Libraries

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: 11 Intuition Special Functions

There are several Intuition topics which, while not large enough to fill chapters of their own, nontheless deserve to be discussed. The subjects covered here include locking IntuitionBase, the Intuition memory functions AllocRemember() and FreeRemember(), using sprites with Intuition, and Intuition's special internal functions.

1.2 11 Intuition Special Functions / Locking IntuitionBase

It is sometimes necessary to examine the IntuitionBase structure. Items such as the address of the active screen and window, current mouse coordinates and more can be found there. It is never a good idea to simply read these fields, as they are prone to sudden change. The IntuitionBase structure must always be locked before looking at its fields.

It is necessary to inform Intuition that an application is about to examine IntuitionBase so that Intuition will not change any variables and IntuitionBase will remain static during the access. The call LockIBase() will lock the state of IntuitionBase so that it may be examined. During the time that the application has IntuitionBase locked, all Intuition input processing is frozen. Make every effort to examine IntuitionBase and release the lock as quickly as possible. The values in IntuitionBase are read-only. Applications should never write values to IntuitionBase.

```
    ULONG LockIBase( unsigned long dontknow );
```

LockIBase() is passed a ULONG (dontknow in the prototype above) indicating the Intuition lock desired. For all foreseeable uses of the call this value should be 0. LockIBase() returns a ULONG, that must be passed to

```
UnlockIBase() later to allow IntuitionBase to change once again.

Every call to LockIBase() must be matched by a subsequent call to
UnlockIBase():

    void UnlockIBase( unsigned long ibLock );

Set the ibLock argument to the value returned by the previous call to
LockIBase().

    About LockIBase().
    ------------------
    This function should not be called while holding any other system
    locks such as Layer and Layer_Info locks.  Between calls to
    LockIBase() and UnlockIBase(), you may not call any Intuition
    or other high-level system functions so it is best to copy the
    information you need and release the lock as quickly as possible.

    About IntuitionBase.
    --------------------
    Never, ever, modify any of the fields in IntuitionBase directly.
    Also, there are fields in IntuitionBase that are considered system
    private that should not be accessed, even for reading.  (Refer to the
    <intuition/intuitionbase> include file.) Application programs
    cannot depend on (and should not use) the contents of these fields;
    their usage is subject to change in future revisions of Intuition.
```

1.3 11 Special Functions / Easy Memory Allocation and Deallocation

```
Intuition has a pair of routines that enable applications to make multiple
memory allocations which are easily deallocated with a single call.  The
Intuition routines for memory management are AllocRemember() and
FreeRemember().  These routines rely upon the Remember structure to track
allocations.

 Intuition Helps You Remember    The Remember Structure
 How to Remember                 An Example of Remembering
```

1.4 11 / Easy Memory Allocation and Deallocation / Helps You Remember

```
The AllocRemember() routine calls the Exec AllocMem() function to perform
the memory allocation.  (Of course, the application may directly call Exec
memory functions, see the chapter "Exec Memory Allocation" for details.)

AllocRemember() performs two allocations each time it is called.  The
first allocation is the actual memory requested by the application.  This
memory is of the size and type specified in the call and is independent of
the second block of memory.  The second allocation is memory for a
Remember structure which is used to save the specifics of the allocation
in a linked list.  When FreeRemember() is called it uses the information
in this linked list to free all previous memory allocations at once.  This
is convenient since normally you would have to free each memory block one
```

at a time which requires knowing the size and base address of each one.

The AllocRemember() call takes three arguments:

```
APTR AllocRemember( struct Remember **rememberKey, unsigned long size,
                    unsigned long flags );
```

The rememberKey is the address of a pointer to a Remember structure. Note that this is a double indirection, not just a simple pointer. The size is the size, in bytes, of the requested allocation. The flags argument is the specification for the memory allocation. These are the same as the specifications for the Exec AllocMem() function described in the chapter on "Exec Memory Allocation".

If AllocRemember() succeeds, it returns the address of the allocated memory block. It returns a NULL if the allocation fails.

The FreeRemember() function gives the option of freeing memory in either of two ways. The first (and most useful) option is to free both the link nodes that AllocRemember() created and the memory blocks to which they correspond. The second option is to free only the link nodes, leaving the memory blocks for further use (and later deallocation via Exec's FreeMem() function). But, as a general rule, the application should never free only the link nodes as this can greatly fragment memory. If the link nodes are not required, use the Exec memory allocation functions.

The FreeRemember() call is as follows:

```
void FreeRemember( struct Remember **rememberKey, long reallyForget );
```

Set the rememberKey argument to the address of a pointer to a Remember structure. This is the same value that was passed to previous calls to AllocRemember(). The reallyForget argument is a boolean that should be set to TRUE. If TRUE, then both the link nodes and the memory blocks are freed. If FALSE, then only the link nodes are freed. Again, applications should avoid using the FALSE value since it can lead to highly fragmented memory.

1.5 11 / Easy Memory Allocation and Deallocation / How to Remember

To use Intuition's memory functions, first create an anchor for the memory to be allocated by declaring a variable that is a pointer to a Remember structure and initializing that pointer to NULL. This variable is called the remember key.

```
struct Remember *rememberKey = NULL;
```

Call AllocRemember() with the address of the remember key, along with the memory requirements for the specific allocation. Multiple allocations may be made before a call to FreeRemember().

```
memBlockA = AllocRemember(&rememberKey, SIZE_A,
                          MEMF_CLEAR | MEMF_PUBLIC);
if (memBlockA == NULL)
    {
```

```
        /* error: allocation failed */
        printf("Memory allocation failed.\n");
        }
    else
        {
        /* use the memory here */
        printf("Memory allocation succeeded.\n");
        }
```

AllocRemember() actually performs two memory allocations per call, one for the memory requested and the other for a Remember structure. The Remember structure is filled in with data describing the allocation, and is linked into the list to which the remember key points.

To free memory that has been allocated, simply call FreeRemember() with the correct remember key.

```
    void FreeRemember(&rememberKey, TRUE);
```

This will free all the memory blocks previously allocated with AllocRemember() in a single call.

1.6 11 / Easy Memory Allocation and Deallocation / The Remember Structure

The Remember structure is defined in <intuition/intuition> as follows:

```
    struct Remember
        {
        struct Remember *NextRemember;
        ULONG RememberSize;
        UBYTE *Memory;
        };
```

Generally, the Remember structure is handled only by the system. Here are its fields:

```
    NextRemember - The link to the next Remember structure.
    RememberSize - The size of the memory tracked by this node.
    Memory       - A pointer to the memory tracked by this node.
```

1.7 11 Intuition Special Functions / Current Time Values

The function CurrentTime() gets the current time values. To use this function, first declare the variables Seconds and Micros. Then, when the application call the function, the current time is copied into the argument pointers.

```
    void CurrentTime( ULONG *seconds, ULONG *micros );
```

See the DOS library Autodocs in the AmigaDOS Manual (Bantam Books) for more information on functions dealing with the date and time. The DOS library includes such functions as DateToStr(), StrToDate(), SetFileDate()

and CompareDates().

1.8 11 Special Functions / Using Sprites in Intuition Windows and Screens

Sprite functionality has limitations under Intuition. The hardware and graphics library sprite systems manage sprites independently of the Intuition display. In particular:

* Sprites cannot be attached to any particular screen. Instead, they always appear in front of every screen.

* When a screen is moved, the sprites do not automatically move with it. The sprites move to their correct locations only when the appropriate function is called (either DrawGList() or MoveSprite()).

Hardware sprites are of limited use under the Intuition paradigm. They travel out of windows and out of screens, unlike all other Intuition mechanisms (except the Intuition pointer, which is meant to be global).

Remember that sprite data must be in Chip memory to be accessible to the custom chips. This may be done with a compiler specific feature, such as the __chip keyword of SAS/C. Otherwise, Chip memory can be allocated with the Exec AllocMem() function or the Intuition AllocRemember() function, setting the memory requirement flag to MEMF_CHIP. The sprite data may then be copied to Chip memory using a function like CopyMem() in the Exec library. See the chapter "Graphics Sprites, Bobs and Animation" for more information.

1.9 11 Intuition Special Functions / Intuition and Preferences

The SetPrefs() function is used to configure Intuition's internal data states according to a given Preferences structure. This call relies on the Preferences system used in V34 and earlier versions of the OS. The old system has been largely superceded in Release 2. See the "Preferences" chapter for details. This routine is called only by:

* The Preferences program itself after the user changes Preferences settings (under V34 and earlier).

* AmigaDOS when the system is being booted up. AmigaDOS opens the devs:system-configuration file and passes the information found there to the SetPrefs() routine. This way, the user can create an environment and have that environment restored every time the system is booted.

The function takes three arguments:

```
struct Preferences *SetPrefs(struct Preferences *prefbuf,
                             long size, long realThing)
```

The prefbuf argument is a pointer to a Preferences structure that will be used for Intuition's internal settings. The size is the number of bytes

contained in your Preferences structure. Typically, you will use sizeof(struct Preferences) for this argument. The realThing argument is a boolean TRUE or FALSE designating whether or not this is an intermediate or final version of the Preferences. The difference is that final changes to Intuition's internal Preferences settings cause a global broadcast of NEWPREFS events to every application that is listening for this event. Intermediate changes may be used, for instance, to update the screen colors while the user is playing with the color gadgets.

About SetPrefs().

The intended use for the SetPrefs() call is entirely to serve the user. You should never use this routine to make your programming or design job easier at the cost of yanking the rug out from beneath the user.

Refer to the chapter "Preferences" for information about the Preferences structure and the new Preferences procedure calls used in Release 2.

1.10 11 Intuition Special Functions / Function Reference

The following are brief descriptions of the Intuition functions discussed in this chapter. See the Amiga ROM Kernel Reference Manual: Includes and Autodocs for details on each function call.

Table 11-1: Other Functions for Intuition

Function	Description
AllocRemember()	Allocate memory and track the allocation.
FreeRemember()	Free memory allocated with AllocRemember().
LockIBase()	Lock IntuitionBase for reading.
UnlockIBase()	Unlock IntuitionBase when done reading.
CurrentTime()	Get the system time in seconds and micro-seconds.
SetPrefs()	An Intuition internal function you should try to avoid.