iffparse

COLLABORATORS			
	TITLE : iffparse		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

iffparse

1.1 iffparse.doc

```
AllocIFF()
AllocLocalItem()
CloseClipboard()
CloseIFF()
CollectionChunk()
CollectionChunks()
CurrentChunk()
EntryHandler()
ExitHandler()
FindCollection()
FindLocalItem()
FindProp()
FindPropContext()
FreeIFF()
FreeLocalItem()
GoodID()
GoodType()
IDtoStr()
InitIFF()
InitIFFasClip()
InitIFFasDOS()
LocalItemData()
OpenClipboard()
OpenIFF()
ParentChunk()
ParseIFF()
PopChunk()
PropChunk()
PropChunks()
PushChunk()
ReadChunkBytes()
ReadChunkRecords()
SetLocalItemPurge()
StopChunk()
StopChunks()
StopOnExit()
StoreItemInContext()
StoreLocalItem()
```

```
    WriteChunkBytes()
    WriteChunkRecords()
```

1.2 iffparse.library/AllocIFF

```
 NAME
AllocIFF -- create a new IFFHandle structure. (V36)

 SYNOPSIS
iff = AllocIFF()
D0

struct IFFHandle *AllocIFF(VOID);

 FUNCTION
Allocates and initializes a new IFFHandle structure.
This function is the only supported way to create an IFFHandle
structure since there are private fields that need to be initialized.

 RESULT
iff - pointer to IFFHandle structure or NULL if the allocation failed.

 SEE ALSO
FreeIFF(), <libraries/iffparse.h>
```

1.3 iffparse.library/AllocLocalItem

```
 NAME
AllocLocalItem -- create a local context item structure. (V36)

 SYNOPSIS
item = AllocLocalItem(type, id, ident, dataSize);
D0                    D0    D1  D2     D3

struct LocalContextItem *AllocLocalItem(LONG, LONG, LONG, LONG);

 FUNCTION
Allocates and initializes a LocalContextItem structure with "dataSize"
bytes of associated user data. This is the only supported way to
create such an item. The user data can be accessed with the
LocalItemData() function. An item created with this function
automatically has its purge vectors set up correctly to dispose of
itself and its associated user data area. Any additional cleanup
should be done with a user-supplied purge vector.

 INPUTS
type,id - additional longword identification values
ident - longword identifier for class of context item
dataSize - number of bytes of user data to allocate for this item

 RESULT
item - pointer to initialized LocalContextItem or NULL if the
```

```
       allocation failed.

 SEE ALSO
FreeLocalItem(), LocalItemData(), StoreLocalItem(),
StoreItemInContext(), SetLocalItemPurge(), <libraries/iffparse.h>
```

1.4 iffparse.library/CloseClipboard

```
 NAME
CloseClipboard -- close and free an open ClipboardHandle. (V36)

 SYNOPSIS
CloseClipboard(clipHandle);
               A0

VOID CloseClipboard(struct ClipboardHandle *);

 FUNCTION
Closes the clipboard.device and frees the ClipboardHandle structure.

 INPUTS
clipHandle - pointer to ClipboardHandle struct created with
             OpenClipboard(). Starting with V39, this may be NULL.

 SEE ALSO
OpenClipboard(), InitIFFasClip(), <libraries/iffparse.h>
```

1.5 iffparse.library/CloseIFF

```
 NAME
CloseIFF -- close an IFF context. (V36)

 SYNOPSIS
CloseIFF(iff);
         A0

VOID CloseIFF(struct IFFHandle *);

 FUNCTION
Completes an IFF read or write operation by closing the IFF context
established for this IFFHandle structure. The IFFHandle structure
itself is left ready for re-use and a new context can be opened with
OpenIFF(). This function can be used for cleanup if a read or write
fails partway through.

As part of its cleanup operation, CloseIFF() calls the client-
supplied stream hook vector. The IFFStreamCmd packet will be set
as follows:

  sc_Command: IFFCMD_CLEANUP
  sc_Buf:   (Not applicable)
  sc_NBytes:  (Not applicable)
```

```
This operation is NOT permitted to fail;  any error code returned
will be ignored (best to return 0, though). DO NOT write to this
structure.

 INPUTS
iff - pointer to IFFHandle structure previously opened with
      OpenIFF(). Starting with V39, this may be NULL.

 SEE ALSO
OpenIFF(), InitIFF(), <libraries/iffparse.h>
```

1.6 iffparse.library/CollectionChunk

```
 NAME
CollectionChunk -- declare a chunk type for collection. (V36)

 SYNOPSIS
error = CollectionChunk(iff, type, id);
D0                      A0   D0    D1

LONG CollectionChunk(struct IFFHandle *, LONG, LONG);

 FUNCTION
Installs an entry handler for chunks with the given type and id so
that the contents of those chunks will be stored as they are
encountered. This is like PropChunk() except that more than one
chunk of this type can be stored in lists which can be returned by
FindCollection(). The storage of these chunks still follows the
property chunk scoping rules for IFF files so that at any given
point, stored collection chunks will be valid in the current context.

 INPUTS
iff - pointer to IFFHandle structure (does not need to be open)
type - type code for the chunk to declare (ex. "ILBM")
id - identifier for the chunk to declare (ex. "CRNG")

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
CollectionChunks(), FindCollection(), PropChunk(),
<libraries/iffparse.h>
```

1.7 iffparse.library/CollectionChunks

```
 NAME
CollectionChunks -- declare many collection chunks at once. (V36)

 SYNOPSIS
error = CollectionChunks(iff, propArray, numPairs);
D0                       A0   A1         D0
```

```
LONG CollectionChunks(struct IFFHandle *, LONG *, LONG);

 FUNCTION
Declares multiple collection chunks from a list. The propArray argument
is a pointer to an array of longwords arranged in pairs. The format
for the list is as follows:

  TYPE1, ID1, TYPE2, ID2, ..., TYPEn, IDn

The argument numPairs is the number of pairs. CollectionChunks() just calls
CollectionChunk() numPairs times.

 INPUTS
iff - pointer to IFFHandle structure (does not need to be open)
propArray - pointer to array of longword chunk types and identifiers
numPairs - number of pairs in array.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful

 SEE ALSO
CollectionChunk(), <libraries/iffparse.h>
```

1.8 iffparse.library/CurrentChunk

```
 NAME
CurrentChunk -- get context node for current chunk. (V36)

 SYNOPSIS
top = CurrentChunk(iff);
D0                 A0

struct ContextNode *CurrentChunk(struct IFFHandle *);

 FUNCTION
Returns the top context node for the given IFFHandle structure. The top
context node corresponds to the chunk most recently pushed on the
stack, which is the chunk where the stream is currently positioned.
The ContextNode structure contains information on the type of chunk
currently being parsed (or written), its size and the current
position within the chunk.

 INPUTS
iff - pointer to IFFHandle structure

 RESULT
top - pointer to top context node or NULL if none

 SEE ALSO
PushChunk(), PopChunk(), ParseIFF(), ParentChunk(),
<libraries/iffparse.h>
```

1.9 iffparse.library/EntryHandler

```
 NAME
EntryHandler -- add an entry handler to the IFFHandle context. (V36)

 SYNOPSIS
error = EntryHandler(iff, type, id, position, handler, object);
D0                   A0   D0    D1  D2        A1       A2

LONG EntryHandler(struct IFFHandle *, LONG, LONG, LONG,
                  struct Hook *, APTR);

 FUNCTION
Installs an entry handler vector for a specific type of chunk into
the context for the given IFFHandle structure. Type and id are the
longword identifiers for the chunk to handle. The handler is a client-
supplied standard Hook structure, properly initialized. position
tells where to put the handler in the context. The handler will be
called whenever the parser enters a chunk of the given type, so the
IFF stream will be positioned to read the first data byte in the
chunk. The handler will execute in the same context as whoever
called ParseIFF(). The handler will be called (through the hook)
with the following arguments:

  A0: the Hook pointer you passed.
  A2: the 'object' pointer you passed.
  A1: pointer to a LONG containing the value
    IFFCMD_ENTRY.

The error code your call-back routine returns will affect the parser
in three different ways:

Return value    Result
------------    ------
0:      Normal success;  ParseIFF() will continue
      through the file.
IFF_RETURN2CLIENT:  ParseIFF() will stop and return the value 0.
      (StopChunk() is internally implemented using
      this return value.)
Any other value:  ParseIFF() will stop and return the value
      you supplied. This is how errors should be
      returned.

 INPUTS
iff - pointer to IFFHandle structure.
type - type code for chunk to handle (ex. "ILBM").
id - ID code for chunk to handle (ex. "CMAP").
position- local context item position. One of the IFFSLI_#? codes.
handler - pointer to Hook structure.
object - a client-defined pointer which is passed in A2 during call-
   back.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 BUGS
```

```
Returning the values IFFERR_EOF or IFFERR_EOC from the call-back
routine *may* confuse the parser.

There is no way to explicitly remove a handler once installed.
However, by installing a do-nothing handler using IFFSLI_TOP,
previous handlers will be overridden until the context expires.

 SEE ALSO
ExitHandler(), StoreLocalItem(), StoreItemInContext(),
<utility/hooks.h>, <libraries/iffparse.h>
```

1.10 iffparse.library/ExitHandler

```
 NAME
ExitHandler -- add an exit handler to the IFFHandle context. (V36)

 SYNOPSIS
error = ExitHandler(iff, type, id, position, handler, object);
D0                  A0   D0    D1  D2        A1       A2

LONG ExitHandler(struct IFFHandle *, LONG, LONG, LONG,
                 struct Hook *, APTR object(;

 FUNCTION
Installs an exit handler vector for a specific type of chunk into the
context for the given IFFHandle structure. Type and id are the longword
identifiers for the chunk to handle. The handler is a client-supplied
standard Hook structure, properly initialized. Position tells
where to put the handler in the context. The handler will be called
just before the parser exits the given chunk in the "pause" parse
state. The IFF stream may not be positioned predictably within the
chunk. The handler will execute in the same context as whoever
called ParseIFF(). The handler will be called (through the hook)
with the following arguments:

  A0: the Hook pointer you passed.
  A2: the 'object' pointer you passed.
  A1: pointer to a LONG containing the value
    IFFCMD_EXIT.

The error code your call-back routine returns will affect the parser
in three different ways:

Return value    Result
------------    ------
0:      Normal success;  ParseIFF() will continue
      through the file.
IFF_RETURN2CLIENT:  ParseIFF() will stop and return the value 0.
      (StopChunk() is internally implemented using
      this return value.)
Any other value:  ParseIFF() will stop and return the value
      you supplied. This is how errors should be
      returned.

 INPUTS
```

```
iff - pointer to IFFHandle structure.
type - type code for chunk to handle (ex. "ILBM").
id - identifier code for chunk to handle (ex. "CMAP").
position - local context item position. One of the IFFSLI_#? codes.
handler - pointer to Hook structure.
object - a client-defined pointer which is passed in A2 during call-
   back.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 BUGS
Returning the values IFFERR_EOF or IFFERR_EOC from the call-back
routine *may* confuse the parser.

There is no way to explicitly remove a handler once installed.
However, by installing a do-nothing handler using IFFSLI_TOP,
previous handlers will be overridden until the context expires.

 SEE ALSO
EntryHandler(), StoreLocalItem(), StoreItemInContext(),
<utility/hooks.h>, <libraries/iffparse.h>
```

1.11 iffparse.library/FindCollection

```
 NAME
FindCollection -- get a pointer to the current list of collection
      items. (V36)

 SYNOPSIS
ci = FindCollection(iff, type, id);
D0                  A0   D0    D1

struct CollectionItem *FindCollection(struct IFFHandle *, LONG, LONG);

 FUNCTION
Returns a pointer to a list of CollectionItem structures for each of
the collection chunks of the given type encountered so far in the
course of parsing this IFF file. The items appearing first in the
list will be the ones encountered most recently.

 INPUTS
iff - pointer to IFFHandle structure.
type - type code to search for.
id - identifier code to search for.

 RESULT
ci - pointer to last collection chunk encountered with
     links to previous ones.

 SEE ALSO
CollectionChunk(), CollectionChunks(), <libraries/iffparse.h>
```

1.12 iffparse.library/FindLocalItem

```
 NAME
FindLocalItem -- return a local context item from the context stack.
                 (V36)

 SYNOPSIS
lci = FindLocalItem(iff, type, id, ident);
D0                  A0   D0    D1  D2

struct LocalContextItem *FindLocalItem(struct IFFHandle *,
                                       LONG, LONG, LONG);

 FUNCTION
Searches the context stack of the given IFFHandle structure for a
local context item which matches the given ident, type and id. This
function searches the context stack from the most current context
backwards, so that the item found (if any) will be the one with
greatest precedence in the context stack.

 INPUTS
iff - pointer to IFFHandle structure.
type - type code to search for.
id - ID code to search for.
ident - ident code for the class of context item to search for
  (ex. "exhd" -- exit handler).

 RESULT
lci - pointer to local context item, or NULL if nothing matched.

 SEE ALSO
StoreLocalItem(), <libraries/iffparse.h>
```

1.13 iffparse.library/FindProp

```
 NAME
FindProp -- search for a stored property chunk. (V36)

 SYNOPSIS
sp = FindProp(iff, type, id);
D0            A0   D0    D1

struct StoredProperty *FindProp(struct IFFHandle *, LONG, LONG);

 FUNCTION
Searches for the stored property which is valid in the given context.
Property chunks are automatically stored by ParseIFF() when
pre-declared by PropChunk() or PropChunks(). The StoredProperty
struct, if found, contains a pointer to a data buffer containing the
contents of the stored property.

 INPUTS
iff - pointer to IFFHandle structure.
type - type code for chunk to search for (ex. "ILBM").
```

```
id - identifier code for chunk to search for (ex. "CMAP").

 RESULT
sp - pointer to stored property, or NULL if none found.

 SEE ALSO
PropChunk(), PropChunks(), <libraries/iffparse.h>
```

1.14 iffparse.library/FindPropContext

```
 NAME
FindPropContext -- get the property context for the current state.
                   (V36)

 SYNOPSIS
cn = FindPropContext(iff);
D0                   A0

struct ContextNode *FindPropContext(struct IFFHandle *);

 FUNCTION
Locates the context node which would be the scoping chunk for
properties in the current parsing state. (Huh?)  This is used for
locating the proper scoping context for property chunks i.e. the
scope from which a property would apply. This is usually the FORM
or LIST with the highest precedence in the context stack.

If you don't understand this, read the IFF spec a couple more times.

 INPUTS
iff - pointer to IFFHandle structure.

 RESULT
cn - ContextNode of property scoping chunk.

 SEE ALSO
CurrentChunk(), ParentChunk(), StoreItemInContext(),
<libraries/iffparse.h>
```

1.15 iffparse.library/FreeIFF

```
 NAME
FreeIFF -- deallocate an IFFHandle structure. (V36)

 SYNOPSIS
FreeIFF(iff);
        A0

VOID FreeIFF(struct IFFHandle *);

 FUNCTION
Deallocates all resources associated with this IFFHandle structure.
```

```
The structure MUST have already been closed with CloseIFF().

 INPUTS
iff - pointer to IFFHandle structure to free. Starting with V39,
      this may be NULL.

 SEE ALSO
AllocIFF(), CloseIFF(), <libraries/iffparse.h>
```

1.16 iffparse.library/FreeLocalItem

```
 NAME
FreeLocalItem -- deallocate a local context item structure. (V36)

 SYNOPSIS
FreeLocalItem(localItem);
              A0

VOID FreeLocalItem(struct LocalContextItem *);

 FUNCTION
Frees the memory for the local context item and any associated user
memory as allocated with AllocLocalItem(). User purge vectors should
call this function after they have freed any other resources
associated with this item.

Note that FreeLocalItem() does NOT call the custom purge vector set
up through SetLocalItemPurge(); all it does is free the local context
item. (This implies that your custom purge vector would want to call
this to ultimately free the LocalContextItem.)

 INPUTS
localItem - pointer to LocalContextItem created with AllocLocalItem.
            Starting with V39, this may be NULL.

 SEE ALSO
AllocLocalItem(), <libraries/iffparse.h>
```

1.17 iffparse.library/GoodID

```
 NAME
GoodID -- test if an identifier follows the IFF 85 specification. (V36)

 SYNOPSIS
isok = GoodID(id);
D0            D0

LONG GoodID(LONG);

 FUNCTION
Tests the given longword identifier to see if it meets all the EA IFF
85 specifications for a chunk ID. If so, it returns non-zero,
```

```
otherwise 0.

 INPUTS
id - potential 32 bit identifier.

 RESULT
isok - non-zero if this is a valid ID, 0 otherwise.

 SEE ALSO
GoodType()
```

1.18 iffparse.library/GoodType

```
 NAME
GoodType -- test if a type follows the IFF 85 specification. (V36)

 SYNOPSIS
isok = GoodType(type)
D0              D0

LONG GoodType(LONG);

 FUNCTION
Tests the given longword type identifier to see if it meets all the
EA IFF 85 specifications for a FORM type (requirements for a FORM
type are more stringent than those for a simple chunk ID). If it
complies, GoodType() returns non-zero, otherwise 0.

 INPUTS
type - potential 32 bit format type identifier.

 RESULT
isok - non-zero if this is a valid type id, 0 otherwise.

 SEE ALSO
GoodID()
```

1.19 iffparse.library/IDtoStr

```
 NAME
IDtoStr -- convert a longword identifier to a null-terminated string.
           (V36)

 SYNOPSIS
str = IDtoStr(id, buf);
D0            D0  A0

STRPTR IDtoStr(LONG, STRPTR);

 FUNCTION
Writes the ASCII equivalent of the given longword ID into buf as a
null-terminated string.
```

```
 INPUTS
id - longword ID.
buf - character buffer to accept string (at least 5 chars).

 RESULT
str - the value of 'buf'.
```

1.20 iffparse.library/InitIFF

```
 NAME
InitIFF -- initialize an IFFHandle structure as a user stream. (V36)

 SYNOPSIS
InitIFF(iff, flags, streamHook);
        A0   D0     A1

VOID InitIFF(struct IFFHandle *, LONG, struct Hook *);

 FUNCTION
Initializes an IFFHandle as a general user-defined stream by
allowing the user to declare a hook that the library will call to
accomplish the low-level reading, writing, and seeking of the stream.
Flags are the stream I/O flags for the specified stream; typically a
combination of the IFFF_?SEEK flags.

The stream vector is called with the following arguments:

  A0: pointer to streamhook.
  A2: pointer to IFFHandle structure.
  A1: pointer to IFFStreamCmd structure.

The IFFStreamCmd packet appears as follows:

  sc_Command: Contains an IFFCMD_#? value
  sc_Buf:   Pointer to memory buffer
  sc_NBytes:  Number of bytes involved in operation

The values taken on by sc_Command, and their meaning, are as follows:

IFFCMD_INIT:
  Prepare your stream for reading. This is used for certain
  streams that can't be read immediately upon opening, and need
  further preparation. (The clipboard.device is an example of
  such a stream.)  This operation is allowed to fail;  any
  error code will be returned directly to the client. sc_Buf
  and sc_NBytes have no meaning here.
IFFCMD_CLEANUP:
  Terminate the transaction with the associated stream. This
  is used with streams that can't simply be closed. (Again,
  the clipboard is an example of such a stream.)  This
  operation is not permitted to fail;  any error returned will
  be ignored (best to return 0, though). sc_Buf and sc_NBytes
  have no meaning here.
IFFCMD_READ:
```

```
    Read from the stream. You are to read sc_NBytes from the
    stream and place them in the buffer pointed to by sc_Buf.
    Any (non-zero) error returned will be remapped by the parser
    into IFFERR_READ.
  IFFCMD_WRITE:
    Write to the stream. You are to write sc_NBytes to the
    stream from the buffer pointed to by sc_Buf. Any (non-zero)
    error returned will be remapped by the parser into
    IFFERR_WRITE.
  IFFCMD_SEEK:
    Seek on the stream. You are to perform a seek on the stream
    relative to the current position. sc_NBytes is signed;
    negative values mean seek backward, positive values mean seek
    forward. sc_Buf has no meaning here. Any (non-zero) error
    returned will be remapped by the parser into IFFERR_SEEK.

  All errors are returned in D0. A return of 0 indicates success.
  UNDER NO CIRCUMSTANCES are you permitted to write to the IFFStreamCmd
  structure.

   INPUTS
  iff - pointer to IFFHandle structure to initialize.
  flags - stream I/O flags for the IFFHandle.
  streamHook - pointer to Hook structure.

   SEE ALSO
  <utility/hooks.h>, <libraries/iffparse.h>
```

1.21 iffparse.library/InitIFFasClip

```
   NAME
  InitIFFasClip -- initialize an IFFHandle as a clipboard stream. (V36)

   SYNOPSIS
  InitIFFasClip(iff);
                A0

  VOID InitIFFasClip(struct IFFHandle *);

   FUNCTION
  Initializes the given IFFHandle to be a clipboard stream. The
  function initializes the stream processing vectors to operate on
  streams of the ClipboardHandle type. The iff_Stream field will still
  need to be initialized to point to a ClipboardHandle as returned from
  OpenClipboard().

   INPUTS
  iff - pointer to IFFHandle structure.

   SEE ALSO
  InitIFF(), OpenClipboard(), <libraries/iffparse.h>
```

1.22 iffparse.library/InitIFFasDOS

```
 NAME
InitIFFasDOS -- initialize an IFFHandle as a DOS stream. (V36)

 SYNOPSIS
InitIFFasDOS(iff)
             A0

InitIFFasDOS(struct IFFHandle *);

 FUNCTION
The function initializes the given IFFHandle to operate on DOS
streams. The iff_Stream field will need to be initialized as a BPTR
returned from the DOS function Open().

 INPUTS
iff - pointer to IFFHandle structure.

 SEE ALSO
InitIFF()
```

1.23 iffparse.library/LocalItemData

```
 NAME
LocalItemData -- get pointer to user data for local context item. (V36)

 SYNOPSIS
data = LocalItemData(localItem);
D0                   A0

APTR LocalItemData(struct LocalContextItem *);

 FUNCTION
Returns pointer to the user data associated with the given local
context item. The size of the data area depends on the "dataSize"
argument used when allocating this item. If the pointer to the item
given (localItem) is NULL, this function returns NULL.

 INPUTS
localItem - pointer to local context item or NULL.

 RESULT
data - pointer to user data area or NULL if localItem is NULL.

 BUGS
Currently, there is no way to determine the size of the user data
area; you have to 'know'.

 SEE ALSO
AllocLocalItem(), FreeLocalItem(), <libraries/iffparse.h>
```

1.24 iffparse.library/OpenClipboard

```
 NAME
OpenClipboard -- create a handle on a clipboard unit. (V36)

 SYNOPSIS
ch = OpenClipboard(unitNumber)
D0                 D0

struct ClipboardHandle *OpenClipboard(LONG);

 FUNCTION
Opens the clipboard.device and opens a stream for the specified unit
(usually PRIMARY_CLIP). This handle structure will be used as the
clipboard stream for IFFHandles initialized as clipboard streams by
InitIFFasClip().

 INPUTS
unitNumber - clipboard unit number (usually PRIMARY_CLIP).

 RESULT
ch - pointer to ClipboardHandle structure or NULL if unsuccessful.

 BUGS
This function had several bugs prior to V39.

     First bug was that if the clipboard.device couldn't open, two calls
to FreeSignal() were made with uninitialized values as parameters.
The result of this was a corrupt signal mask in the Task field.

Second bug was that OpenDevice() was called with an IO request that
didn't have a valid MsgPort pointer in it.

Third bug was that the two message ports allocated by the function
(ClipboardHandle->cbh_CBport and ClipboardHandle->cbh_SatisfyPort)
were not being initialized correctly and would cause a system crash
if a message ever got to either of them.

 SEE ALSO
InitIFFasClip(), CloseClipboard(), <libraries/iffparse.h>
```

1.25 iffparse.library/OpenIFF

```
 NAME
OpenIFF -- prepare an IFFHandle to read or write a new IFF stream.
           (V36)

 SYNOPSIS
error = OpenIFF(iff, rwMode);
D0              A0   D0

LONG OpenIFF(struct IFFHandle *, LONG);

 FUNCTION
```

```
Initializes an IFFHandle structure for a new read or write. The
direction of the I/O is given by the value of rwMode, which can be
either IFFF_READ or IFFF_WRITE.

As part of its initialization procedure, OpenIFF() calls the client-
supplied stream hook vector. The IFFStreamCmd packet will contain
the following:

  sc_Command: IFFCMD_INIT
  sc_Buf:   (Not applicable)
  sc_NBytes:  (Not applicable)

This operation is permitted to fail. DO NOT write to this structure.

 INPUTS
iff - pointer to IFFHandle structure. Starting with V39, this may be
      NULL, in which case IFFERR_NOMEM is returned.
rwMode - IFFF_READ or IFFF_WRITE

 RESULT
error - contains an error code or 0 if successful

 SEE ALSO
CloseIFF(), InitIFF(), <libraries/iffparse.h>
```

1.26 iffparse.library/ParentChunk

```
 NAME
ParentChunk -- get the nesting context node for the given chunk. (V36)

 SYNOPSIS
parent = ParentChunk(contextNode);
D0                   A0

struct ContextNode *ParentChunk(struct ContextNode *);

 FUNCTION
Returns a context node for the chunk containing the chunk for the
given context node. This function effectively moves down the context
stack into previously pushed contexts. For example, to get a
ContextNode pointer for the enclosing FORM chunk while reading a data
chunk, use: ParentChunk(CurrentChunk(iff)) to find this pointer.
The ContextNode structure contains information on the type of chunk
and its size.

 INPUTS
contextNode - pointer to a context node.

 RESULT
parent - pointer to the enclosing context node or NULL if none.

 SEE ALSO
CurrentChunk(), <libraries/iffparse.h>
```

1.27 iffparse.library/ParseIFF

```
 NAME
ParseIFF -- parse an IFF file from an IFFHandle structure stream. (V36)

 SYNOPSIS
error = ParseIFF(iff, control);
D0               A0   D0

LONG ParseIFF(struct IFFHandle *, LONG);

 FUNCTION
This is the biggie.

Traverses a file opened for read by pushing chunks onto the context
stack and popping them off directed by the generic syntax of IFF
files. As it pushes each new chunk, it searches the context stack
for handlers to apply to chunks of that type. If it finds an entry
handler it will invoke it just after entering the chunk. If it finds
an exit handler it will invoke it just before leaving the chunk.
Standard handlers include entry handlers for pre-declared
property chunks and collection chunks and entry and exit handlers for
for stop chunks - that is, chunks which will cause the ParseIFF()
function to return control to the client. Client programs can also
provide their own custom handlers.

The control flag can have three values:

IFFPARSE_SCAN:
  In this normal mode, ParseIFF() will only return control to
  the caller when either:
    1) an error is encountered,
    2) a stop chunk is encountered, or a user handler
       returns the special IFF_RETURN2CLIENT code, or
    3) the end of the logical file is reached, in which
       case IFFERR_EOF is returned.

  ParseIFF() will continue pushing and popping chunks until one
  of these conditions occurs. If ParseIFF() is called again
  after returning, it will continue to parse the file where it
  left off.

IFFPARSE_STEP and _RAWSTEP:
  In these two modes, ParseIFF() will return control to the
  caller after every step in the parse, specifically, after
  each push of a context node and just before each pop. If
  returning just before a pop, ParseIFF() will return
  IFFERR_EOC, which is not an error, per se, but is just an
  indication that the most recent context is ending. In STEP
  mode, ParseIFF() will invoke the handlers for chunks, if
  any, before returning. In RAWSTEP mode, ParseIFF() will not
  invoke any handlers and will return right away. In both
  cases the function can be called multiple times to step
  through the parsing of the IFF file.

 INPUTS
```

```
iff - pointer to IFFHandle structure.
control - control code (IFFPARSE_SCAN, _STEP or _RAWSTEP).

 RESULT
error - 0 or IFFERR_#? value or return value from user handler.

 SEE ALSO
PushChunk(), PopChunk(), EntryHandler(), ExitHandler(),
PropChunk(), CollectionChunk(), StopChunk(), StopOnExit(),
<libraries/iffparse.h>
```

1.28 iffparse.library/PopChunk

```
 NAME
PopChunk -- pop top context node off context stack. (V36)

 SYNOPSIS
error = PopChunk(iff);
D0               A0

LONG PopChunk(struct IFFHandle *);

 FUNCTION
Pops top context chunk and frees all associated local context items.
The function is normally called only for writing files and signals
the end of a chunk.

 INPUTS
iff - pointer to IFFHandle structure.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
PushChunk(), <libraries/iffparse.h>
```

1.29 iffparse.library/PropChunk

```
 NAME
PropChunk -- specify a property chunk to store. (V36)

 SYNOPSIS
error = PropChunk(iff, type, id);
D0                A0   D0    D1

LONG PropChunk(struct IFFHandle *, LONG, LONG);

 FUNCTION
Installs an entry handler for chunks with the given type and ID so
that the contents of those chunks will be stored as they are
encountered. The storage of these chunks follows the property chunk
scoping rules for IFF files so that at any given point, a stored
```

```
property chunk returned by FindProp() will be the valid property for
the current context.

 INPUTS
iff - pointer to IFFHandle structure (does not need to be open).
type - type code for the chunk to declare (ex. "ILBM").
id - identifier for the chunk to declare (ex. "CMAP").

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
PropChunks(), FindProp(), CollectionChunk(), <libraries/iffparse.h>
```

1.30 iffparse.library/PropChunks

```
 NAME
PropChunks -- declare many property chunks at once. (V36)

 SYNOPSIS
error = PropChunks(iff, propArray, numPairs);
D0                 A0   A1         D0

LONG PropChunks(struct IFFHandle *, LONG *, LONG);

 FUNCTION
Declares multiple property chunks from a list. The propArray argument
if a pointer to an array of longwords arranged in pairs, and has the
following format:

  TYPE1, ID1, TYPE2, ID2, ..., TYPEn, IDn

The argument numPairs is the number of pairs. PropChunks() just calls
PropChunk() numPairs times.

 INPUTS
iff - pointer to IFFHandle structure.
propArray - pointer to array of longword chunk types and identifiers.
numPairs - number of pairs in the array.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
PropChunk(), <libraries/iffparse.h>
```

1.31 iffparse.library/PushChunk

```
 NAME
PushChunk -- push a new context node on the context stack. (V36)

 SYNOPSIS
```

```
error = PushChunk(iff, type, id, size);
D0                A0   D0    D1  D2

LONG PushChunk(struct IFFHandle *, LONG, LONG, LONG);

 FUNCTION
Pushes a new context node on the context stack by reading it from the
stream if this is a read file, or by creating it from the passed
parameters if this is a write file. Normally this function is only
called in write mode, where the type and id codes specify the new
chunk to create. If this is a leaf chunk, i.e. a local chunk inside
a FORM or PROP chunk, then the type argument is ignored. If the size
is specified then the chunk writing functions will enforce this size.
If the size is given as IFFSIZE_UNKNOWN, the chunk will expand to
accommodate whatever is written into it.

 INPUTS
iff - pointer to IFFHandle structure.
type - chunk type specifier (ex. ILBM) (ignored for read mode or
       leaf chunks).
id - chunk id specifier (ex. CMAP) (ignored for read mode).
size - size of the chunk to create or IFFSIZE_UNKNOWN (ignored for
       read mode).

 RESULT
error - 0 if successful or an IFFERR_#? error code if not unsuccessful.

 SEE ALSO
PopChunk(), WriteChunkRecords(), WriteChunkBytes(),
<libraries/iffparse.h>
```

1.32 iffparse.library/ReadChunkBytes

```
 NAME
ReadChunkBytes -- read bytes from the current chunk into a buffer.
                  (V36)

 SYNOPSIS
actual = ReadChunkBytes(iff, buf, numBytes);
D0                      A0   A1   D0

LONG ReadChunkBytes(struct IFFHandle *, APTR buf, LONG);

 FUNCTION
Reads the IFFHandle stream into the buffer for the specified number
of bytes. Reads are limited to the size of the current chunk and
attempts to read past the end of the chunk will truncate. This
function returns positive number of bytes read or a negative error
code.

 INPUTS
iff - pointer to IFFHandle structure.
buf - pointer to buffer area to receive data.
numBytes - number of bytes to read.
```

```
 RESULT
actual - (positive) number of bytes read if successful or a
   (negative) IFFERR_#? error code if unsuccessful.

 SEE ALSO
ReadChunkRecords(), ParseIFF(), WriteChunkBytes(),
<libraries/iffparse.h>
```

1.33 iffparse.library/ReadChunkRecords

```
 NAME
ReadChunkRecords -- read record elements from the current chunk into
        a buffer. (V36)

 SYNOPSIS
actual = ReadChunkRecords(iff, buf, bytesPerRecord, numRecords);
D0                        A0   A1   D0              D1

LONG ReadChunkRecords(struct IFFHandle *, APTR, LONG, LONG);

 FUNCTION
Reads records from the current chunk into buffer. Truncates attempts
to read past end of chunk (only whole records are read; remaining
bytes that are not of a whole record size are left unread and
available for ReadChunkBytes()).

 INPUTS
iff - pointer to IFFHandle structure.
buf - pointer to buffer area to receive data.
bytesPerRecord  - size of data records to read.
numRecords - number of data records to read.

 RESULT
actual - (positive) number of whole records read if successful or a
   (negative) IFFERR_#? error code if unsuccessful.

 SEE ALSO
ReadChunkBytes(), ParseIFF(), WriteChunkRecords(),
<libraries/iffparse.h>
```

1.34 iffparse.library/SetLocalItemPurge

```
 NAME
SetLocalItemPurge -- set purge vector for a local context item. (V36)

 SYNOPSIS
SetLocalItemPurge(localItem, purgeHook);
                  A0         A1

VOID SetLocalItemPurge(struct LocalContextItem *, struct Hook *);

 FUNCTION
```

```
Sets a local context item to use a client-supplied cleanup (purge)
vector for disposal when its context is popped. The purge vector
will be called when the ContextNode containing this local item is
popped off the context stack and is about to be deleted itself. If
the purge vector has not been set, the parser will use FreeLocalItem()
to delete the item, but if this function is used to set the purge
vector, the supplied vector will be called with the following
arguments:

  A0: pointer to purgeHook.
  A2: pointer to LocalContextItem to be freed.
  A1: pointer to a LONG containing the value
    IFFCMD_PURGELCI.

The user purge vector is then responsible for calling FreeLocalItem()
as part of its own cleanup. Although the purge vector can return a
value, it will be ignored -- purge vectors must always work (best to
return 0, though).

 INPUTS
localItem - pointer to a local context item.
purgeHook - pointer to a Hook structure.

 SEE ALSO
AllocLocalItem(), FreeLocalItem(), <utility/hooks.h>
<libraries/iffparse.h>
```

1.35 iffparse.library/StopChunk

```
 NAME
StopChunk -- declare a chunk which should cause ParseIFF to return.
             (V36)

 SYNOPSIS
error = StopChunk(iff, type, id);
D0                A0   D0    D1

LONG StopChunk(struct IFFHandle *, LONG, LONG);    type;

 FUNCTION
Installs an entry handler for the specified chunk which will cause
the ParseIFF() function to return control to the caller when this
chunk is encountered. This is only of value when ParseIFF() is
called with the IFFPARSE_SCAN control code.

 INPUTS
iff - pointer to IFFHandle structure (need not be open).
type - type code for chunk to declare (ex. "ILBM").
id - identifier for chunk to declare (ex. "BODY").

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
```

```
StopChunks(), ParseIFF(), <libraries/iffparse.h>
```

1.36 iffparse.library/StopChunks

```
 NAME
StopChunks -- declare many stop chunks at once. (V36)

 SYNOPSIS
error = StopChunks(iff, propArray, numPairs);
D0                 A0   A1         D0

LONG StopChunks(struct IFFHandle *, LONG *, LONG);

 FUNCTION
(is to StopChunk() as PropChunks() is to PropChunk().)

 INPUTS
iff - pointer to IFFHandle structure.
propArray - pointer to array of longword chunk types and identifiers.
numPairs - number of pairs in the array.

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
StopChunk(), <libraries/iffparse.h>
```

1.37 iffparse.library/StopOnExit

```
 NAME
StopOnExit -- declare a stop condition for exiting a chunk. (V36)

 SYNOPSIS
error = StopOnExit(iff, type, id);
D0                 A0   D0    D1

LONG StopOnExit(struct IFFHandle *, LONG, LONG);

 FUNCTION
Installs an exit handler for the specified chunk which will cause the
ParseIFF() function to return control to the caller when this chunk
is exhausted. ParseIFF() will return IFFERR_EOC when the declared
chunk is about to be popped. This is only of value when ParseIFF()
is called with the IFFPARSE_SCAN control code.

 INPUTS
iff - pointer to IFFHandle structure (need not be open).
type - type code for chunk to declare (ex. "ILBM").
id - identifier for chunk to declare (ex. "BODY").

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.
```

```
 SEE ALSO
ParseIFF(), <libraries/iffparse.h>
```

1.38 iffparse.library/StoreItemInContext

```
 NAME
StoreItemInContext -- store local context item in given context node.
                      (V36)

 SYNOPSIS
StoreItemInContext(iff, localItem, contextNode);
                   A0   A1         A2

VOID StoreItemInContext(struct IFFHandle *, struct LocalContextItem *,
                            struct ContextNode *);

 FUNCTION
Adds the LocalContextItem to the list of items for the given context
node. If an LCI with the same Type, ID, and Ident is already
present in the ContextNode, it will be purged and replaced with the
new one. This is a raw form of StoreLocalItem().

 INPUTS
iff - pointer to IFFHandle structure for this context.
localItem - pointer to a LocalContextItem to be stored.
contextNode - pointer to context node in which to store item.

 SEE ALSO
StoreLocalItem(), <libraries/iffparse.h>
```

1.39 iffparse.library/StoreLocalItem

```
 NAME
StoreLocalItem -- insert a local context item into the context stack.
                  (V36)

 SYNOPSIS
error = StoreLocalItem(iff, localItem, position);
D0                     A0   A1         D0

LONG StoreLocalItem(struct IFFHandle *, struct LocalContextItem *,
                    LONG);

 FUNCTION
Adds the local context item to the list of items for one of the
context nodes on the context stack and purges any other item in the
same context with the same ident, type and id. The position argument
determines where in the stack to add the item:

IFFSLI_ROOT:
  Add item to list at root (default) stack position.
```

```
IFFSLI_TOP:
  Add item to the top (current) context node.
IFFSLI_PROP:
  Add element in top property context. Top property context is
  either the top FORM chunk, or the top LIST chunk, whichever
  is closer to the top of the stack.

Items added to the root context, or added to the top context before
the IFFHandle has been opened or after it has been closed, are put in
the default context. That is, they will be the local items found
only after all other context nodes have been searched. Items in the
default context are also immune to being purged until the IFFHandle
structure itself is deleted with FreeIFF(). This means that handlers
installed in the root context will still be there after an IFFHandle
structure has been opened and closed. (Note that this implies that
items stored in a higher context will be deleted when that context
ends.)

 INPUTS
iff - pointer to IFFHandle structure.
localItem - pointer to LocalContextItem struct to insert.
position - where to store the item (IFFSLI_ROOT, _TOP or _PROP).

 RESULT
error - 0 if successful or an IFFERR_#? error code if unsuccessful.

 SEE ALSO
FindLocalItem(), StoreItemInContext(), EntryHandler(), ExitHandler(),
<libraries/iffparse.h>
```

1.40 iffparse.library/WriteChunkBytes

```
 NAME
WriteChunkBytes -- write data from a buffer into the current chunk.
                   (V36)

 SYNOPSIS
error = WriteChunkBytes(iff, buf, numBytes);
D0                      A0   A1   D0

LONG WriteChunkBytes(struct IFFHandle *, APTR, LONG);

 FUNCTION
Writes "numBytes" bytes from the specified buffer into the current
chunk. If the current chunk was pushed with IFFSIZE_UNKNOWN, the size
of the chunk gets increased by the size of the buffer written. If
the size was specified for this chunk, attempts to write past the end
of the chunk will be truncated.

 INPUTS
iff - pointer to IFFHandle structure.
buf - pointer to buffer area with bytes to be written.
numBytes - number of bytes to write.

 RESULT
```

```
error - (positive) number of bytes written if successful or a
  (negative) IFFERR_#? error code if unsuccessful.

 SEE ALSO
PushChunk(), PopChunk(), WriteChunkRecords(), <libraries/iffparse.h>
```

1.41 iffparse.library/WriteChunkRecords

```
 NAME
WriteChunkRecords -- write records from a buffer to the current
         chunk. (V36)

 SYNOPSIS
error = WriteChunkRecords(iff, buf, recsize, numrec);
D0                        A0   A1   D0       D1

LONG WriteChunkRecords(struct IFFHandle *, APTR, LONG, LONG);

 FUNCTION
Writes record elements from the buffer into the top chunk. This
function operates much like ReadChunkBytes().

 INPUTS
iff - pointer to IFFHandle structure.
buf - pointer to buffer area containing data.
recsize - size of data records to write.
numrec - number of data records to write.

 RESULT
error - (positive) number of whole records written if successful
  or a (negative) IFFERR_#? error code if unsuccessful.

 SEE ALSO
WriteChunkBytes(), <libraries/iffparse.h>
```