK_ORGAN     18
#define IP_CHURCH_ORGAN   19
#define IP_REED_ORGAN     20
#define IP_ACCORDIAN      21
#define IP_HARMONICA      22
#define IP_TANGO_ACCORDION 23

#define IP_ACOUSTIC_NYLON_GUITAR 24
#define IP_ACOUSTIC_STEEL_GUITAR 25
#define IP_ELECTRIC_JAZZ_GUITAR 26
#define IP_ELECTRIC_CLEAN_GUITAR 27
#define IP_ELECTRIC_MUTED_GUITAR 28
#define IP_OVERDRIVEN_GUITAR 29
#define IP_DISTORTION_GUITAR 30
#define IP_GUITAR_HARMONICS 31

#define IP_ACOUSTIC_BASS  32
#define IP_ELECTRIC_FINGERED_BASS 33
#define IP_ELECTRIC_PICKED_BASS 34
#define IP_FRETLESS_BASS  35
#define IP_SLAP_BASS_1    36
#define IP_SLAP_BASS_2    37
#define IP_SYNTH_BASS_1   38
#define IP_SYNTH_BASS_2   39

#define IP_VIOLIN         40
#define IP_VIOLA          41
#define IP_CELLO          42
#define IP_CONTRABASS     43
#define IP_TREMELO_STRINGS 44
#define IP_PIZZICATO_STRINGS 45
#define IP_ORCHESTRAL_HARP 46
#define IP_TIMPANI        47

#define IP_STRING_ENSAMBLE_1 48
#define IP_STRING_ENSAMBLE_2 49
#define IP_SYNTH_STRINGS_1 50
#define IP_SYNTH_STRINGS_2 51
#define IP_CHIOR_AAHS     52
#define IP_VOICE_OOHS     53
#define IP_SYNTH_VOICE    54
#define IP_ORCHESTRA_HIT  55

#define IP_TRUMPET        56
#define IP_TROMBONE       57
```

Routines

```
#define IP_TUBA            58
#define IP_MUTED_TRUMPET   59
#define IP_FRENCH_HORN     60
#define IP_BRASS_SECTION   61
#define IP_SYNTH_BRASS_1   62
#define IP_SYNTH_BRASS_2   63

#define IP_SOPRANO_SAX     64
#define IP_ALTO_SAX        65
#define IP_TENOR_SAX       66
#define IP_BARITONE_SAX    67
#define IP_OBOE            68
#define IP_ENGLISH_HORN    69
#define IP_BASSOON         70
#define IP_CLARINET        71

#define IP_PICCOLO         72
#define IP_FLUTE           73
#define IP_RECORDER        74
#define IP_PAN_FLUTE       75
#define IP_BLOWN_BOTTLE    76
#define IP_SHAKUHACHI      77
#define IP_WHISTLE         78
#define IP_OCARINA         79

#define IP_LEAD_SQUARE     80
#define IP_LEAD_SAWTOOTH   81
#define IP_LEAD_CALLIOPE   82
#define IP_LEAD_CHIFF      83
#define IP_LEAD_CHARANG    84
#define IP_LEAD_VOICE      85
#define IP_LEAD_FIFTHS     86
#define IP_LEAD_BASS_LEAD  87

#define IP_PAD_NEW_AGE     88
#define IP_PAD_WARM        89
#define IP_PAD_POLYSYNTH   90
#define IP_PAD_CHOIR       91
#define IP_PAD_BOWED       92
#define IP_PAD_METALLIC    93
#define IP_PAD_HALO        94
#define IP_PAD_SWEEP       95

#define IP_FX_RAIN         96
#define IP_FX_SOUNDTRACK   97
#define IP_FX_CRYSTAL      98
#define IP_FX_ATMOSPHERE   99
#define IP_FX_BRIGHTNESS   100
#define IP_FX_GOBLINS      101
#define IP_FX_ECHOES       102
```

```
#define IP_FX_SCI_FI      103

#define IP_SITAR          104
#define IP_BANJO          105
#define IP_SHAMISEN       106
#define IP_KOTO           107
#define IP_KALIMBA        108
#define IP_BAG_PIPE       109
#define IP_FIDDLE         110
#define IP_SHANAI         111

#define IP_TINKLE_BELL    112
#define IP_AGOGO          113
#define IP_STEEL_DRUMS    114
#define IP_WOODBLOCK      115
#define IP_TAIKO_DRUM     116
#define IP_MELODIC_TOM    117
#define IP_SYNTH_DRUM     118
#define IP_REVERSE_CYMBAL 119

#define IP_GUITAR_FRET_NOISE 120
#define IP_BREATH_NOISE   121
#define IP_SEASHORE       122
#define IP_BIRD_TWEET     123
#define IP_TELEPHONE_RING 124
#define IP_HELICOPTER     125
#define IP_APPLAUSE       126
#define IP_GUNSHOT        127

#define IP_ACOUSTIC_BASS_DRUM 128
#define IP_BASS_DRUM_1    129
#define IP_SIDE_STICK     130
#define IP_ACOUSTIC_SNARE 131
#define IP_HAND_CLAP      132
#define IP_ELECTRIC_SNARE 133
#define IP_LOW_FLOOR_TOM  134
#define IP_CLOSED_HI_HAT  135

#define IP_HIGH_FLOOR_TOM 136
#define IP_PEDAL_HI_HAT   137
#define IP_LOW_TOM        138
#define IP_OPEN_HI_HAT    139
#define IP_LOW_MID_TOM    140
#define IP_HI_MID_TOM     141
#define IP_CRASH_CYMBAL_1 142
#define IP_HIGH_TOM       143

#define IP_RIDE_CYMBAL_1  144
#define IP_CHINESE_CYMBAL 145
#define IP_RIDE_BELL      146
```

Routines

```
    #define IP_TAMBOURINE     147
    #define IP_SPLASH_CYMBAL  148
    #define IP_COWBELL        149
    #define IP_CRASH_CYMBAL_2 150
    #define IP_VIBRASLAP      151

    #define IP_RIDE_CYMBAL_2  152
    #define IP_HI_BONGO       153
    #define IP_LOW_BONGO      154
    #define IP_MUTE_HI_CONGA  155
    #define IP_OPEN_HI_CONGA  156
    #define IP_LOW_CONGA      157
    #define IP_HI_TIMBALE     158
    #define IP_LOW_TIMBALE    159

    #define IP_HIGH_AGOGO     160
    #define IP_LOW_AGOGO      161
    #define IP_CABASA         162
    #define IP_MARACAS        163
    #define IP_SHORT_WHISTLE  164
    #define IP_LONG_WHISTLE   165
    #define IP_SHORT_GUIRO    166
    #define IP_LONG_GUIRO     167

    #define IP_CLAVES         168
    #define IP_HI_WOOD_BLOCK  169
    #define IP_LOW_WOOD_BLOCK 170
    #define IP_MUTE_CUICA     171
    #define IP_OPEN_CUICA     172
    #define IP_MUTE_TRIANGLE  173
    #define IP_OPEN_TRIANGLE  174
} InstrumentPatch;
```

These are standard simulated instruments.

■ InstrumentTable

```
typedef enum {
           IT_STANDARD_TABLE=0   /* default table */
} InstrumentTable;
```

The sound library uses this enumerated type to keep track of which table of simulated musical instruments to use.

■ JobStatus

```
typedef struct {
    char            JS_fname[13];    /* std DOS (8.3) spool filename */
    char            JS_parent[FILE_LONGNAME_LENGTH+1];
                                     /* parent app's name */
    char            JS_documentName[FILE_LONGNAME_LENGTH+1];
                                     /* document name */
    word            JS_numPages;     /* # pages in document */
    SpoolTimeStruct JS_time;         /* time spooled */
    byte            JS_printing;     /* TRUE/FALSE if we are printing */
} JobStatus;
```

■ Justification

```
typedef ByteEnum Justification;
    #define J_LEFT   0
    #define J_RIGHT  1
    #define J_CENTER 2
    #define J_FULL   3
```

■ KeyboardShortcut

```
typedef WordFlags KeyboardShortcut;
    #define KS_PHYSICAL        0x8000
    #define KS_ALT             0x4000
    #define KS_CTRL            0x2000
    #define KS_SHIFT           0x1000
    #define KS_CHAR_SET        0x0f00
    #define KS_CHAR            0x00ff
    #define KS_CHAR_SET_OFFSET    8
    #define KS_CHAR_OFFSET        0
```

■ KeyboardType

```
typedef ByteEnum KeyboardType;
    #define KT_NOT_EXTD        1
    #define KT_EXTD            2
    #define KT_BOTH            3
```

■ KeyMapType

```
typedef enum /* word */ {
    KEYMAP_US_EXTD=1,
    KEYMAP_US,
    KEYMAP_UK_EXTD,
    KEYMAP_UK,
    KEYMAP_GERMANY_EXTD,
    KEYMAP_GERMANY,
    KEYMAP_SPAIN_EXTD,
    KEYMAP_SPAIN,
    KEYMAP_DENMARK_EXTD,
    KEYMAP_DENMARK,
    KEYMAP_BELGIUM_EXTD,
    KEYMAP_BELGIUM,
    KEYMAP_CANADA_EXTD,
    KEYMAP_CANADA,
    KEYMAP_ITALY_EXTD,
    KEYMAP_ITALY,
    KEYMAP_LATIN_AMERICA_EXTD,
    KEYMAP_LATIN_AMERICA,
    KEYMAP_NETHERLANDS,
    KEYMAP_NETHERLANDS_EXTD,
    KEYMAP_NORWAY_EXTD,
    KEYMAP_NORWAY,
    KEYMAP_PORTUGAL_EXTD,
    KEYMAP_PORTUGAL,
    KEYMAP_SWEDEN_EXTD,
    KEYMAP_SWEDEN,
    KEYMAP_SWISS_FRENCH_EXTD,
    KEYMAP_SWISS_FRENCH,
    KEYMAP_SWISS_GERMAN_EXTD,
    KEYMAP_SWISS_GERMAN,
    KEYMAP_FRANCE_EXTD,
    KEYMAP_FRANCE,
} KeyMapType;
```

Routines

■ Language

```
typedef ByteEnum Language;
    #define L_DEFAULT         0
    #define L_GRAPHIC         0
    #define L_ENGLISH         1
    #define L_GERMAN          2
    #define L_FRENCH          3
    #define L_SPANISH         4
    #define L_ITALIAN         5
    #define L_DANISH          6
    #define L_DUTCH           7
```

■ LargeMouseData

```
typedef struct {
    PointDWFixed              LMD_location;
    byte                      LMD_buttonInfo;
    UIFunctionsActive         LMD_uiFunctionsActive;
} LargeMouseData;
```

■ LayerPriority

```
typedef ByteEnum LayerPriority;
    #define LAYER_PRIO_MODAL       6
    #define LAYER_PRIO_ON_TOP      8
    #define LAYER_PRIO_STD        12
    #define LAYER_PRIO_ON_BOTTOM  14
```

■ LexicalOrder

```
typedef ByteEnum LexicalOrder;
    #define LEX_SPACE             0x20
    #define LEX_NONBRKSPACE       1
    #define LEX_EXCLAMATION       2
    #define LEX_EXCLAMDOWN        3
    #define LEX_QUOTE             4
    #define LEX_GUILLEDDBLLEFT    5
    #define LEX_GUILLEDDBLRIGHT   6
    #define LEX_GUILSNGLEFT       7
    #define LEX_GUILSNGRIGHT      8
    #define LEX_QUOTEDBLLEFT      9
    #define LEX_QUOTEDBLRIGHT     10
    #define LEX_DBLQUOTELOW       11
    #define LEX_NUMBER_SIGN       12
    #define LEX_DOLLAR_SIGN       13
    #define LEX_PERCENT           14
    #define LEX_AMPERSAND         15
    #define LEX_SNG_QUOTE         16
    #define LEX_QUOTEDSNGLEFT     17
    #define LEX_QUOTEDSNGRIGHT    18
```

```
#define LEX_SNGQUOTELOW      19
#define LEX_LEFT_PAREN       20
#define LEX_RIGHT_PAREN      21
#define LEX_ASTERISK         22
#define LEX_PLUS             23
#define LEX_COMMA            24
#define LEX_MINUS            25
#define LEX_PERIOD           26
#define LEX_SLASH            27
#define LEX_ZERO             28
#define LEX_ONE              29
#define LEX_TWO              30
#define LEX_THREE            31
#define LEX_FOUR             32
#define LEX_FIVE             33
#define LEX_SIX              34
#define LEX_SEVEN            35
#define LEX_EIGHT            36
#define LEX_NINE             37
#define LEX_COLON            38
#define LEX_SEMICOLON        39
#define LEX_LESS_THAN        40
#define LEX_EQUAL            41
#define LEX_GREATER_THAN     42
#define LEX_QUESTION_MARK    43
#define LEX_QUESTIONDOWN     44
#define LEX_AT_SIGN          45
#define LEX_UA               46
#define LEX_UA_ACUTE         47
#define LEX_UA_GRAVE         48
#define LEX_UA_CIRCUMFLEX    49
#define LEX_UA_DIERESIS      50
#define LEX_U_AE             51
#define LEX_UA_TILDE         52
#define LEX_UA_RING          53
#define LEX_LA               54
#define LEX_LA_ACUTE         55
#define LEX_LA_GRAVE         56
#define LEX_LA_CIRCUMFLEX    57
#define LEX_LA_DIERESIS      58
#define LEX_L_AE             59
#define LEX_LA_TILDE         60
#define LEX_LA_RING          61
#define LEX_UB               62
#define LEX_LB               63
#define LEX_UC               64
#define LEX_UC_CEDILLA       65
#define LEX_LC               66
#define LEX_LC_CEDILLA       67
#define LEX_UD               68
```

Routines

```
#define LEX_LD               69
#define LEX_UE               70
#define LEX_UE_ACUTE         71
#define LEX_UE_GRAVE         72
#define LEX_UE_CIRCUMFLEX    73
#define LEX_UE_DIERESIS      74
#define LEX_LE               75
#define LEX_LE_ACUTE         76
#define LEX_LE_GRAVE         77
#define LEX_LE_CIRCUMFLEX    78
#define LEX_LE_DIERESIS      79
#define LEX_UF               80
#define LEX_LF               81
#define LEX_UG               82
#define LEX_LG               83
#define LEX_UH               84
#define LEX_LH               85
#define LEX_UI               86
#define LEX_UI_ACUTE         87
#define LEX_UI_GRAVE         88
#define LEX_UI_CIRCUMFLEX    89
#define LEX_UI_DIERESIS      90
#define LEX_LI               91
#define LEX_LI_ACUTE         92
#define LEX_LI_GRAVE         93
#define LEX_LI_CIRCUMFLEX    94
#define LEX_LI_DIERESIS      95
#define LEX_LI_DOTLESS       96
#define LEX_UJ               97
#define LEX_LJ               98
#define LEX_UK               99
#define LEX_LK               100
#define LEX_UL               101
#define LEX_LL               102
#define LEX_UM               103
#define LEX_LM               104
#define LEX_UN               105
#define LEX_UN_TILDE         106
#define LEX_LN               107
#define LEX_LN_TILDE         108
#define LEX_UO               109
#define LEX_UO_ACUTE         110
#define LEX_UO_GRAVE         111
#define LEX_UO_CIRCUMFLEX    112
#define LEX_UO_DIERESIS      113
#define LEX_U_OE             114
#define LEX_UO_TILDE         115
#define LEX_UO_SLASH         116
#define LEX_LO               117
#define LEX_LO_ACUTE         118
```

```
#define LEX_LO_GRAVE          119
#define LEX_LO_CIRCUMFLEX     120
#define LEX_LO_DIERESIS       121
#define LEX_L_OE              122
#define LEX_LO_TILDE          123
#define LEX_LO_SLASH          124
#define LEX_UP                125
#define LEX_LP                126
#define LEX_UQ                127
#define LEX_LQ                128
#define LEX_UR                129
#define LEX_LR                130
#define LEX_US                131
#define LEX_LS                132
#define LEX_GERMANDBLS        133
#define LEX_UT                134
#define LEX_LT                135
#define LEX_UU                136
#define LEX_UU_ACUTE          137
#define LEX_UU_GRAVE          138
#define LEX_UU_CIRCUMFLEX     139
#define LEX_UU_DIERESIS       140
#define LEX_LU                141
#define LEX_LU_ACUTE          142
#define LEX_LU_GRAVE          143
#define LEX_LU_CIRCUMFLEX     144
#define LEX_LU_DIERESIS       145
#define LEX_UV                146
#define LEX_LV                147
#define LEX_UW                148
#define LEX_LW                149
#define LEX_UX                150
#define LEX_LX                151
#define LEX_UY                152
#define LEX_UY_ACUTE          153
#define LEX_UY_DIERESIS       154
#define LEX_LY                155
#define LEX_LY_ACUTE          156
#define LEX_LY_DIERESIS       157
#define LEX_UZ                158
#define LEX_LZ                159
#define LEX_LEFT_BRACKET      160
#define LEX_BACKSLASH         161
#define LEX_RIGHT_BRACKET     162
#define LEX_ASCII_CIRCUMFLEX  163
#define LEX_UNDERSCORE        164
#define LEX_BACKQUOTE         165
#define LEX_LEFT_BRACE        166
#define LEX_VERTICAL_BAR      167
#define LEX_RIGHT_BRACE       168
```

Routines

```
#define LEX_ASCII_TILDE       169
#define LEX_DELETE            170
#define LEX_DAGGER            171
#define LEX_DBLDAGGER         172
#define LEX_DEGREE            173
#define LEX_CENT              174
#define LEX_STERLING          175
#define LEX_CURRENCY          176
#define LEX_YEN               177
#define LEX_SECTION           178
#define LEX_BULLET            179
#define LEX_DIAMONDBULLET     180
#define LEX_PARAGRAPH         181
#define LEX_REGISTERED        182
#define LEX_COPYRIGHT         183
#define LEX_TRADEMARK         184
#define LEX_NOTEQUAL          185
#define LEX_INFINITY          186
#define LEX_PLUSMINUS         187
#define LEX_LESSEQUAL         188
#define LEX_GREATEREQUAL      189
#define LEX_APPROX_EQUAL      190
#define LEX_L_MU              191
#define LEX_L_DELTA           192
#define LEX_U_SIGMA           193
#define LEX_U_PI              194
#define LEX_L_PI              195
#define LEX_INTEGRAL          196
#define LEX_ORDFEMININE   197
#define LEX_ORDMASCULINE  198
#define LEX_U_OMEGA       199
#define LEX_LOGICAL_NOT   200
#define LEX_ROOT          201
#define LEX_FLORIN        202
#define LEX_U_DELTA       203
#define LEX_ELLIPSIS      204
#define LEX_ENDASH        205
#define LEX_EMDASH        206
#define LEX_DIVISION      207
#define LEX_FRACTION      208
#define LEX_CNTR_DOT      209
#define LEX_PERTHOUSAND   210
#define LEX_LOGO          211
#define LEX_ACUTE         212
#define LEX_DIERESIS      213
#define LEX_CIRCUMFLEX    214
#define LEX_TILDE         215
#define LEX_MACRON        216
#define LEX_BREVE         217
#define LEX_DOTACCENT     218
```

```
    #define LEX_RING          219
    #define LEX_CEDILLA       220
    #define LEX_HUNGARUMLAT   221
    #define LEX_OGONEK        222
    #define LEX_CARON         223
```

■ LexFirstOrder

```
typedef ByteEnum Lex1stOrder;
    #define LEX1_SPACE           0x20
    #define LEX1_EXCLAMATION     1
    #define LEX1_QUOTE           2
    #define LEX1_NUMBER_SIGN     3
    #define LEX1_DOLLAR_SIGN     4
    #define LEX1_PERCENT         5
    #define LEX1_AMPERSAND       6
    #define LEX1_SNG_QUOTE       7
    #define LEX1_PARENTHESIS     8
    #define LEX1_ASTERISK        9
    #define LEX1_PLUS            10
    #define LEX1_COMMA           11
    #define LEX1_MINUS           12
    #define LEX1_PERIOD          13
    #define LEX1_SLASH           14
    #define LEX1_ZERO            15
    #define LEX1_ONE             16
    #define LEX1_TWO             17
    #define LEX1_THREE           18
    #define LEX1_FOUR            19
    #define LEX1_FIVE            20
    #define LEX1_SIX             21
    #define LEX1_SEVEN           22
    #define LEX1_EIGHT           23
    #define LEX1_NINE            24
    #define LEX1_COLON           25
    #define LEX1_SEMICOLON       26
    #define LEX1_LESS_THAN       27
    #define LEX1_EQUAL           28
    #define LEX1_GREATER_THAN    29
    #define LEX1_QUESTION_MARK   30
    #define LEX1_AT_SIGN         31
    #define LEX1_A               32
    #define LEX1_B               33
    #define LEX1_C               34
    #define LEX1_D               35
    #define LEX1_E               36
    #define LEX1_F               37
    #define LEX1_G               38
    #define LEX1_H               39
    #define LEX1_I               40
```

Routines

```
#define LEX1_J               41
#define LEX1_K               42
#define LEX1_L               43
#define LEX1_M               44
#define LEX1_N               45
#define LEX1_O               46
#define LEX1_P               47
#define LEX1_Q               48
#define LEX1_R               49
#define LEX1_S               50
#define LEX1_T               51
#define LEX1_U               52
#define LEX1_V               53
#define LEX1_W               54
#define LEX1_X               55
#define LEX1_Y               56
#define LEX1_Z               57
#define LEX1_LEFT_BRACKET    58
#define LEX1_BACKSLASH       59
#define LEX1_RIGHT_BRACKET   60
#define LEX1_ASCII_CIRCUMFLEX 61
#define LEX1_UNDERSCORE      62
#define LEX1_BACKQUOTE       63
#define LEX1_LEFT_BRACE      64
#define LEX1_VERTICAL_BAR    65
#define LEX1_RIGHT_BRACE     66
#define LEX1_ASCII_TILDE     67
#define LEX1_ASCII_DELETE    68
#define LEX1_DAGGER          69
#define LEX1_DEGREE          70
#define LEX1_CENT            71
#define LEX1_STERLING        72
#define LEX1_SECTION         73
#define LEX1_BULLET          74
#define LEX1_PARAGRAPH       75
#define LEX1_REGISTERED      76
#define LEX1_COPYRIGHT       77
#define LEX1_TRADEMARK       78
#define LEX1_ACUTE           79
#define LEX1_DIERESIS        80
#define LEX1_NOTEQUAL        81
#define LEX1_INFINITY        82
#define LEX1_PLUSMINUS       83
#define LEX1_LESSEQUAL       84
#define LEX1_GREATEREQUAL    85
#define LEX1_YEN             86
#define LEX1_MU              87
#define LEX1_DELTA           88
#define LEX1_SIGMA           89
#define LEX1_PI              90
```

```
#define LEX1_INTEGRAL         91
#define LEX1_ORDFEMININE      92
#define LEX1_ORDMASCULINE     93
#define LEX1_OMEGA            94
#define LEX1_QUESTIONDOWN     95
#define LEX1_EXCLAMDOWN       96
#define LEX1_LOGICALNOT       97
#define LEX1_ROOT             98
#define LEX1_FLORIN           99
#define LEX1_APPROX_EQUAL     100
#define LEX1_ELLIPSIS         101
#define LEX1_ENDASH           102
#define LEX1_EMDASH           103
#define LEX1_DIVISION         104
#define LEX1_DIAMONDBULLET    105
#define LEX1_FRACTION         106
#define LEX1_CURRENCY         107
#define LEX1_CNTR_DOT         108
#define LEX1_PERTHOUSAND      109
#define LEX1_LOGO             110
#define LEX1_CIRCUMFLEX       111
#define LEX1_TILDE            112
#define LEX1_MACRON           113
#define LEX1_BREVE            114
#define LEX1_DOTACCENT        115
#define LEX1_RING             116
#define LEX1_CEDILLA          117
#define LEX1_HUNGARUMLAT      118
#define LEX1_OGONEK           119
#define LEX1_CARON            120
```

■ LibraryCallType

```
typedef enum /* word */ {
    LCT_ATTACH,               /* The library was just loaded. */
    LCT_DETACH,               /* The library is about to be unloaded. */
    LCT_NEW_CLIENT,           /* A new client of the library was just loaded. */
    LCT_NEW_CLIENT_THREAD,    /* A new thread was just created for a
                               * current client of the library. */
    LCT_CLIENT_THREAD_EXIT,   /* A thread was just exited for a current
                               * client of the library. */
    LCT_CLIENT_EXIT,          /* Library's client is about to be unloaded. */
} LibraryCallType
```

This type is used by library entry point routines. Library entry point routines take a value of this enumerated type to determine what, if anything, is to be done.

Routines

■ LineAttr

```
typedef struct {
    byte         LA_colorFlag;
    RGBValue     LA_color;
    SysDrawMask  LA_mask;
    ColorMapMode LA_mapMode;
    LineEnd      LA_end;
    LineJoin     LA_join;
    LineStyle    LA_style;
    WWFixed      LA_width;
} LineAttr;
```

■ LineEnd

```
typedef ByteEnum LineEnd;
    #define LE_BUTTCAP         0
    #define LE_ROUNDCAP        1
    #define LE_SQUARECAP       2
    #define LAST_LINE_END_TYPE LE_SQUARECAP
```

Line ends determine how the graphics system will draw the end of a line segment.

■ LineJoin

```
typedef ByteEnum LineJoin;
    #define LJ_MITERED          0
    #define LJ_ROUND            1
    #define LJ_BEVELED          2
    #define LAST_LINE_JOIN_TYPE LJ_BEVELED
```

This enumerated type determines how the graphics system will draw corners of rectangles and polylines.

■ LineStyle

```
typedef ByteEnum LineStyle;
    #define LS_SOLID             0
    #define LS_DASHED            1
    #define LS_DOTTED            2
    #define LS_DASHDOT           3
    #define LS_DASHDDOT          4
    #define LS_CUSTOM            5
    #define MAX_DASH_ARRAY_PAIRS 5
```

The **LineStyle** type describes a line's "dottedness." Lines using custom dashes will work with the **DashPairArray** structure.

■ LMemBlockHeader

```
typedef struct {
    MemHandle    LMBH_handle;
    word         LMBH_offset;
    word         LMBH_flags;
    LMemTypes    LMBH_lmemType;
    word         LMBH_blockSize;
    word         LMBH_nHandles;
    word         LMBH_freeList;
    word         LMBH_totalFree;
} LMemBlockHeader;
```

This structure is found at the beginning of every block which contains an LMem heap. You can examine any of the fields by locking the block and casting its address to a ***LMemBlockHeader**. You should not, however, change any of the fields yourself; they are managed by the LMem routines.

Contents: The header has the following fields:

LMBH_handle
The handle of this block.

LMBH_offset
The offset from the beginning of the block to the beginning of the heap.

LMBH_flags
The **LocalMemoryFlags** currently set for the block. The flags are described in the entry for **LMemInitHeap()**.

LMBH_lmemType
The type of LMem heap in this block. This field is a member of the **LMemType** enumerated type, described in the entry for **LMemInitHeap()**.

LMBH_blockSize
The total size of this block. This size may change in either direction as a result of chunk allocation and heap compaction.

LMBH_nHandles
The number of handles available in the chunk handle table. Not all of these chunks are necessarily allocated as owned or free chunks. The table grows automatically when necessary.

LMBH_freeList
The chunk handle of the first free chunk in the linked list of free chunks.

Routines

LMBH_totalFree
The total amount of free space in the LMem heap.

Warnings: Do not change the settings of the **LMemBlockHeader**. They are automatically maintained by the LMem routines.

Include: lmem.h

See Also: LMemInitHeap()

■ LMemType

```
typdef enum {
    LMEM_TYPE_GENERAL,
    LMEM_TYPE_WINDOW,
    LMEM_TYPE_OBJ_BLOCK,
    LMEM_TYPE_GSTATE,
    LMEM_TYPE_FONT_BLK,
    LMEM_TYPE_GSTRING,
    LMEM_TYPE_DB_ITEMS
} LMemType;
```

LMem heaps are created for many different purposes. Some of these purposes require the heap to have special functionality. For this reason, when you create an LMem heap, you must specify what it will be used for. The following types are available:

LMEM_TYPE_GENERAL
The LMem heap will be used for general data storage, possibly including a chunk, name, or element array. When an application creates an LMem heap, it will almost always be of type “General” or “Object.”

LMEM_TYPE_OBJ_BLOCK
Objects are stored in object blocks, which are LMem heaps. An object block has some extra header information and contains one chunk which contains only flags. All the objects in the block are stored as chunks on the heap. Applications can directly create object blocks.

LMEM_TYPE_WINDOW
Windows are stored in memory as LMem heaps. The header contains information about the window; each region in the window is stored as a chunk. Applications will not directly create Window heaps.

LMEM_TYPE_GSTATE
A GState is an LMem heap. The GState information is in the header, and the application clip-rectangle is stored in a chunk.

Applications do not directly create GState blocks; rather, they call a GState creation routine, which creates the block.

LMEM_TYPE_FONT_BLOCK
: Font blocks are stored as LMem heaps. Applications do not create font blocks directly.

LMEM_TYPE_GSTRING
: Whenever a GString is created or loaded, a GString LMem heap is created, and elements are added as chunks. The heap is created automatically by the GString routines; applications should not create GString blocks.

LMEM_TYPE_DB_ITEMS
: The Virtual Memory mechanism provides routines to create and manage database items, short pieces of data which are dynamically allocated and are saved with the VM file. Applications do not directly allocate DB blocks; rather, they call DB routines, which see to it that the blocks are created.

Include: lmem.h

■ LocalDistanceFlags

```
typedef WordFlags LocalDistanceFlags;
    #define LDF_FULL_NAMES              0x8000
    #define LDF_PRINT_PLURAL_IF_NEEDED  0x4000
```

■ LocalCmpStringsDosToGeosFlags

```
typedef ByteFlags LocalCmpStringsDosToGeosFlags;
    #define LCSDTG_NO_CONVERT_STRING_2      0x02
    #define LCSDTGF_NO_CONVERT_STRING_1     0x01
```

■ LocalCurrencyFormat

```
typedef struct {
    byte    CurrencyFormatFlags;
    byte    currencyDigits;
    word    thousandsSeparator;
    word    decimalSeparator;
    word    listSeparator;
} LocalCurrencyFormat;
```

■ LocalMemoryFlags

```
typedef WordFlags LocalMemoryFlags;
    #define LMF_HAS_FLAGS     0x8000
    #define LMF_IN_RESOURCE   0x4000
    #define LMF_DETACHABLE    0x2000
```

Routines

```
#define LMF_DUPLICATED    0x1000
#define LMF_RELOCATED     0x0800
#define LMF_AUTO_FREE     0x0400
#define LMF_IN_LMEM_ALLOC 0x0200
#define LMF_IS_VM         0x0100
#define LMF_NO_HANDLES    0x0080
#define LMF_NO_ENLARGE    0x0040
#define LMF_RETURN_ERRORS 0x0020
#define LMF_DEATH_COUNT   0x0007
#define STD_LMEM_OBJECT_FLAGS    (LMF_HAS_FLAGS | LMF_RELOCATED)
```

When an LMem heap is allocated, certain flags are passed to indicate properties the heap should have. Some of the flags are passed only for system-created heaps. The flags are stored in a word-length record (**LocalMemoryFlags**); the record also contains flags indicating the current state of the heap. The **LocalMemoryFlags** are listed below:

LMF_HAS_FLAGS
: Set if the block has a chunk containing only flags. This flag is set for object blocks; it is usually cleared for general LMem heaps.

LMF_IN_RESOURCE
: Set if the block has just been loaded from a resource and has not been changed since being loaded. This flag is set only for object blocks created by the compiler.

LMF_DETACHABLE
: Set if the block is an object block which can be saved to a state file.

LMF_DUPLICATED
: Set if block is an object block created by the **ObjDuplicateBlock()** routine. This flag should not be set by applications.

LMF_RELOCATED
: Set if all the objects in the block have been relocated. The object system sets this when it has relocated all the objects in the block.

LMF_AUTO_FREE
: This flag is used by several object routines. It indicates that if the block's in-use count drops to zero, the block may be freed. This flag should not be set by applications.

LMF_IN_MEM_ALLOC
: This flag is used in error-checking code to prevent the heap

from being validated while a chunk is being allocated. For internal use only—do not modify.

LMF_IS_VM
: Set if the LMem heap is in a VM block and the block should be marked dirty whenever a chunk is marked dirty. This flag is automatically set by the VM code when an LMem heap is created in or attached to a VM file. This flag should not be set by applications.

LMF_NO_HANDLES
: Set if block does not use chunk handles. A block can be set to simulate the C **malloc()** routine; in this case, chunks are not relocated after being created, so chunk handles are not needed. Ordinarily, these blocks are created by the **malloc()** routine, not by applications.

LMF_NO_ENLARGE
: Indicates that the local-memory routines should not enlarge this block to fulfill chunk requests. This guarantees that the block will not be moved by a chunk allocation request; however, it makes these requests more likely to fail.

LMF_RETURN_ERRORS
: Set if local memory routines should return errors when allocation requests cannot be fulfilled. If the flag is not set, allocation routines will fatal-error if they cannot comply with requests. This flag is generally clear for expandable LMem blocks, since many system routines (such as **ObjInstantiate()**) are optimized in such a way that they cannot deal with LMem allocation errors.

LMF_DEATH_COUNT
: This field occupies the least significant three bits of the flag field. It means nothing if the value is zero. If it is non-zero, it indicates the number of remove-block messages left which must hit **BlockDeathCommon** before it will free the block. This flag is used by the handlers for MSG_FREE_DUPLICATE and MSG_REMOVE_BLOCK.

STD_LMEM_OBJ_FLAGS
: This is a constant which combines the LMF_HAS_FLAGS and LMF_RELOCATED flags. These flags should be set for all object blocks.

Include: lmem.h

Routines

■ LocalNumericFormat

```
typedef struct {
    byte    numberFormatFlags;
    byte    decimalDigits;
    word    thousandsSeparator;
    word    decimalSeparator;
    word    listSeparator;
} LocalNumericFormat;
```

■ LocalQuotes

```
typedef struct {
    word    frontSingle;
    word    endSingle;
    word    frontDouble;
    word    endDouble;
} LocalQuotes;
```

■ ManufacturerID

```
typedef enum { /* word */
    MANUFACTURER_ID_GEOWORKS
} ManufacturerID;
```

■ MapColorToMono

```
typedef ByteEnum MapColorToMono;
    #define CMT_CLOSEST       0
    #define CMT_DITHER        1
```

This type determines what the graphics system will do when trying to draw in an unavailable color. It will either draw in the closest color, or else mix two or more close colors to get as close as possible overall.

■ MapListBlockHeader

```
typedef struct {
    LMemBlockHeader         MLBH_base;
    word                    MLBH_numDestFields;
    word                    MLBH_chunk1;
} MapListBlockHeader;
```

■ MarginDimensions

```
typedef struct {
    int     leftMargin;
    int     topMargin;
    int     rightMargin;
    int     bottomMargin;
} MarginDimensions;
```

■ MeasurementType

```
typedef ByteEnum MeasurementType;
    #define MEASURE_US        0
    #define MEASURE_METRIC    1
```

■ MediaType

```
typedef enum /* byte */ {
    #define MEDIA_NONEXISTENT 0
    #define MEDIA_160K        1
    #define MEDIA_180K        2
    #define MEDIA_320K        3
    #define MEDIA_360K        4
    #define MEDIA_720K        5
    #define MEDIA_1M2         6
    #define MEDIA_1M44        7
    #define MEDIA_2M88        8
    #define MEDIA_FIXED_DISK  9
    #define MEDIA_CUSTOM      10
} MediaType;
```

The **MediaType** enumerated type indicates how a disk is formatted. A member of this enumerated type is returned by some disk-information routines (e.g. **DriveGetDefaultMedia()**). A **MediaType** value is also passed to **DiskFormat()**, indicating how the disk should be formatted.

■ MemGetInfoType

```
typedef enum /* word */ {
    MGIT_SIZE=0,                  /* size in bytes */
    MGIT_FLAGS_AND_LOCK_COUNT=2,  /* use MGI_LOCK_COUNT and MGI_FLAGS */
    MGIT_OWNER_OR_VM_FILE_HANDLE=4,
    MGIT_ADDRESS=6,
    MGIT_OTHER_INFO=8,
    MGIT_EXEC_THREAD=10
} MemGetInfoType;
```

■ MemHandle

```
typedef Handle MemHandle;
```

■ Message

```
typedef word Message;
```

Routines

■ MessageError

```
typedef enum /* word */ {
     MESSAGE_NO_ERROR,         /* no error was encountered */
     MESSAGE_NO_HANDLES        /* no handle could be allocated
                                * and MF_CAN_DISCARD_IF_DESPARATE
                                * was passed */
} MessageErrors;
```

A **MessageError** is returned by **ObjMessage()** in assembly to indicate whether the message was successfully sent. This is not encountered by C applications.

■ MessageFlags

```
typedef WordFlags MessageFlags;
    #define MF_CALL                       0x8000    /* @call */
    #define MF_FORCE_QUEUE                0x4000
    #define MF_STACK                      0x2000    /* @stack */
    #define MF_CHECK_DUPLICATE            0x0800
    #define MF_CHECK_LAST_ONLY            0x0400
    #define MF_REPLACE                    0x0200
    #define MF_CUSTOM                     0x0100
    #define MF_FIXUP_DS                   0x0080
    #define MF_FIXUP_ES                   0x0040
    #define MF_DISCARD_IF_NO_MATCH        0x0020
    #define MF_MATCH_ALL                  0x0010
    #define MF_INSERT_AT_FRONT            0x0008    /* puts at front of queue */
    #define MF_CAN_DISCARD_IF_DESPERATE   0x0004
    #define MF_RECORD                     0x0002    /* @record */
    #define MF_DISPATCH_DONT_FREE         0x0002
```

MessageFlags are specified in the assembly version of **ObjMessage()**. Most of these flags are set properly by Goc and the kernel in C. See the reference entries for the Goc keywords **@call** and **@send**.

■ MessageHandle

```
typedef Handle MessageHandle;
```

■ MessageMethod

```
typedef void MessageMethod();
```

■ MinIncrementType

```
typedef union {
    MinUSMeasure              MIT_US;
    MinMetricMeasure          MIT_METRIC;
    MinPointMeasure           MIT_POINT;
    MinPicaMeasure            MIT_PICA;
} MinIncrementType;
```

■ MinMetricMeasure

```
typedef ByteEnum MinMetricMeasure;
    #define MMM_MILLIMETER       0
    #define MMM_HALF_CENTIMETER  1
    #define MMM_CENTIMETER       2
```

■ MinPicaMeasure

```
typedef ByteEnum MinPicaMeasure;
    #define MPM_PICA             0
    #define MPM_INCH             1
```

■ MinPointMeasure

```
typedef ByteEnum MinPointMeasure;
    #define MPM_25_POINT         0
    #define MPM_50_POINT         1
    #define MPM_100_POINT        2
```

■ MinUSMeasure

```
typedef ByteEnum MinUSMeasure;
    #define MUSM_EIGHTH_INCH     0
    #define MUSM_QUARTER_INCH    1
    #define MUSM_HALF_INCH       2
    #define MUSM_ONE_INCH        3
```

■ MixMode

```
typedef ByteEnum MixMode;
    #define MM_CLEAR0         /* clear destination */
    #define MM_COPY 1         /* new drawing is opaque */
    #define MM_NOP  2         /* no drawing */
    #define MM_AND  3         /* logical AND of new and old colors */
    #define MM_INVERT 4       /* inverse of old color */
    #define MM_XOR  5         /* XOR of new and old colors */
    #define MM_SET  6         /* set destination black */
    #define MM_OR   7         /* logical OR of new and old colors */
```

The **MixMode** determines what the graphics system will do when drawing one thing on top of another.

■ MonoTransfer

```
typedef struct {
    byte MT_gray[256];
} MonoTransfer;
```

Routines

■ MouseReturnFlags

```
typedef WordFlags MouseReturnFlags;
    #define MRF_PROCESSED                0x8000
    #define MRF_REPLAY                   0x4000
    #define MRF_PREVENT_PASS_THROUGH     0x2000
    #define MRF_SET_POINTER_IMAGE        0x1000
    #define MRF_CLEAR_POINTER_IMAGE      0x0800
```

These flags are used in various parts of the system that work with mouse input. Which values are appropriate to pass will vary based on context.

■ MouseReturnParams

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

This structure is used in certain areas of the system which work with mouse input.

■ NameArrayAddFlags

```
typedef WordFlags NameArrayAddFlags;
    #define NAAF_SET_DATA_ON_REPLACE 0x8000
```

■ NameArrayElement

```
typedef struct {
    RefElementHeader NAE_meta;
} NameArrayElement;
```

■ NameArrayHeader

```
typedef struct{
    ElementArrayHeader          NAH_meta;
    word                        NAH_dataSize;     /* Size of data section of
                                                   * each element */
} NameArrayHeader;
```

Every name array must begin with a **NameArrayHeader**. Since name arrays are special kinds of element arrays, the **NameArrayHeader** must itself begin with an **ElementArrayHeader**. The structure contains one additional field, *NAH_dataSize*. This field specifies how long the data section of every element is. Applications may examine this field, but they must not change it.

■ NameArrayMaxElement

```
typedef struct {
    RefElementHeader NAME_meta;
    byte NAME_data[NAME_ARRAY_MAX_DATA_SIZE];
    char NAME_name[NAME_ARRAY_MAX_NAME_SIZE];
} NameArrayMaxElement;
```

■ NO_ERROR_RETURNED

```
#define NO_ERROR_RETURNED     0
```

■ NoteType

```
typedef ByteEnum NoteType;
    #define NT_INK  0
    #define NT_TEXT 1
```

■ NotificationType

```
typedef struct {
    ManufacturerID  NT_manuf;
    word            NT_type;
} NotificationType;
```

■ NotifyInkHasTarget

```
typedef struct {
    optr    NIHT_optr;
} NotifyInkHasTarget;
```

■ NULL

```
#undef NULL
#define NULL       0
```

■ NullChunk

```
#define NullChunk  ((ChunkHandle) 0)
```

■ NullClass

```
#define NullClass  ((ClassStruct *) 0)
```

■ NullHandle

```
#define NullHandle ((Handle) 0)
```

■ NullOptr

```
#define NullOptr   ((optr) 0)
```

Routines

■ NumberFormatFlags

```
typedef ByteFlags NumberFormatFlags;
    #define NFF_LEADING_ZERO  0x01
```

■ NumberType

```
typedef ByteEnum NumberType;
    #define NT_VALUE          0
    #define NT_BOOLEAN        1
    #define NT_DATE_TIME      2
```

■ ObjChunkFlags

```
typedef ByteFlags ObjChunkFlags;
    #define OCF_VARDATA_RELOC     0x10
    #define OCF_DIRTY             0x08
    #define OCF_IGNORE_DIRTY      0x04
    #define OCF_IN_RESOURCE       0x02
    #define OCF_IS_OBJECT         0x01
```

This record is stored at the beginning of each chunk and gives specific information about the chunk. The flags are internal.

■ ObjLMemBlockHeader

```
typedef struct {
    LMemBlockHeader OLMBH_header;          /* standard LMem block header */
    word            OLMBH_inUseCount;
    word            OLMBH_interactibleCount;
    optr            OLMBH_output;
    word            OLMBH_resourceSize;
} ObjLMemBlockHeader;
```

This is the standard Object Block header that begins every object block; you can set additional header fields with the **@header** Goc keyword. The fields of this structure are

OLMBH_header
The standard LMem block header. See the **LMemBlockHeader** structure type.

OLMBH_inUseCount
The "in use" count for the block. If not zero, then the block may not safely be freed.

OLMBH_interactibleCount
The "interactable" count for the block. If not zero, then one or more objects in the block are either visible to the user or about

to be activated by the user (e.g. via keyboard shortcut). A block with a non-zero interactible count may not be swapped.

OLMBH_output
The optr of the object that will be notified about changes in resource status, such as in-use count changing to or from zero. Messages may also be sent to this output object via the **TravelOption** TO_OBJ_BLOCK_OUTPUT.

OLMBH_resourceSize
The size of the object block (resource).

■ ObjRelocation

```
typedef struct {
    ObjRelocationType        OR_type;
    word                     OR_offset;
} ObjRelocation;
```

■ ObjRelocationSource

```
typedef ByteEnum ObjRelocationSource;
    #define ORS_NULL                         0
    #define ORS_OWNING_GEODE                 1
    #define ORS_KERNEL                       2
    #define ORS_LIBRARY                      3
    #define ORS_CURRENT_BLOCK                4
    #define ORS_VM_HANDLE                    5
    #define ORS_OWNING_GEODE_ENTRY_POINT     6
    #define ORS_NON_STATE_VM                 7
    #define ORS_UNKNOWN_BLOCK                8
    #define ORS_EXTERNAL                     9
    #define RID_SOURCE_OFFSET                12
```

■ ObjRelocationType

```
typedef ByteEnum ObjRelocationType;
    #define RELOC_END_OF_LIST         0
    #define RELOC_RELOC_HANDLE        1
    #define RELOC_RELOC_SEGMENT       2
    #define RELOC_RELOC_ENTRY_POINT   3
```

■ OperatorStackElement

```
typedef struct {
    EvalStackOperatorType OSE_type;
    EvalStackOperatorType OSE_data;
} OperatorStackElement;
```

Routines

■ OperatorType

```
typedef ByteEnum OperatorType;
    #define OP_RANGE_SEPARATOR                  0
    #define OP_NEGATION                         1
    #define OP_PERCENT                          2
    #define OP_EXPONENTIATION                   3
    #define OP_MULTIPLICATION                   4
    #define OP_DIVISION                         5
    #define OP_MODULO                           6
    #define OP_ADDITION                         7
    #define OP_SUBTRACTION                      8
    #define OP_EQUAL                            9
    #define OP_NOT_EQUAL                        10
    #define OP_LESS_THAN                        11
    #define OP_GREATER_THAN                     12
    #define OP_LESS_THAN_OR_EQUAL               13
    #define OP_GREATER_THAN_OR_EQUAL            14
    #define OP_STRING_CONCAT                    15
    #define OP_RANGE_INTERSECTION               16
    #define OP_NOT_EQUAL_GRAPHIC                17
    #define OP_DIVISION_GRAPHIC                 18
    #define OP_LESS_THAN_OR_EQUAL_GRAPHIC       19
    #define OP_GREATER_THAN_OR_EQUAL_GRAPHIC    20
    #define OP_PERCENT_MODULO                   21
    #define OP_SUBTRACTION_NEGATION             22
```

■ optr

```
typedef dword optr;
```

■ PageLayout

```
typedef union {
    PageLayoutPaper          PL_paper;
    PageLayoutEnvelope       PL_envelope;
    PageLayoutLabel          PL_label;
} PageLayout;
```

■ PageLayoutEnvelope

```
typedef WordFlags PageLayoutEnvelope;
    #define PLE_PATH           0x0040
    #define PLE_ORIENTATION    0x0010
    #define PLE_TYPE           0x0004
```

■ PageLayoutLabel

```
typedef WordFlags PageLayoutLabel;
    #define PLL_ROWS           0x7e00    /* labels down */
    #define PLL_COLUMNS        0x01f8    /* labels across */
    #define PLL_TYPE          0x0004    /* PT_LABEL */
```

■ PageLayoutPaper

```
typedef WordFlags PageLayoutPaper;
    #define PLP_ORIENTATION   0x0008
    #define PLP_TYPE          0x0004
```

■ PageSize

```
typedef struct {
    word         unused;
    word         PS_width;
    word         PS_height;
    PageLayout   PS_layout;
} PageSize;
```

■ PageSizeCtrlAttrs

```
typedef WordFlags PageSizeCtrlAttrs;
    #define PZCA_ACT_LIKE_GADGET  0x8000
    #define PZCA_PAPER_SIZE       0x4000
    #define PZCA_INITIALIZE       0x2000
```

■ PageSizeCtrlFeatures

```
typedef ByteFlags PageSizeControlFeatures;
    #define PSIZECF_MARGINS   0x04
    #define PSIZECF_ALL       0x02
    #define PSIZECF_PAGE_TYPE 0x01
```

■ PageSizeReport

```
typedef struct {
    dword           PSR_width;
    dword           PSR_height;
    PageLayout      PSR_layout;
    PCMarginParams  PSR_margins;
} PageSizeReport:
```

■ PageType

```
typedef enum {
    PT_PAPER,
    PT_ENVELOPE,
    PT_LABEL
} PageType;
```

■ PaperOrientation

```
typedef ByteEnum PaperOrientation;
    #define PO_PORTRAIT       0x00
```

Routines

```
    #define PO_LANDSCAPE      0x01
```

■ ParallelUnit

```
typedef  enum
    {
            PARALLEL_LPT1= 0,
            PARALLEL_LPT2= 2,
            PARALLEL_LPT3= 4,
            PARALLEL_LPT4= 6,
    } ParallelUnit;
```

■ ParserFlags

```
typedef ByteFlags ParserFlags;
    #define PF_HAS_LOOKAHEAD         0x80
    #define PF_CONTAINS_DISPLAY_FUNC 0x40
    #define PF_OPERATORS             0x20
    #define PF_NUMBERS               0x10
    #define PF_CELLS                 0x08
    #define PF_FUNCTIONS             0x04
    #define PF_NAMES                 0x02
    #define PF_NEW_NAMES             0x01
```

■ ParserParameters

```
typedef struct {
    CommonParametersPP_common;
    word            PP_parserBufferSize;
    ParserFlags     PP_flags;
    dword           PP_textPtr;
    ScannerToken    PP_currentToken;
    ScannerToken    PP_lookAheadToken;
    byte            PP_error;           /* ParserScannerEvaluatorError */
    word            PP_tokenStart;
    word            PP_tokenEnd;
} ParserParameters;
```

■ ParserScannerEvaluatorError

```
typedef ByteEnum ParserScannerEvaluatorError;
    /*
     * Scanner errors
     */
    #define PSEE_BAD_NUMBER                   0
    #define PSEE_BAD_CELL_REFERENCE           1
    #define PSEE_NO_CLOSE_QUOTE               2
    #define PSEE_COLUMN_TOO_LARGE             3
    #define PSEE_ROW_TOO_LARGE                4
    #define PSEE_ILLEGAL_TOKEN                5
    /*
```

```
 * Parser errors
 */
#define PSEE_GENERAL                        6
#define PSEE_TOO_MANY_TOKENS                7
#define PSEE_EXPECTED_OPEN_PAREN            8
#define PSEE_EXPECTED_CLOSE_PAREN           9
#define PSEE_BAD_EXPRESSION                 10
#define PSEE_EXPECTED_END_OF_EXPRESSION     11
#define PSEE_MISSING_CLOSE_PAREN            12
#define PSEE_UNKNOWN_IDENTIFIER             13
#define PSEE_NOT_ENOUGH_NAME_SPACE          14
/*
 * Serious evaluator errors
 */
#define PSEE_OUT_OF_STACK_SPACE             15
#define PSEE_NESTING_TOO_DEEP               16
/*
 * Evaluator errors that are returned as the result of formulas.
 * These are returned on the argument stack.
 */
#define PSEE_ROW_OUT_OF_RANGE               17
#define PSEE_COLUMN_OUT_OF_RANGE            18
#define PSEE_FUNCTION_NO_LONGER_EXISTS      19
#define PSEE_BAD_ARG_COUNT                  20
#define PSEE_WRONG_TYPE                     21
#define PSEE_DIVIDE_BY_ZERO                 22
#define PSEE_UNDEFINED_NAME                 23
#define PSEE_CIRCULAR_REF                   24
#define PSEE_CIRCULAR_DEP                   25
#define PSEE_CIRC_NAME_REF                  26
#define PSEE_NUMBER_OUT_OF_RANGE            27
#define PSEE_GEN_ERR                        28
#define PSEE_NA                             29
/*
 * Dependency errors
 */
#define PSEE_TOO_MANY_DEPENDENCIES          30
#define PSEE_SSHEET_BASE                 0xc0
#define PSEE_FLOAT_BASE                  250
#define PSEE_APP_BASE                    230
#define PSEE_FLOAT_POS_INFINITY          PSEE_FLOAT_BASE
#define PSEE_FLOAT_NEG_INFINITY          (PSEE_FLOAT_BASE + 1)
#define PSEE_FLOAT_GEN_ERR               (PSEE_FLOAT_BASE + 2)
```

■ ParserToken

```
typedef struct {
    ParserTokenType PT_type;
```

Routines

```
    ParserTokenData PT_data;
} ParserToken;
```

■ ParserTokenCellData

```
typedef struct {
    CellReference   PTCD_cellRef;
} ParserTokenCellData;
```

■ ParserTokenData

```
typedef union {
    ParserTokenNumberData     PTD_number;
    ParserTokenStringData     PTD_string;
    ParserTokenNameData       PTD_name;
    ParserTokenCellData       PTD_cell;
    ParserTokenFunctionData   PTD_function;
    ParserTokenOperatorData   PTD_operator;
} ParserTokenData;
```

■ ParserTokenFunctionData

```
typedef struct {
    word        PTFD_functionID;
} ParserTokenFunctionData;
```

■ ParserTokenNameData

```
typedef struct {
    word        PTND_name;
} ParserTokenNameData;
```

■ ParserTokenNumberData

```
typedef struct {
    FloatNum    PTND_value;
} ParserTokenNumberData;
```

■ ParserTokenOperatorData

```
typedef struct {
    OperatorType    PTOD_operatorID;
} ParserTokenOperatorData;
```

■ ParserTokenStringData

```
typedef struct {
    word        PTSD_length;
} ParserTokenStringData;
```

■ ParserTokenType

```
typedef ByteEnum ParserTokenType;
    #define PARSER_TOKEN_NUMBER             0
    #define PARSER_TOKEN_STRING             1
    #define PARSER_TOKEN_CELL               2
    #define PARSER_TOKEN_END_OF_EXPRESSION  3
    #define PARSER_TOKEN_OPEN_PAREN         4
    #define PARSER_TOKEN_CLOSE_PAREN        5
    #define PARSER_TOKEN_NAME               6
    #define PARSER_TOKEN_FUNCTION           7
    #define PARSER_TOKEN_CLOSE_FUNCTION     8
    #define PARSER_TOKEN_ARG_END            9
    #define PARSER_TOKEN_OPERATOR           10
```

■ PathCombineType

```
typedef enum /* word */ {
    PCT_NULL,                  /* wipe out old path */
    PCT_REPLACE,               /* replace old path with upcoming path */
    PCT_UNION,                 /* union old path with new */
    PCT_INTERSECTION           /* intersect old path with new */
} PathCombineType;
```

■ PathName

```
typedef char PathName[PATH_BUFFER_SIZE];
```

■ PatternType

```
typedef ByteEnum PatternType;
    #define PT_SOLID           0
    #define PT_SYSTEM_HATCH    1
    #define PT_SYSTEM_BITMAP   2
    #define PT_USER_HATCH      3
    #define PT_USER_BITMAP     4
    #define PT_CUSTOM_HATCH    5
    #define PT_CUSTOM_BITMAP   6
```

■ PCDocSizeParams

```
typedef struct {
    dword   PCDSP_width;
    dword   PCDSP_height;
} PCDocSizeParams;
```

Use this structure to communicate document sizes to a Print Control.

■ PCMarginParams

```
typedef struct {
    word    PCMP_left;        /* left margin */
    word    PCMP_top;         /* top margin */
    word    PCMP_right;       /* right margin */
    word    PCMP_bottom;      /* bottom margin */
} PCMarginParams
```

This structure holds information about a document's or printer's margins.

■ PCProgressType

```
typedef enum {
    PCPT_PAGE,
    PCPT_PERCENT,
    PCPT_TEXT
} PCProgressType;
```

■ Point

```
typedef struct {
    sword P_x;
    sword P_y;
} Point;
```

■ PointDWord

```
typedef struct {
    sdword PD_x;
    sdword PD_y;
} PointDWord;
```

■ PointDWFixed

```
typedef struct {
    DWFixed PDF_x;
    DWFixed PDF_y;
} PointDWFixed;
```

■ PointerDef

```
typedef struct {
    sbyte   PD_hotX;
    sbyte   PD_hotY;
    byte    PD_mask[STANDARD_CURSOR_IMAGE_SIZE];
    byte    PD_image[STANDARD_CURSOR_IMAGE_SIZE];
} PointerDef;
     STANDARD_CURSOR_IMAGE_SIZE = 32
```

This structure defines a mouse pointer.

■ PointWWFixed

```
typedef struct {
    WWFixed PF_x;
    WWFixed PF_y;
} PointWWFixed;
```

These structures are used to specify graphics point coordinates. Which point structure to use depends on size of the coordinate space and accuracy required.

■ PrintControlAttrs

```
typedef WordFlags PrintControlAttrs;
    #define PCA_MARK_APP_BUSY     0x2000    /* mark busy while printing */
    #define PCA_VERIFY_PRINT      0x1000    /* verify before printing */
    #define PCA_SHOW_PROGRESS     0x0800    /* show print progress dialog box */
    #define PCA_PROGRESS_PERCENT  0x0400    /* show progress by percentage */
    #define PCA_PROGRESS_PAGE     0x0200    /* show progress by page */
    #define PCA_FORCE_ROTATION    0x0100    /* Force rotation of output */
    #define PCA_COPY_CONTROLS     0x0080    /* Copy controls are available */
    #define PCA_PAGE_CONTROLS     0x0040    /* Page range controls available */
    #define PCA_QUALITY_CONTROLS  0x0020    /* Quality controls available */
    #define PCA_USES_DIALOG_BOX   0x0010    /* Dialog box should appear */
    #define PCA_GRAPHICS_MODE     0x0008    /* Supports graphics mode output */
    #define PCA_TEXT_MODE         0x0004    /* Supports text mode output */
    #define PCA_DEFAULT_QUALITY   0x0002    /* default print quality */
```

■ PrintControlFeatures

```
typedef ByteFlags PrintControlFeatures;
    #define PRINTCF_PRINT_TRIGGER 0x02      /* wants a print trigger */
    #define PRINTCF_FAX_TRIGGER   0x01      /* wants a fax trigger */
```

■ PrintControlStatus

```
typedef enum {
    PCS_PRINT_BOX_VISIBLE,
    PCS_PRINT_BOX_NOT_VISIBLE
} PrintControlStatus;
```

■ PrintControlToolboxFeatures

```
typedef ByteFlags PrintControlToolboxFeatures;
    #define PRINTCTF_PRINT_TRIGGER   0x02   /* wants a print tool trigger */
    #define PRINTCTF_FAX_TRIGGER     0x01   /* wants a fax tool trigger */
```

■ PrinterDriverType

```
typedef enum PrinterDriverType;
    PDT_PRINTER,
```

Routines

```
    PDT_PLOTTER,
    PDT_FACSIMILE,
    PDT_CAMERA,
    PDT_OTHER,
} PrinterDriverType;
```

This enumerated type indeicates the type of printer driver that we are dealing with.

■ PrinterOutputModes

```
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
    #define PRINT_GRAPHICS =  (POM_GRAPHICS_LOW | POM_GRAPHICS_MEDIUM |
                               POM_GRAPHICS_HIGH )
    #define PRINT_TEXT =      (POM_TEXT_DRAFT | POM_TEXT_NLQ)
```

■ PrintQualityEnum

```
typedef enum {
    PQT_HIGH,
    PQT_MEDIUM,
    PQT_LOW
} PrintQualityEnum;
```

■ ProtocolNumber

```
typedef struct {
    word    PN_major;
    word    PN_minor;
} ProtocolNumber;
```

Defines the protocol level of a file, geode, or document. *PN_major* represents significant compatibility comparisons, and *PN_minor* represents less significant differences. If the major protocol is different between to items, they are incompatible. If the minor protocol is different, they may or may not be incompatible.

■ QueueHandle

```
typedef Handle QueueHandle;
```

■ QuickSortParameters

```
typedef struct _QuickSortParameters {
    word _pascal (*QSP_compareCallback) (void *   el1,
                                          void *   el2,
                                     word     valueForCallback));
    void _pascal (*QSP_lockCallback) (void *el, word valueForCallback));
    void _pascal (*QSP_unlockCallback) (void *el, word valueForCallback));
    word         QSP_insertLimit;
    word         QSP_medianLimit;

    /* These are set internally by the quicksort algorithm and should not
     * be set by the caller: */
    word         QSP_nLesser;
    word         QSP_nGreater;

    } QuickSortParameters;
```

This structure is passed to **ArrayQuickSort**. The fields have the following meanings:

**QSP_compareCallback*
: This routine is called to compare elements. It should be declared _pascal. It should return a positive value if **el1* ought to come before **e2* in the sorted list; a negative value if **el1* ought to come after **e2* in the sorted list; and zero if it doesn't matter which comes first.

**QSP_lockCallback*
: This routine is called before **ArrayQuickSort** examines or changes any element. It should be declared _pascal. You can pass a null function pointer, indicating that no locking callback routine should be called.

**QSP_lockCallback*
: This routine is called after **ArrayQuickSort** examines or changes any element. It should be declared _pascal. You can pass a null function pointer, indicating that no unlocking callback routine should be called.

QSP_insertLimit
: If there are fewer than *QSP_insertLimit* elements in a sublist, **ArrayQuickSort** will use an insertion sort for that sublist, rather than a QuickSort.

QSP_medianLimit
: If there are fewer than *QSP_medianLimit* elements in a sublist, ArrayQuickSort will use the first element as a partition, instead of searching for the median element.

■ RangeEnumCallbackParams

```
typedef struct {
    RangeEnumParams *RECP_params;
```

Routines

```
    word            RECP_row;
    word            RECP_column;
    word            RECP_cellData;
} RangeEnumCallbackParams;
```

This structure is passed to the callback routine for **RangeEnum()**.

■ RangeEnumFlags

```
typedef ByteFlags RangeEnumFlags;
    #define REF_ALL_CELLS               0x80
    #define REF_NO_LOCK                 0x40
    #define REF_COLUMN_FLAGS            0x20
    #define REP_MATCH_COLUMN_FLAGS      0x10
    #define REF_CELL_ALLOCATED          0x08
    #define REF_CELL_FREED              0x04
    #define REF_OTHER_ALLOC_OR_FREE     0x02
    #define REF_COLUMN_FLAGS_MODIFIED   0x01
```

These flags are used by **RangeEnum()**.

■ RangeEnumParams

```
typedef struct {
    PCB(RANGE_ENUM_CALLBACK_RETURN_TYPE, REP_callback,
                                        (RangeEnumCallbackParams));
    Rectangle                 REP_bounds;
    byte                      REP_columnFlags;
    word                      *REP_columnFlagsArray;
    CellFunctionParameters    *REP_cfp;
    byte                      REP_matchFlags;
    word                      *REP_locals;
} RangeEnumParams;
```

This structure is used by two routines, **RangeEnum()** and **CellGetExtent()**. When it is used by **RangeEnum()**, the structure specifies all the details about how **RangeEnum()** will function. **CellGetExtent()** is passed a blank **RangeEnumParams** structure; it fills in the *REP_bounds* field.

The callback routine, if any, should be declared _pascal.

Include: cell.h

■ RangeInsertParams

```
typedef  struct {
    Rectangle    RIP_bounds;
    Point        RIP_delta;
    dword        RIP_cfp;
} RangeInsertParams;
```

RangeInsert() is passed the address of a **RangeInsertParams** structure. This structure specifies three things:

RIP_bounds	Which cells should be shifted.
RIP_delta	How far the cells should be shifted and in which direction.
RIP_cfp	The address of the **CellFunctionParameters** structure. You don't have to initialize this; the routine will do so automatically.

Include: cell.h

See Also: RangeInsert()

■ RangeSortError

```
typedef enum /* word */ {
    RSE_NO_ERROR,
    RSE_UNABLE_TO_ALLOC,
} RangeSortError;
```

■ RangeSortCellExistFlags

```
typedef ByteFlags RangeSortCellExistsFlags;
    #define RSCEF_SECOND_CELL_EXISTS 0x02
    #define RSCEF_FIRST_CELL_EXISTS 0x01
```

■ RangeSortFlags

```
typedef ByteFlags RangeSortFlags;
    #define RSF_SORT_ROWS            0x80
    #define RSF_SORT_ASCENDING       0x40
    #define RSF_IGNORE_CASE          0x20
```

■ RangeSortParams

```
typedef struct {
    Rectangle    RSP_range;
    Point        RSP_active;
    dword        RSP_callback;
    byte         RSP_flags; /* RangeSortFlags */
    dword        RSP_cfp;
    word         RSP_sourceChunk;
    word         RSP_destChunk;
    word         RSP_base;
    dword        RSP_lockedEntry;
    byte         RSP_cachedFlags;
} RangeSortParams;
```

Routines

Rectangle

```
typedef struct {
    sword   R_left;
    sword   R_top;
    sword   R_right;
    sword   R_bottom;
} Rectangle;
```

This structure represents a graphics rectangle.

RectDWord

```
typedef struct {
    sdword  RD_left;
    sdword  RD_top;
    sdword  RD_right;
    sdword  RD_bottom;
} RectDWord;
```

This structure represents a graphics rectangle.

RectRegion

```
typedef struct {
    word    RR_y1M1;
    word    RR_eo1;/* EOREGREC */
    word    RR_y2;
    word    RR_x1;
    word    RR_x2;
    word    RR_eo2; /* EOREGREC */
    word    RR_eo3; /* EOREGREC */
} RectRegion;
```

RefElementHeader

```
typedef struct {
     WordAndAHalf REH_refCount;
} RefElementHeader;
```

Region

```
typedef word Region;
    #define EOREGREC          0x8000
    #define EOREG_HIGH         0x80
```

This structure represents a region of a graphics coordinate space.

Routines

Regions are described in terms of a rectangular array (thus the similarity to bitmaps). Instead of specifying an on/off value for each pixel, however, regions assume that the region will be fairly undetailed and that the data structure can thus be treated in the manner of a sparse array. Only the cells in which the color value of a row changes are recorded. The tricky part here is keeping in mind that when figuring out whether or not a row is the same as a previous row, the system works its way up from the bottom, so that you should compare each row with the row beneath it to determine whether it needs an entry.

The easiest region to describe is the null region, which is a special case described by a single word with the value EOREGREC (a constant whose name stands for *E*nd Of *REG*ion *REC*ord value). Describing a non-null region requires several numbers.

The first four numbers of the region description give the bounds of the region. Next come one or more series of numbers. Each series describes a row, specifying which pixels of that row are part of the region. The only rows which need to be described are those which are different from the row below. The first number of each row description is the row number, its *y* coordinate. The last number of each series is a special token, EOREGREC, which lets the kernel know that the next number of the description will be the start of another row. Between the row number and EOREGREC are the column numbers where the pixels toggle on and off. The first number after the row number corresponds to the first column in which the pixel is on; the next number is the first subsequent column in which the pixel is off; and so on.

■ RegionFillRule

```
typedef ByteEnum RegionFillRule;
    #define ODD_EVEN          0
    #define WINDING           1
```

This enumerated type determines how a path or region should be filled. Winding fill is more versatile, but requires that the path or polygon's edges run in the correct direction.

■ ReleaseNumber

```
typedef struct {
    word    RN_major;
    word    RN_minor;
    word    RN_change;
    word    RN_engineering;
} ReleaseNumber;
```

Routines

Used to record what version a file, document, or geode is. This represents the release level; the most significant numbers are *RN_major* and *RN_minor*. The other fields are typically used only internally to a manufacturer.

■ ResolveStandardPathFlags

```
typedef WordFlags FileResolveStandardPathFlags;
    #define FRSPF_ADD_DRIVE_NAME     0x0002
    #define FRSPF_RETURN_FIRST_DIR   0x0001
```

■ RGBColorAsDWord

```
typedef dword RGBColorAsDWord;
    RGB_RED(val) ( val & 0xff)
    RGB_GREEN(val) ( (val >> 8) & 0xff )
    RGB_BLUE(val) ( (val >> 16) & 0xff )
    RGB_INDEX(val) ( (val >> 24) & 0xff )
```

See the **ColorQuad** data structure to find out the meanings of the fields.

■ RGBDelta

```
typedef struct {
     byte RGBD_red;
     byte RGBD_green;
     byte RGBD_blue;
} RGBDelta;
```

■ RGBTransfer

```
typedef struct {
    byte    RGBT_red[256];
    byte    RGBT_green[256];
    byte    RGBT_blue[256];
} RGBTransfer;
```

■ RGBValue

```
typedef struct {
    byte    RGB_red;
    byte    RGB_green;
    byte    RGB_blue;
} RGBValue;
```

■ SampleFormat

```
typedef struct {
          DACSampleFormat SMID_format:15;
          DACReferenceByte SMID_reference:1;
} SampleFormat;
```

■ SampleFormatDescription

```
typedef struct {
    word   SFD_manufact;
    word   SFD_format;
    word   SFD_rate;
    word   SFD_playFlags;
} SampleFormatDescription;
```

This structure acts as a header for a sampled sound, giving format information needed to properly interpret the sound data.

■ SansFace

```
typedef byte SansFace;
    #define SF_A_CLOSED 0x0080
    #define SF_A_OPEN 0x0000
```

■ sbyte

```
typedef char sbyte;
```

■ ScannerToken

```
typedef struct {
    ScannerTokenType          ST_type;
    ScannerTokenData          ST_data;
} ScannerToken;
```

■ ScannerTokenCellData

```
typedef struct {
    CellReference   STCD_cellRef;
} ScannerTokenCellData;
```

■ ScannerTokenData

```
typedef union {
    ScannerTokenNumberData       STD_number;
    ScannerTokenStringData       STD_string;
    ScannerTokenCellData         STD_cell;
    ScannerTokenIdentifierData   STD_identifier;
```

Routines

```
    ScannerTokenOperatorData      STD_operator;
} ScannerTokenData;
```

■ ScannerTokenIdentifierData

```
typedef struct {
    word         STID_start;
} ScannerTokenIdentifierData;
```

■ ScannerTokenNumberData

```
typedef struct {
    FloatNum     STND_value;
} ScannerTokenNumberData;
```

■ ScannerTokenOperatorData

```
typedef struct {
    OperatorType    STOD_operatorID;
} ScannerTokenOperatorData;
```

■ ScannerTokenStringData

```
typedef struct {
    word    STSD_start;
    word    STSD_length;
} ScannerTokenStringData;
```

■ ScannerTokenType

```
typedef ByteEnum ScannerTokenType;
    #define SCANNER_TOKEN_NUMBER              0
    #define SCANNER_TOKEN_STRING              1
    #define SCANNER_TOKEN_CELL                2
    #define SCANNER_TOKEN_END_OF_EXPRESSION   3
    #define SCANNER_TOKEN_OPEN_PAREN          4
    #define SCANNER_TOKEN_CLOSE_PAREN         5
    #define SCANNER_TOKEN_IDENTIFIER          6
    #define SCANNER_TOKEN_OPERATOR            7
    #define SCANNER_TOKEN_LIST_SEPARATOR      8
```

■ ScriptAttrAsWord

```
typedef word ScriptAttrAsWord;
    /*      High byte is a vertical offset, as a fraction of the font size.
            Low byte is a fractional scale to use.
            Thus, setting a subscript attr to 0x8020 would result in subscript
            characters being printed half a line down and at 1/4 normal size. */
```

This structure specifies the offset and scale factor with which sub- and superscript characters should draw.

■ ScriptFace

```
typedef byte ScriptFace;
    #define SF_CURSIVE 0x0080
    #define SF_CALLIGRAPHIC 0x0000
```

■ sdword

```
typedef long sdword;
```

■ segment

```
typedef word segment;
```

■ SemaphoreError

```
typedef enum {
    SE_NO_ERROR,                  /* No error occurred */
    SE_TIMEOUT,                   /* The semaphore timed out before
                                   * it could be grabbed by the thread */
    SE_PREVIOUS_OWNER_DIED        /* The current holder of the semaphore
                                   * exited abnormally */
} SemaphoreError;
```

Determines the error encountered by semaphore and threadlock routines such as **ThreadPSem()** and **ThreadPTimedSem()**.

■ SerialBaud

```
typedef  enum
    {
            SERIAL_BAUD_115200= 1,
            SERIAL_BAUD_57600 = 2,
            SERIAL_BAUD_38400 = 3,
            SERIAL_BAUD_19200 = 6,
            SERIAL_BAUD_14400 = 8,
            SERIAL_BAUD_9600  = 12,
            SERIAL_BAUD_7200  = 16,
            SERIAL_BAUD_4800  = 24,
            SERIAL_BAUD_3600  = 32,
            SERIAL_BAUD_2400  = 48,
            SERIAL_BAUD_2000  = 58,
            SERIAL_BAUD_1800  = 64,
            SERIAL_BAUD_1200  = 96,
            SERIAL_BAUD_600   = 192,
            SERIAL_BAUD_300   = 384
    } SerialBaud;
```

Routines

■ SerialFormat

```
typedef  ByteFlags SerialFormat;
    #define SERIAL_FORMAT_DLAB_OFFSET   (7)
    #define SERIAL_FORMAT_DLAB          (0x01 << SERIAL_FORMAT_DLAB_OFFSET)

    #define SERIAL_FORMAT_BREAK_OFFSET  (6)
    #define SERIAL_FORMAT_BREAK         (0x01 << SERIAL_FORMAT_BREAK_OFFSET)

    #define SERIAL_FORMAT_PARITY_OFFSET (3)
    #define SERIAL_FORMAT_PARITY        (0x07 << SERIAL_FORMAT_PARITY_OFFSET)

    #define SERIAL_FORMAT_EXTRA_STOP_OFFSET (2)
    #define SERIAL_FORMAT_EXTRA_STOP    (0x01 <<\
                                           SERIAL_FORMAT_EXTRA_STOP_OFFSET)

    #define SERIAL_FORMAT_LENGTH_OFFSET (0)
    #define SERIAL_FORMAT_LENGTH        (0x03 << SERIAL_FORMAT_LENGTH_OFFSET)
```

■ SerialMode

```
typedef  enum {
            SERIAL_MODE_RAW,
            SERIAL_MODE_RARE,
            SERIAL_MODE_COOKED
    } SerialMode;
```

■ SerialModem

```
typedef  ByteFlags SerialModem;
    #define SERIAL_MODEM_OUT2_OFFSET (3)
    #define SERIAL_MODEM_OUT2        (0x01 << SERIAL_MODEM_OUT2_OFFSET)

    #define SERIAL_MODEM_OUT1_OFFSET (2)
    #define SERIAL_MODEM_OUT1        (0x01 << SERIAL_MODEM_OUT1_OFFSET)

    #define SERIAL_MODEM_RTS_OFFSET  (1)
    #define SERIAL_MODEM_RTS         (0x01 << SERIAL_MODEM_RTS_OFFSET)

    #define SERIAL_MODEM_DTR_OFFSET  (0)
    #define SERIAL_MODEM_DTR         (0x01 << SERIAL_MODEM_DTR_OFFSET)
```

■ SerialPortNum

```
typedef  enum
    {
            SERIAL_COM1       = 0,
            SERIAL_COM2       = 2,
            SERIAL_COM3       = 4,
            SERIAL_COM4       = 6,
            SERIAL_COM5       = 8,
            SERIAL_COM6        = 10,
            SERIAL_COM7        = 12,
            SERIAL_COM8        = 14
    } SerialPortNum;
```

■ SerialUnit

```
typedef  enum
    {
            SERIAL_COM1        = 0,
            SERIAL_COM2        = 2,
            SERIAL_COM3        = 4,
            SERIAL_COM4        = 6,
            SERIAL_COM5        = 8,
            SERIAL_COM6        = 10,
            SERIAL_COM7        = 12,
            SERIAL_COM8        = 14
    } SerialUnit;
```

■ SemaphoreHandle

```
typedef Handle SemaphoreHandle;
```

■ SerifFace

```
typedef byte SerifFace;
    #define SF_SLAB 0x00c0
    #define SF_MODERN 0x0080
    #define SF_TRANS 0x0040
    #define SF_OLD 0x0000
```

■ SetPalType

```
typedef ByteEnum SetPalType;
    #define SPT_DEFAULT       0
    #define SPT_CUSTOM        1
```

■ ShiftState

```
typedef ByteFlags ShiftState;
    #define SS_LALT           0x80
    #define SS_RALT           0x40
    #define SS_LCTRL          0x20
    #define SS_RCTRL          0x10
    #define SS_LSHIFT         0x08
    #define SS_RSHIFT         0x04
    #define SS_FIRE_BUTTON_1  0x02
    #define SS_FIRE_BUTTON_2  0x01
```

Modifiers which will be incorporated into input information. Corresponds to alt keys, control keys, shift keys, or special system modifiers. Note that these bits will only be set if not already accounted for; that is, if you are passed the character "E", the shift modifiers of this structure will not be marked.

■ SoundDriverCapability

```
typedef WordFlags SoundDriverCapability;
#define SDC_NOISE              0x8000
#define SDC_WAVEFORM           0x6000
#define SDC_TIMBRE             0x1800
#define SDC_ENVELOPE           0x0600

    typedef WordFlags SoundDriverNoiseCapability;
    #define SDNC_NO_NOISE     0x0000
    #define SDNC_WHITE_NOISE  0x8000

    typedef WordFlags SoundDriverWaveFormCapability
    #define SDWFC_NONE        0x0000
    #define SDWFC_SELECT      0x2000
    #define SDWFC_GENERATE    0x4000

    typedef WordFlags SoundDriverTimbreCapability;
    #define SDTC_TONE_GENERATOR 0x0000
    #define SDTC_ADDITIVE     0x0800
    #define SDTC_MODULATOR    0x1000
    #define SDTC_SELECTIVE    0x1800

    typedef WordFlags SoundDriverEnvelopeCapability;
    #define SDEC_NONE         0x0000
    #define SDEC_ADSR         0x0200
    #define SDEC_DSP          0x0400
```

These fields encode information about what the sound driver is capable of in terms of music synthesis.

■ SoundPlayFlags

```
typedef WordFlags SoundPlayFlags;
    #define SPF_HIGH_PRIORITY 0x8000
```

■ SoundPriority

```
typedef enum {
     SP_SYSTEM_LEVEL=10,      /* most urgent */
     SP_ALARM=20,
     SP_STANDARD=30,
     SP_GAME=40,
     SP_BACKGROUND=50         /* least urgent */
} SoundPriority;
```

Routines

```
#define SP_IMMEDIATE      -1
#define SP_THEME          +1
```

If the user's sound device can't play all requested sounds, it will use **SoundPriority** values to determine which sounds are the most important.

The highest priority sound you may construct using these values is (SP_SYSTME_LEVEL + SP_IMMEDIATE). The least priority sound would be (SP_BACKGROUND + SP_THEME).

SoundSteamDeltaTimeType

```
typedef enum {
     SSDTT_MSEC=8,            /* wait for N mili seconds */
     SSDTT_TICKS=10,          /* wait for N ticks */
     SSDTT_TEMPO=12,          /* wait for N beats */
} SoundStreamDeltaTimeType;
    /* The following macros may help when constructing music buffers */
    #define DeltaTick(time)   SSDTT_TICKS, time
    #define DeltaMS(time)     SSDTT_MSEC, time
    #define DeltaTempo(time)  SSDTT_TEMPO, time
```

These are the units by which you can specify a sound's duration: milliseconds, timer "ticks" (each 1/60 second), or by means of an independently supplied tempo.

SoundStreamEvents

```
typedef enum {
     SSE_VOICE_ON=0,          /* turn on voice event */
     SSE_VOICE_OFF=2,         /* turn off voice event */
     SSE_CHANGE=4,            /* change instrument */
     SSE_GENERAL=6            /* system-specific event */
} SoundStreamEvents;
    /* The following macros may help when constructing music buffers */
    #define General(command)  SSE_GENERAL, command

    #define Rest(duration)  General(GE_NO_EVENT), DeltaTick(duration)

    #define VoiceOn(voice,freq,attack)  SSE_VOICE_ON, voice, freq, attack
    #define VoiceOff(voice) SSE_VOICE_OFF, voice
    #define ChangeEnvelope(voice, instrument, table)  \
                             SSE_CHANGE, voice, instrument, table

    #define SoundNote(voice,freq,duration,attack)  \
            VoiceOn(voice, freq, attack), DeltaTempo(duration), VoiceOff(voice)
```

Routines

```
#define Staccato(voice,freq,duration,attack) \
        VoiceOn(voice, freq, attack), DeltaTempo(((duration*0x03)/0x04)), \
        VoiceOff(voice), DeltaTempo((duration/0x4))
#define Natural(voice,freq,duration,attack) \
        VoiceOn(voice, freq, attack), DeltaTempo(((duration*0x07)/0x08)),
        VoiceOff(voice), DeltaTempo((duration/0x8))
#define Legato(voice,freq,duration,attack)  \
        SoundNote(voice, freq, duration, attack)
```

These are the “events” that make up a music buffer.

■ SoundStreamSize

```
typedef word SoundStreamSize;
#define SSS_ONE_SHOT 128 /* 128 bytes (very small) */
#define SSS_SMALL 256 /* 256 bytes */
#define SSS_MEDIUM 512 /* 512 bytes (nice size) */
#define SSS_LARGE 1024
```

■ SoundStreamType

```
#define SST_ONE_SHOT      128
#define SST_SMALL         256
#define SST_MEDIUM        512
#define SST_LARGE         1024
```

■ SpecialFunctions

```
typedef enum /* word */ {
    SF_FILENAME,
    SF_PAGE,
    SF_PAGES,
} SpecialFunctions;
```

■ SpoolError

```
typedef enum /* word */ {
    SERROR_NO_SPOOL_FILE,
    SERROR_NO_PRINT_DRIVER,
    SERROR_NO_PORT_DRIVER,
    SERROR_NO_PRINTERS,
    SERROR_NO_MODE_AVAIL,
    SERROR_CANT_ALLOC_BITMAP,
    SERROR_NO_VIDMEM_DRIVER,
    SERROR_MANUAL_PAPER_FEED,
    SERROR_CANT_LOAD_PORT_DRIVER,
    SERROR_PORT_BUSY,
    SERROR_TEST_NO_PAPER,
    SERROR_TEST_OFFLINE,
    SERROR_TEST_PARALLEL_ERROR,
    SERROR_MISSING_COM_PORT,
    SERROR_PRINT_ON_STARTUP
} SpoolError;
```

■ SpoolFileName

```
typedef struct {
    char    SFN_base[5];
    char    SFN_num[3];
    char    SFN_ext[5];
} SpoolFileName;
```

■ SpoolInfoType

```
typedef enum /* word */ {
    SIT_JOB_INFO,
    SIT_QUEUE_INFO
} SpoolInfoType;
```

■ SpoolOpStatus

```
typedef enum /* word */ {
    SPOOL_OPERATION_SUCCESSFUL,
    SPOOL_JOB_NOT_FOUND,
    SPOOL_QUEUE_EMPTY,
    SPOOL_QUEUE_NOT_EMPTY,
    SPOOL_QUEUE_NOT_FOUND,
    SPOOL_CANT_VERIFY_PORT,
    SPOOL_OPERATION_FAILED
} SpoolOpStatus;
```

■ SpoolTimeStruct

```
typedef struct {
    byte    STS_second;       /* second of the minute (0-59) */
    byte    STS_minute;       /* minute of the hour (0-59) */
    byte    STS_hour;         /* hour of the day (0-23) */
} SpoolTimeStruct;
```

■ SpoolVerifyAction

```
typedef enum {
    SVA_NO_MESSAGE,
    SVA_WARNING,
    SVA_PRINTING
} SpoolVerifyAction;
```

■ StandardDialogBoxType

```
typedef enum {
    SDBT_FILE_NEW_CANNOT_CREATE_TEMP_NAME,
    SDBT_FILE_NEW_INSUFFICIENT_DISK_SPACE,
    SDBT_FILE_NEW_ERROR,
    SDBT_FILE_NEW_WRITE_PROTECTED,
    SDBT_FILE_OPEN_SHARING_DENIED,
    SDBT_FILE_OPEN_FILE_NOT_FOUND,
    SDBT_FILE_OPEN_INVALID_VM_FILE,
    SDBT_FILE_OPEN_INSUFFICIENT_DISK_SPACE,
    SDBT_FILE_OPEN_ERROR,
    SDBT_FILE_OPEN_READ_ONLY,
    SDBT_FILE_OPEN_VM_DIRTY,
    SDBT_FILE_OPEN_APP_MORE_RECENT_THAN_DOC,
    SDBT_FILE_OPEN_DOC_MORE_RECENT_THAN_APP,
    SDBT_FILE_SAVE_INSUFFICIENT_DISK_SPACE,
    SDBT_FILE_SAVE_ERROR,
    SDBT_FILE_SAVE_WRITE_PROTECTED,
    SDBT_FILE_SAVE_AS_FILE_EXISTS,
    SDBT_FILE_SAVE_AS_SHARING_DENIED,
    SDBT_FILE_CLOSE_SAVE_CHANGES,
    SDBT_FILE_CLOSE_ATTACH_DIRTY,
    SDBT_FILE_REVERT_CONFIRM,
    SDBT_FILE_REVERT_ERROR,
    SDBT_FILE_ATTACH_DISK_NOT_FOUND,
    SDBT_CANNOT_OPEN_VOLUME_SELECTED,
    SDBT_QUERY_SAVE_AS_TEMPLATE,
    SDBT_QUERY_SAVE_AS_EMPTY,
    SDBT_QUERY_SAVE_AS_DEFAULT,
    SDBT_QUERY_SAVE_AS_MULTI_USER,
    SDBT_QUERY_SAVE_AS_PUBLIC,
    SDBT_QUERY_RESET_EMPTY_FILE,
    SDBT_QUERY_RESET_DEFAULT_FILE,
    SDBT_CANNOT_OPEN_EMPTY_FILE
} StandardDialogBoxType;
```

■ StandardDialogParams

```
typedef struct {
    word                                 SDP_customFlags;
    char                                *SDP_customString;
    char                                *SDP_stringArg1;
    char                                *SDP_stringArg2;
    StandardDialogResponseTriggerTable  *SDP_customTriggers;
} StandardDialogParams;
```

■ StandardDialogOptrParams

```
typedef struct {
    word    SDOP_customFlags;
    optr    SDOP_customString;
    optr    SDOP_stringArg1;
    optr    SDOP_stringArg2;
    optr    SDOP_customTriggers;
} StandardDialogOptrParams;
```

StandardDialogResponseTriggerEntry

```
typedef struct {
    optr    SDRTE_moniker;
    word    SDRTE_responseValue;
} StandardDialogResponseTriggerEntry;
```

StandardDialog1ResponseTriggerTable

```
typedef struct {
     word                               SD1RTT_numTriggers;
     StandardDialogResponseTriggerEntry SD1RTT_trigger1;
} StandardDialog1ResponseTriggerTable;
```

StandardDialog2ResponseTriggerTable

```
typedef struct {
     word                               SD2RTT_numTriggers;
     StandardDialogResponseTriggerEntry SD2RTT_trigger1;
     StandardDialogResponseTriggerEntry SD2RTT_trigger2;
} StandardDialog2ResponseTriggerTable;
```

StandardDialog3ResponseTriggerTable

```
typedef struct {
     word                               SD3RTT_numTriggers;
     StandardDialogResponseTriggerEntry SD3RTT_trigger1;
     StandardDialogResponseTriggerEntry SD3RTT_trigger2;
     StandardDialogResponseTriggerEntry SD3RTT_trigger3;
} StandardDialog3ResponseTriggerTable;
```

StandardDialog4ResponseTriggerTable

```
typedef struct {
     word                                   SD4RTT_numTriggers;
     StandardDialogResponseTriggerEntry     SD4RTT_trigger1;
     StandardDialogResponseTriggerEntry     SD4RTT_trigger2;
     StandardDialogResponseTriggerEntry     SD4RTT_trigger3;
     StandardDialogResponseTriggerEntry     SD4RTT_trigger4;
} StandardDialog4ResponseTriggerTable;
```

StandardPath

```
typedef enum /* word */ {
    SP_NOT_STANDARD_PATH=0,
    SP_TOP=1,
```

Routines

```
    SP_APPLICATION=3,
    SP_DOCUMENT=5,
    SP_SYSTEM=7,
    SP_PRIVATE_DATA=9,
    SP_STATE=11,
    SP_FONT=13,
    SP_SPOOL=15,
    SP_SYS_APPLICATION=17,
    SP_PUBLIC_DATA=19,
    SP_MOUSE_DRIVERS=21,
    SP_PRINTER_DRIVERS=23,
    SP_FILE_SYSTEM_DRIVERS=25,
    SP_VIDEO_DRIVERS=27,
    SP_SWAP_DRIVERS=29,
    SP_KEYBOARD_DRIVERS=31,
    SP_FONT_DRIVERS=33,
    SP_IMPORT_EXPORT_DRIVERS=35,
    SP_TASK_SWITCH_DRIVERS=37,
    SP_HELP_FILES=39,
    SP_TEMPLATE=41,
    SP_POWER_DRIVERS=43,
    SP_DOS_ROOM=45,
    SP_HWR=47,
    SP_WASTE_BASKET=49,
    SP_BACKUP=51,
    SP_PAGER_DRIVERS=53
    SP_DUMMY=256
} StandardPath;
```

Most routines which are passed disk handles can also be passed members of the **StandardPath** enumerated type. Standard paths let applications access files in a disk-independent manner. Standard paths are usually arranged in a certain hierarchy; for example, the STATE directory usually belongs to the PRIVDATA directory. However, this is entirely at the user's discretion; applications may not make any assumption about how the standard paths are arranged.

■ StandardSoundType

```
typedef enum /* word */ {
    SST_ERROR,              /* Sound produced when an Error box comes up. */
    SST_WARNING,            /* General warning beep sound */
    SST_NOTIFY,             /* General notify beep */
    SST_NO_INPUT,           /* Sound produced when the user's
                             * keystrokes/mouse presses are not going
                             * anywhere */
    SST_CUSTOM_BUFFER=0xfffe, /* Allows applications to play a custom
                             * note buffer and does all the checking
                             * for sound being off, etc. */
```

```
        SST_CUSTOM_NOTE=0xffff    /* Allows applications to play a custom
                                   * note and does all the checking for sound
                                   * being off, etc. */
    } StandardSoundType;
```

■ StreamBlocker

```
typedef  enum{
            STREAM_BLOCK      = 2,
            STREAM_NO_BLOCK   = 0
    } StreamBlocker;
```

■ StreamError

```
typedef  enum{
            STREAM_WOULD_BLOCK,
            STREAM_CLOSING,
            STREAM_CANNOT_ALLOC,
            STREAM_BUFFER_TOO_LARGE,
            STREAM_CLOSED,
            STREAM_SHORT_READ_WRITE
    } StreamError;
```

■ StreamOpenFlags

```
typedef  enum {
            STREAM_OPEN_NO_BLOCK  = 0x01,
            STREAM_OPEN_TIMEOUT   = 0x02
    } StreamOpenFlags
```

■ StreamToken

```
typedef Handleword StreamToken;
```

■ StreamRoles

```
typedef  enum {
            STREAM_ROLES_WRITER   = 0,
            STREAM_ROLES_READER   = -1,
            STREAM_ROLES_BOTH     = -2
    } StreamRoles;
```

■ StyleChunkDesc

```
typedef struct {
    VMFileHandle    SCD_vmFile;
    word            SCD_vmBlockOrMemHandle;
    ChunkHandle     SCD_chunk;
} StyleChunkDesc;
```

Routines

■ StyleElementFlags

```
typedef WordFlags StyleElementFlags;
    #define SEF_DISPLAY_IN_TOOLBOX   0x8000
```

■ StyleElementHeader

```
typedef struct {
    NameArrayElement          SEH_meta;
    word                      SEH_baseStyle;
    StyleElementFlags         SEH_flags;
    dword                     SEH_privateData;
} StyleElementHeader;
```

■ StyleSheetElementHeader

```
typedef struct {
     RefElementHeader         SSEH_meta;
     word                     SSEH_style;
} StyleSheetElementHeader;
```

■ SupportedEnvelopeFormat

```
typedef enum {
     SEF_NO_FORMAT,
     SEF_SBI_FORMAT,
     SEF_CTI_FORMAT
} SupportedEnvelopeFormat;
```

These values specify how a sound device can simulate musical instruments, if it can at all.

■ sword

```
typedef signed short sword;
```

■ SysConfigFlags

```
typedef ByteFlags SysConfigFlags;
    #define SCF_UNDER_SWAT    0x80
    #define SCF_2ND_IC        0x40
    #define SCF_RTC           0x20
    #define SCF_COPROC        0x10
    #define SCF_RESTARTED     0x08
    #define SCF_CRASHED       0x04
    #define SCF_MCA           0x02
    #define SCF_LOGGING       0x01
```

The above flags indicate the system configuration. Any or all of these flags may be set at a time; if a flag is set, the description is true. These flags are used by the kernel and can be retrieved with **SysGetConfig()**.

■ SysDrawMask

```
typedef ByteFlags SysDrawMask;
    #define SDM_INVERSE       0x80
    #define SDM_MASK          0x7f
```

■ SysGetInfoType

See: **SysGetInfo()**.

■ SysMachineType

```
typedef ByteEnum SysMachineType;
    #define SMT_UNKNOWN       0
    #define SMT_PC            1
    #define SMT_PC_CONV       2
    #define SMT_PC_JR         3
    #define SMT_PC_XT         4
    #define SMT_PC_XT_286     5
    #define SMT_PC_AT         6
    #define SMT_PS2_30        7
    #define SMT_PS2_50        8
    #define SMT_PS2_60        9
    #define SMT_PS2_80        10
    #define SMT_PS1           11
```

A byte-sized value indicating the type of machine running GEOS. This value can be retrieved with **SysGetConfig()**.

■ SysNotifyFlags

See: **SysNotify()**.

■ SysProcessorType

```
typedef ByteEnum SysProcessorType;
    #define SPT_8088          0
    #define SPT_8086          0
    #define SPT_80186         1
    #define SPT_80286         2
    #define SPT_80386         3
    #define SPT_80486         4
```

This enumerated type is a byte that indicates the type of processor on the system running GEOS. It can be retrieved with **SysGetConfig()**.

Routines

■ SysShutdownType

See: **SysShutdown()**.

■ SysStats

```
typedef struct {
    dword         SS_idleCount;        /* Idle ticks in the last second. */
    SysSwapInfo   SS_swapOuts;         /* Outward-bound swap activity. */
    SysSwapInfo   SS_swapIns;          /* Inward-bound swap actividy. */
    word          SS_contextSwitches;  /* Context switches in last second. */
    word          SS_interrupts;       /* Interrupts in the last second. */
    word          SS_runQueue;         /* Runnable threads at end of
                                        * last second. */
} SysStats;
```

This structure is returned by **SysStatistics()** and represents the current performance statistics of GEOS.

■ SysSwapInfo

```
typedef struct {
    word    SSI_paragraphs;    /* Number of paragraphs swapped. */
    word    SSI_blocks;        /* Number of blocks swapped. */
} SysSwapInfo;
```

Structure used to represent current swap activity in **SysStats** structure.

■ SystemDrawMask

```
typedef ByteEnum SystemDrawMask;
    #define SDM_TILE              0
    #define SDM_SHADED_BAR        1
    #define SDM_HORIZONTAL        2
    #define SDM_VERTICAL          3
    #define SDM_DIAG_NE           4
    #define SDM_DIAG_NW           5
    #define SDM_GRID              6
    #define SDM_BIG_GRID          7
    #define SDM_BRICK             8
    #define SDM_SLANT_BRICK       9
    #define SDM_0                 89
    #define SDM_12_5              81
    #define SDM_25                73
    #define SDM_37_5              65
    #define SDM_50                57
    #define SDM_62_5              49
    #define SDM_75                41
    #define SDM_87_5              33
    #define SDM_100               25
    #define SDM_CUSTOM           0x7f
    #define SET_CUSTOM_PATTERN   SDM_CUSTOM
```

SystemHatch

```
typedef ByteEnum SystemHatch;
    #define SH_VERTICAL       0
    #define SH_HORIZONTAL     1
    #define SH_45_DEGREE      2
    #define SH_135_DEGREE     3
    #define SH_BRICK          4
    #define SH_SLANTED_BRICK  5
```

TargetLevel

```
typedef enum /* word */ {
    TL_TARGET                 = 0,
    TL_CONTENT,
    TL_GENERIC_OBJECTS        = 1000,
    TL_GEN_SYSTEM,
    TL_GEN_FIELD,
    TL_GEN_APPLICATION,
    TL_GEN_PRIMARY,
    TL_GEN_DISPLAY_CTRL,
    TL_GEN_DISPLAY,
    TL_GEN_VIEW,
    TL_LIBRARY_LEVELS         = 2000,
    TL_APPLICATION_OBJECTS    = 3000,
} TargetLevel;
```

TestRectReturnType

```
typedef ByteEnum TestRectReturnType;
    #define TRRT_OUT          0
    #define TRRT_PARTIAL      1
    #define TRRT_IN           2
```

TextAttr

```
typedef struct {
    byte            TA_colorFlag;
    RGBValue        TA_color;
    SysDrawMask     TA_mask;
    GraphicPattern  TA_pattern;
    TextStyle       TA_styleSet;
    TextStyle       TA_styleClear;
    TextMode        TA_modeSet;
    TextMode        TA_modeClear;
    WBFixed         TA_spacePad;
    FontID          TA_font;
    WBFixed         TA_size;
```

Routines

```
    sword           TA_trackKern;
} TextAttr;
```

■ TextMode

```
typedef ByteFlags TextMode;
    #define TM_TRACK_KERN            0x40
    #define TM_PAIR_KERN             0x20
    #define TM_PAD_SPACES            0x10
    #define TM_DRAW_BASE             0x08
    #define TM_DRAW_BOTTOM           0x04
    #define TM_DRAW_ACCENT           0x02
    #define TM_DRAW_OPTIONAL_HYPHENS 0x01
```

■ TextStyle

```
typedef ByteFlags TextStyle;
    #define TS_OUTLINE        0x40
    #define TS_BOLD           0x20
    #define TS_ITALIC         0x10
    #define TS_SUPERSCRIPT    0x08
    #define TS_SUBSCRIPT      0x04
    #define TS_STRIKE_THRU    0x02
    #define TS_UNDERLINE      0x01
```

■ ThreadException

```
typedef enum {
    TE_DIVIDE_BY_ZERO=0,
    TE_OVERFLOW=4,
    TE_BOUND=8,
    TE_FPU_EXCEPTION=12,
    TE_SINGLE_STEP=16,
    TE_BREAKPOINT=20
} ThreadException;
```

Processor exceptions used primarily for debugging, these are used with **ThreadHandleException()**.

■ ThreadGetInfoType

```
typedef enum {
    TGIT_PRIORITY_AND_USAGE,  /* high byte is thread’s recent CPU usage
                               * low byte is thread’s base priority */
    TGIT_THREAD_HANDLE,       /* handle of the thread */
    TGIT_QUEUE_HANDLE,        /* handle of thread’s event queue */
} ThreadGetInfoType;
```

Used with the routine **ThreadGetInfo()**, it determines the type of information returned by that routine. Use the macros TGI_PRIORITY and

TGI_RECENT_CPU_USAGE to separate the TGIT_PRIORITY_AND_USAGE value into its components.

ThreadHandle

```
typedef Handle ThreadHandle;
```

ThreadLockHandle

```
typedef Handle ThreadLockHandle;
```

ThreadModifyFlags

```
typedef ByteFlags ThreadModifyFlags;
    #define TMF_BASE_PRIO     0x80
    #define TMF_ZERO_USAGE    0x40
```

Used with **ThreadModify()**, these flags determine what aspect of the thread is modified.

TimerCompressedDate

```
typedef WordFlags TimerCompressedDate;
    #define TCD_YEAR           0xfe00    /* years since 1980; e.g. 1988 is ‘8’ */
    #define TCD_MONTH          0x01e0    /* months (1 - 12) (0 illegal) */
    #define TCD_DAY            0x001f    /* days (1-31) (0 illegal) */
```

TimerDateAndTime

```
typedef struct {
    word            TDAT_year;       /* Year based on 1980. (10 => 1990) */
    word            TDAT_month;      /* Number of month (1 through 12) */
    word            TDAT_day;        /* Number of day in month (1 through 31) */
    DaysOfTheWeek   TDAT_dayOfWeek;  /* DayOfTheWeek enumeration */
    word            TDAT_hours;      /* Hour of the day (0 through 23) */
    word            TDAT_minutes;    /* Minute in the hour (0 through 59) */
    word            TDAT_seconds;    /* Second in the minute (0 through 59) */
} TimerDateAndTime;
```

This structure is used to keep track of the current time and date.

TimerHandle

```
typedef Handle TimerHandle;
```

TimerType

See: **TimerStart()**.

ToggleState

```
typedef ByteFlags ToggleState;
```

Routines

```
#define TS_CAPSLOCK       0x80
#define TS_NUMLOCK        0x40
#define TS_SCROLLLOCK     0x20
```

This structure describes the state of certain “toggles” which will affect how input is interpreted. These toggles correspond to the caps lock, num lock, and scroll lock keys.

■ TokenChars

```
typedef char TokenChars[TOKEN_CHARS_LENGTH]; /* TOKEN_CHARS_LENGTH=4 */
```

■ TokenDBItem

```
typedef DBGroupAndItem TokenDBItem;
```

■ TokenEntry

```
typedef struct {
    GeodeToken      TE_token;        /* A GeodeToken structure for this file */
    TokenDBItem     TE_monikerList;  /* A list of monikers for this token */
    TokenFlags      TE_flags;        /* Flags indicating relocation status */
    ReleaseNumber   TE_release;      /* Release number of the token DB */
    ProtocolNumber  TE_protocol;     /* Protocol number of the toke DB */
} TokenEntry;
```

Used for the token entry in the map item of the token database, this structure identifies the structures and other information of each token. The *TE_monikerList* field points to a chunk containing the item numbers of the chunks of the token.

■ TokenFlags

```
typedef WordFlags TokenFlags;
    #define TF_NEED_RELOCATION 0x8000
```

Used by token management routines, this flags record indicates whether the token has fields which must be relocated when the token is loaded or unloaded.

■ TokenGroupEntry

```
typedef struct {
    TokenIndexType  TGE_type;        /* The type of structure this is. */
    GroupType       TGE_groupType;   /* The type of the group item. */
    word            TGE_groupNum;    /* The number of the group. */
    word            TGE_groupSize;   /* The size of the group. */
} TokenGroupEntry;
```

Used to index token groups in the token database.

■ TokenGroupType

```
typedef enum {
    TGT_MAP_GROUP,              /* The TokenGroupEntry is a map group. */
    TGT_MONIKER_LIST_GROUP,     /* The TokenGroupEntry is a moniker list group. */
    TGT_TEXT_MONIKER_GROUP,     /* The TokenGroupEntry is a text moniker group. */
    TGT_CGA_MONIKER_GROUP,      /* The TokenGroupEntry is a CGA moniker group. */
    TGT_EGA_MONIKER_GROUP,      /* The TokenGroupEntry is an EGA moniker group. */
    TGT_VGA_MONIKER_GROUP,      /* The TokenGroupEntry is a VGA moniker group. */
    TGT_HGC_MONIKER_GROUP,      /* The TokenGroupEntry is an HGC moniker group. */
} TokenGroupType;
```

This enumerated type describes which type of moniker group is stored in the particular chunk.

■ TokenIndexType

```
typedef enum {
    TIT_TOKEN_ENTRY,            /* The type is a TokenEntry structure. */
    TIT_GROUP_ENTRY,            /* The type is a GroupEntry structure. */
} TokenIndexType;
```

Used to indicate the types of structures that may be stored in the token database's map item.

■ TokenMonikerInfo

```
typedef struct {
    TokenDBItem  TMI_moniker;
    word         TMI_fileFlag;    /* 0 if token is in shared token DB file;
                                   * Non-0 if it’s in local file */
    } TokenMonikerInfo;
```

■ TokenRangeFlags

```
typedef WordFlags TokenRangeFlags;
    #define TRF_ONLY_GSTRING0x8000
    #define TRF_ONLY_PASSED_MANUFID0x4000
    #define TRF_UNUSED0x3fff
```

■ TransError

```
typedef enum {
     TE_NO_ERROR, /* No error */
     TE_ERROR, /* General error */
     TE_INVALID_FORMAT, /* Format is invalid */
     TE_IMPORT_NOT_SUPPORTED, /* Format is not supported for export */
     TE_EXPORT_NOT_SUPPORTED, /* Format is not supported for export */
     TE_IMPORT_ERROR, /* General error during import */
     TE_EXPORT_ERROR, /* General error during export */
     TE_FILE_ERROR, /* Generic file error */
```

Routines

```
     TE_DISK_FULL, /* The disk is full */
     TE_FILE_OPEN, /* Error in opening a file */
     TE_FILE_READ, /* Error in reading from a file */
     TE_FILE_WRITE, /* Error in writing to a file */
     TE_FILE_TOO_LARGE, /* File is too large to process */
     TE_OUT_OF_MEMORY, /* Insufficient memory for import/export */
     TE_METAFILE_CREATION_ERROR, /* Error in creating the metafile */
     TE_EXPORT_FILE_EMPTY, /* File to be exported is empty */
     TE_CUSTOM /* Custom error message */
} TransError;
```

This enumerated type contains error values the impex library may wish to generate when translating.

TransErrorInfo

```
typedef struct {
     TransError transError;
     /* NOTE: customMsgHandle will be valid only if transError is TE_CUSTOM. */
    word customMsgHandle;
} TransErrorInfo;
```

TransferBlockID

```
typedef dword TransferBlockID;
    #define BlockIDFromFileAndBlock(f,b)   (((dword)(f) << 16) | (b))
    #define FileFromTransferBlockID(id)    ((VMFileHandle) ((id) >> 16))
    #define BlockFromTransferBlockID(id)   ((VMBlockHandle) (id))
```

TransMatrix

```
typedef struct {
    WWFixed      TM_e11;
    WWFixed      TM_e12;
    WWFixed      TM_e21;
    WWFixed      TM_e22;
    DWFixed      TM_e31;
    DWFixed      TM_e32;
} TransMatrix;
```

The six variable elements of a coordinate transformation matrix.

TravelOption

```
typedef enum {
     TO_NULL,
     TO_SELF,
     TO_OBJ_BLOCK_OUTPUT,
    TO_PROCESS
} TravelOption;
/* VisClass defines one other travel option: */
typedef enum {
    TO_VIS_PARENT=_FIRST_VisClass
} VisTravelOption;
/* GenClass defines some more travel options: */
typedef enum /* word */ {
    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;
```

This enumerated type can be used to specify the recipient of a message. Note that the values set up in the **TravelOption**, **VisTravelOption**, and **GenTravelOption** have been set up as descrete values.

■ TRUE

```
#define TRUE       -1    /* use as return value, not for comparisons */
#define FALSE       0
```

■ UIFunctionsActive

```
typedef ByteFlags UIFunctionsActive;
    #define UIFA_SELECT        0x80
    #define UIFA_MOVE_COPY     0x40
    #define UIFA_FEATURES      0x20
    #define UIFA_CONSTRAIN     0x10
    #define UIFA_PREF_A        0x08
    #define UIFA_PREF_B        0x04
    #define UIFA_PREF_C        0x02
    #define UIFA_IN            0x01
    #define UIFA_ADJUST        0x08
    #define UIFA_EXTEND        0x04
    #define UIFA_MOVE          0x08
    #define UIFA_COPY          0x04
    #define UIFA_POPUP         0x08
    #define UIFA_PAN           0x04
```

Routines

These flags describe the context of the user's input, providing some modal information.

■ UIInterfaceLevel

```
typedef enum /* word */ {
    UIIL_NOVICE,
    UIIL_BEGINNING_INTERMEDIATE,
    UIIL_ADVANCED_INTERMEDIATE,
    UIIL_ADVANCED,
    UIIL_GURU
} UIInterfaceLevel;
```

■ UndoActionDataFlags

```
typedef struct {
    dword        UADF_flags;
    word         UADF_extraflags;
} UndoActionDataFlags;
```

■ UndoActionDataPtr

```
typedef struct {
    void         *UADP_ptr;
    word         UADP_size;
} UndoActionDataPtr;
```

■ UndoActionDataType

```
typedef enum /* word */ {
    UADT_FLAGS,
    UADT_PTR,
    UADT_VM_CHAIN,
} UndoActionDataType;
```

■ UndoActionDataUnion

```
typedef union {
    /* To find out the type of data stored in this
     * union, check the value of the UndoActionStruct's
     * UAS_dataType field. */
    UndoActionDataFlags       UADU_flags;
    UndoActionDataPtr         UADU_ptr;
    UndoActionDataVMChain     UADU_vmChain;
} UndoActionDataUnion;
#define NULL_UNDO_CONTEXT 0
```

■ UndoActionDataVMChain

```
typedef struct {
    /* This structure is filled in by the code for
     * MSG_META_UNDO. VMChains passed to
     * MSG_GEN_PROCESS_UNDO_ADD_ACTION should lie in the undo
     * file (which can be obtained by sending
     * MSG_GEN_PROCESS_UNDO_GET_FILE). */
    VMChain         UADVMC_vmChain;
    VMFileHandle    UADVMC_file;
} UndoActionDataVMChain;
```

■ UndoActionStruct

```
typedef struct {
    UndoActionDataType        UAS_dataType;
    UndoActionDataUnion       UAS_data;
    dword                     UAS_appType;
} UndoActionStruct;
```

■ UtilAsciiToHexError

```
typedef enum /* word */ {
    UATH_NON_NUMERIC_DIGIT_IN_STRING,
    UATH_CONVERT_OVERFLOW,
    } UtilAsciiToHexError;
```

■ UtilHexToAsciiFlags

```
typedef WordFlags UtilHexToAsciiFlags;
    #define UHTAF_INCLUDE_LEADING_ZEROS 0x0002
    #define UHTAF_NULL_TERMINATE        0x0001
```

■ VarDataCHandler

```
typedef  struct {
    word    VDCH_dataType;
    void    (*VDCH_handler) (MemHandle mh, ChunkHandle ch,
                             VarDataEntry *extraData,
                             word dataType, void *handlerData);
} VarDataCHandler;
```

An entry in a class' vardata handler table. The first field is the data type, which acts as the entry's index in the handler table. The second field is a far pointer to the handler routine.

■ VarDataEntry

```
typedef struct {
    word    VDE_dataType;     /* vardata data type */
```

Routines

```
    word    VDE_entrySize;    /* size of extra data; this field only exists
                               * if the type has extra data. */
} VarDataEntry;
#define VDE_extraData          sizeof(VarDataEntry);
```

Structure of a variable data entry. If the data type has no extra data, there will be no *VDE_entrySize* field. The extra data begins at offset *VDE_extraData*, defined above.

■ VarDataFlags

```
typedef WordFlags VarDataFlags;
    #define VDF_TYPE          0xfffc       /* 14-bit data type */
    #define VDF_EXTRA_DATA    0x0002       /* set if has extra data */
    #define VDF_SAVE_TO_STATE 0x0001       /* set if type saved to state */
```

This is a word record containing three fields. This word is stored in the vardata structure's *VDE_dataType* field (see **VarDataEntry**, above).

■ VarDataKey

```
typedef word VardataKey;
```

■ VarObjRelocation

```
typedef struct {
    VarDataFlags    VOR_type;        /* type and tag */
    word            VOR_offset;
} VarObjRelocation;
```

■ VChar

```
typedef ByteEnum VChar;
    #define VC_NULL           0x0 /* NULL */
    #define VC_CTRL_A         0x1 /* <ctrl>-A */
    #define VC_CTRL_B         0x2 /* <ctrl>-B */
    #define VC_CTRL_C         0x3 /* <ctrl>-C */
    #define VC_CTRL_D         0x4 /* <ctrl>-D */
    #define VC_CTRL_E         0x5 /* <ctrl>-E */
    #define VC_CTRL_F         0x6 /* <ctrl>-F */
    #define VC_CTRL_G         0x7 /* <ctrl>-G */
    #define VC_CTRL_H         0x8 /* <ctrl>-H */
    #define VC_CTRL_I         0x9 /* <ctrl>-I */
    #define VC_CTRL_J         0xa /* <ctrl>-J */
    #define VC_CTRL_K         0xb /* <ctrl>-K */
    #define VC_CTRL_L         0xc /* <ctrl>-L */
    #define VC_CTRL_M         0xd /* <ctrl>-M */
    #define VC_CTRL_N         0xe /* <ctrl>-N */
    #define VC_CTRL_O         0xf /* <ctrl>-O */
    #define VC_CTRL_P         0x10 /* <ctrl>-P */
    #define VC_CTRL_Q         0x11 /* <ctrl>-Q */
```

```
#define VC_CTRL_R          0x12 /* <ctrl>-R */
#define VC_CTRL_S          0x13 /* <ctrl>-S */
#define VC_CTRL_T          0x14 /* <ctrl>-T */
#define VC_CTRL_U          0x15 /* <ctrl>-U */
#define VC_CTRL_V          0x16 /* <ctrl>-V */
#define VC_CTRL_W          0x17 /* <ctrl>-W */
#define VC_CTRL_X          0x18 /* <ctrl>-X */
#define VC_CTRL_Y          0x19 /* <ctrl>-Y */
#define VC_CTRL_Z          0x1a /* <ctrl>-Z */
#define VC_ESCAPE          0x1b /* ESC */
#define VC_BLANK           0x20 /* space */
/*
 * Numeric keypad keys
 */
#define VC_NUMPAD_ENTER    0xd /* only on PS/2 keyboards */
#define VC_NUMPAD_DIV      '/' /* only on PS/2 keyboards */
#define VC_NUMPAD_MULT     '*'
#define VC_NUMPAD_PLUS     '+'
#define VC_NUMPAD_MINUS    '-'
#define VC_NUMPAD_PERIOD   '.'
#define VC_NUMPAD_0        '0'
#define VC_NUMPAD_1        '1'
#define VC_NUMPAD_2        '2'
#define VC_NUMPAD_3        '3'
#define VC_NUMPAD_4        '4'
#define VC_NUMPAD_5        '5'
#define VC_NUMPAD_6        '6'
#define VC_NUMPAD_7        '7'
#define VC_NUMPAD_8        '8'
#define VC_NUMPAD_9        '9'
/*
 * Extended keyboard codes -- non-ASCII
 */
#define VC_F1              0x80 /* Function keys */
#define VC_F2              0x81
#define VC_F3              0x82
#define VC_F4              0x83
#define VC_F5              0x84
#define VC_F6              0x85
#define VC_F7              0x86
#define VC_F8              0x87
#define VC_F9              0x88
#define VC_F10             0x89
#define VC_F11             0x8a /* only on PS/2 keyboards */
#define VC_F12             0x8b /* only on PS/2 keyboards */
#define VC_F13             0x8c /* non-standard key */
#define VC_F14             0x8d /* non-standard key */
#define VC_F15             0x8e /* non-standard key */
#define VC_F16             0x8f /* non-standard key */
#define VC_UP              0x90 /* Cursor keys */
```

Routines

```
#define VC_DOWN           0x91
#define VC_RIGHT          0x92
#define VC_LEFT           0x93
#define VC_HOME           0x94 /* Scroll commands */
#define VC_END            0x95
#define VC_PREVIOUS       0x96
#define VC_NEXT           0x97
#define VC_INS            0x98 /* INS */
#define VC_DEL            0x9a /* DEL */
#define VC_PRINTSCREEN    0x9b /* from <shift>-NUMPAD_MULT */
#define VC_PAUSE          0x9c /* from <ctrl>-NUMLOCK */
#define VC_BREAK          0x9e /* from <ctrl>- or <alt>-combo */
#define VC_SYSTEMRESET    0x9f /* <ctrl>-<alt>-<del> combo */
/*
* Joystick control keys (0xa0 - 0xa9)
*/
#define VC_JOYSTICK_0     0xa0        ; joystick 0 degrees
#define VC_JOYSTICK_45    0xa1        ; joystick 45 degrees
#define VC_JOYSTICK_90    0xa2        ; joystick 90 degrees
#define VC_JOYSTICK_135   0xa3        ; joystick 135 degrees
#define VC_JOYSTICK_180   0xa4        ; joystick 180 degrees
#define VC_JOYSTICK_225   0xa5        ; joystick 225 degrees
#define VC_JOYSTICK_270   0xa6        ; joystick 270 degrees
#define VC_JOYSTICK_315   0xa7        ; joystick 315 degrees
#define VC_FIRE_BUTTON_1  0xa8        ; fire button #1
#define VC_FIRE_BUTTON_2  0xa9        ; fire button #2

/*
 * Shift Keys   (0xe0 - 0xe7)
 */
#define VC_LALT           0xe0
#define VC_RALT           0xe1
#define VC_LCTRL          0xe2
#define VC_RCTRL          0xe3
#define VC_LSHIFT         0xe4
#define VC_RSHIFT         0xe5
#define VC_SYSREQ         0xe6 /* Not on base PC keyboard */
#define VC_ALT_GR         0xe7
/*
 * Toggle state keys (0xe8 - 0xef)
 */
#define VC_CAPSLOCK       0xe8
#define VC_NUMLOCK        0xe9
#define VC_SCROLLLOCK     0xea
/*
 * Extended state keys (0xf0 - 0xf7)
 */
#define VC_INVALID_KEY    0xff
#define VC_BACKSPACE      VC_CTRL_H
#define VC_TAB            VC_CTRL_I
```

```
    #define VC_LF            VC_CTRL_J
    #define VC_ENTER         VC_CTRL_M
```

■ VisRulerType

```
typedef ByteEnum VisRulerType;
    #define VRT_INCHES        0
    #define VRT_CENTIMETERS   1
    #define VRT_POINTS        2
    #define VRT_PICAS         3
    #define VRT_CUSTOM        CUSTOM_RULER_DEFINITION
    #define VRT_NONE          NO_RULERS
    #define VRT_DEFAULT       SYSTEM_DEFAULT
```

■ VisTextVariableType

```
typedef enum {
     VTVT_PAGE_NUMBER,
     VTVT_PAGE_NUMBER_IN_SECTION,
     VTVT_NUMBER_OF_PAGES,
     VTVT_NUMBER_OF_PAGES_IN_SECTION,
     VTVT_SECTION_NUMBER,
     VTVT_NUMBER_OF_SECTIONS,
     VTVT_CREATION_DATE_TIME,
     VTVT_MODIFICATION_DATE_TIME,
     VTVT_CURRENT_DATE_TIME,
     VTVT_STORED_DATE_TIME,
    } VisTextVariableType;
```

■ VisTravelOption

The **VisClass** defines an enumerated value to be used in the place of a standard **TravelOption**. See the entry for **TravelOption** to see all possible values.

■ VisUpdateMode

```
typedef ByteEnum VisUpdateMode;
    #define VUM_MANUAL                  0
    #define VUM_NOW                     1
    #define VUM_DELAYED_VIA_UI_QUEUE    2
    #define VUM_DELAYED_VIA_APP_QUEUE   3
```

■ VMAccessFlags

```
typedef ByteFlags VMAccessFlags;
    #define VMAF_FORCE_READ_ONLY                  0x80
    #define VMAF_FORCE_READ_WRITE                 0x40
    #define VMAF_ALLOW_SHARED_MEMORY              0x20
    #define VMAF_FORCE_DENY_WRITE                 0x10
    #define VMAF_DISALLOW_SHARED_MULTIPLE         0x08
```

Routines

```
    #define VMAF_USE_BLOCK_LEVEL_SYNCHRONIZATION  0x04
```

■ VMAttributes

```
typedef ByteFlags VMAttributes;
    #define VMA_SYNC_UPDATE                 0x80
    #define VMA_BACKUP                      0x40
    #define VMA_OBJECT_RELOC                0x20
    #define VMA_PRESERVE_HANDLES            0x10
    #define VMA_NOTIFY_DIRTY                0x08
    #define VMA_NO_DISCARD_IF_IN_USE        0x04
    #define VMA_COMPACT_OBJ_BLOCK           0x02
    #define VMA_SINGLE_THREAD_ACCESS        0x01
    /*
     * Attributes that must be set for object blocks: */
    #define VMA_OBJECT_ATTRS  (VMA_OBJECT_RELOC | VMA_PRESERVE_HANDLES |
                               VMA_NO_DISCARD_IF_IN_USE |
                               VMA_SINGLE_THREAD_ACCESS)
```

■ VMBlockHandle

```
typedef word VMBlockHandle;
```

■ VMChain

```
typedef dword VMChain;
```

■ VMChainLink

```
typedef struct {
    VMBlockHandle   VMC_next;
} VMChainLink;
```

■ VMChainTree

```
typedef struct {
    VMChainLink  VMCT_meta;
    word         VMCT_offset;
    word         VMCT_count;
} VMChainTree;
```

■ VMFileHandle

```
typedef Handle VMFileHandle;
```

■ VMInfoStruct

```
typedef struct {
    MemHandle    mh;
    word         size;
    word        userId;
} VMInfoStruct;
```

■ VMOpenType

```
typedef ByteEnum VMOpenType;
    #define VMO_OPEN                    0
    #define VMO_TEMP_FILE               1
    #define VMO_CREATE                  2
    #define VMO_CREATE_ONLY             3
    #define VMO_CREATE_TRUNCATE         4
    #define VMO_NATIVE_WITH_EXT_ATTRS   0x80
```

■ VMOperation

```
typedef enum {
    VMO_READ,
    VMO_INTERNAL,
    VMO_SAVE,
    VMO_SAVE_AS,
    VMO_REVERT,
    VMO_UPDATE,
    VMO_WRITE
} VMOperation;
```

■ VMRelocType

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

■ VMStartExclusiveReturnValue

```
typedef enum {
     VMSERV_NO_CHANGES,
     VMSERV_CHANGES,
     VMSERV_TIMEOUT
} VMStartExclusiveReturnValue;
```

VMGrabExclusive() returns a member of this enumerated type. It may have one of the following values:

VMSERV_NO_CHANGES
: No other thread has changed this file since the last time this thread had access to the file.

Routines

VMSERV_CHANGES
: The file may have been altered since the last time this thread had access to it; the thread should take appropriate actions (such as re-reading any cached data).

VMSERV_TIMEOUT
: This call to **VMGrabExclusive()** failed and timed out without getting access to the file.

■ VolumeName

```
typedef char VolumeName[VOLUME_BUFFER_SIZE];
```

■ WBFixed

```
typedef struct {
    byte WBF_frac;
    word WBF_int;
} WBFixed;
```

■ wchar

```
typedef unsigned int wchar;
```

■ WindowHandle

```
typedef Handle WindowHandle;
```

■ WinInfoType

```
typedef enum /* word */ {
    WIT_PRIVATE_DATA =0,
    WIT_COLOR =2,
    WIT_INPUT_OBJ =4,
    WIT_EXPOSURE_OBJ =6,
    WIT_STRATEGY =8,
    WIT_FLAGS =10,
    WIT_LAYER_ID =12,
    WIT_PARENT_WIN =14,
    WIT_FIRST_CHILD_WIN =16,
    WIT_LAST_CHILD_WIN =18,
    WIT_PREV_SIBLING_WIN =20,
    WIT_NEXT_SIBLING_WIN =22,
    WIT_PRIORITY=24,
} WinInfoType;
```

■ WinInvalFlag

```
typedef ByteEnum WinInvalFlag;
    #define WIF_INVALIDATE       0
    #define WIF_DONT_INVALIDATE  1
```

■ WinPassFlags

```
typedef WordFlags WinPassFlags;
    #define WPF_CREATE_GSTATE         0x8000
    #define WPF_ROOT                  0x4000
    #define WPF_SAVE_UNDER            0x2000
    #define WPF_INIT_EXCLUDED         0x1000
    #define WPF_PLACE_BEHIND          0x0800
    #define WPF_PLACE_LAYER_BEHIND    0x0400
    #define WPF_LAYER                 0x0200
    #define WPF_ABS                   0x0100
    #define WPF_PRIORITY              0x00ff
```

■ WinPriority

```
typedef ByteEnum WinPriority;
    #define WIN_PRIO_POPUP         4
    #define WIN_PRIO_MODAL         6
    #define WIN_PRIO_ON_TOP        8
    #define WIN_PRIO_COMMAND      10
    #define WIN_PRIO_STD          12
    #define WIN_PRIO_ON_BOTTOM    14
```

■ word

```
typedef unsigned int word;
```

■ WordAndAHalf

```
typedef struct {
    word    WAAH_low;
    byte    WAAH_high;
} WordAndAHalf;
```

■ WordFlags

```
typedef word WordFlags;
```

■ WWFixed

```
typedef struct {
    word WWF_frac;
    word WWF_int;
} WWFixed;
```

■ WWFixedAsDWord

```
typedef dword WWFixedAsDWord
```

Routines

■ XYOffset

```
typedef struct {
    sword XYO_x;
    sword XYO_y;
} XYOffset;
```

A graphics coordinate offset.

■ XYSize

```
typedef struct {
    word XYS_width;
    word XYS_height;
} XYSize;
```

A graphics size, in two dimensions.

■ XYValueAsDWord

```
typedef dword XYValueAsDWord;
```

A graphics size, in two dimensions, expressed as a DWord.

Routines