VInstall — Install vertical retrace interrupt task

#include <Retrace.h> **Vert. Retrace Mgr**

OSErr	**VInstall(***vblTaskPtr* **);**	
QElemPtr	*vblTaskPtr* ;	address of a 14-byte VBLTask structure
	returns	16-bit Error Code; 0=no error

VInstall sets up to perform a task periodically. It installs an element into the vertical retrace task queue that will be executed as often as 60 times per second.

vblTaskPtr is the address of a 14-byte VBLTask structure. You must initialize the fields of the structure before making the call.

Returns: an error return code indicating success or failure of the function. It will be one of:

noErr	(0)	No error
vTypErr	(-2)	VBLTask.qType must be vType

Notes: The VBL (Vertical BLanking) interrupt occurs 60.15 times per second. Orginally, on the Macintosh Plus, this corresponded to the period when the video display beam was sweeping back up to the top of the screen after each refresh of the screen image. This 1/60-th second interval is often called a "tick" and occurs once every 16.66 ms.

With the advent of slots, a variety of screens are available, each with a potentially vertical retrace period. A slot-specific version of **VInstall**, **SlotVInstall** is now available. You should use **SlotVInstall** if you are trying to synchronize the execution of a task with the refresh rate of a particular device. **VInstall** is still useful as a way of performing periodic tasks based on ticks, though. A special system-generated interrupt that occurs 60.15 times per second handle the execution of tasks installed via **VInstall**.

The system initially installs its own set of VBL tasks, including code to move the mouse cursor, post disk-insert events, and the task that updates the Ticks variable. (Note that if you wait until this variable changes, and then take action quickly, you can draw smooth animation on the screen).

Before calling **VInstall**, you must prepare all fields of the VBLTask structure except for the qLink field:

qType Set this to vType (defined as 1 in OSUtil.h)

vblAddr Address of the routine to be executed.

vblCount Number of 1/60-th second (16.66ms) intervals to pass between calls. (**Note:** This gets decremented to 0 before your VBL task gets called. If you want to get another timeslice, you must set this back each time.)

vblPhase Ticks to skip before installing this task (usually 0). This is needed if you install several tasks and wish them to take place at different ticks (to avoid overloading the system).

The code of **the task is executed at interrupt time**, and so must be written with the following constraints:

- It must not call the Memory Manager functions directly or indirectly.
- It must not depend on handles of unlocked memory blocks to be valid.
- It must preserve the values of all registers except A0-A3 and D0-D3.
- If it accesses application globals, it must be sure that the A5 register is valid.
- It must not cause memory to move (by calling Memory Manager routines as discussed above, etc.).

The latter requirement is important. At interrupt time, you cannot depend on A5 to be the same as when the application is running. Since the application's "world" starts at A5, you must set it before accessing application variables. If you are running under Finder, simply use **SetUpA5** , **RestoreA5**, **CurrentA5** and **SetCurrentA5**. With MultiFinder, there's a fair chance that the value of the global variable CurrentA5 won't be *your* A5.

Though undocumented in IM, Apple Technical Notes state that upon entry to your VBL task, AO will point to the start of the VBLTask structure. Thus, if you store the correct value for A5 at an address relative to that structure, you can easily retrieve your A5, as in the following example.

Other ways to obtain a time slice include using the Time Manager functions **InsTime** and **PrimeTime**, or you can write a device driver in which the dNeedTime flag in the driver header is set. In that case, you will get a shot at execution each time any application calls **SystemTask**.

Example

```
#include <Retrace.h>
#define kInterval 6   /* set for 1 second between calls */

typedef struct VBLRec {
    VBLTask    myVBLTask;
    long       vblA5; /* 4 bytes before the VBLTask data */
} VBLRec, *VBLRecPtr;

/* Prototypes */
OSErr InstallVBL (void);
pascal long GetVBLRec (void);
void DoVBL (void);

/* Globals */
VBLRec gMyVBLRec    /* global VBL record */

pascal long GetVBLRec (void)
    = 0x2E88;

OSErr InstallVBL ()
{
    gMyVBLRec.myVBLTask.qType = vType;
    gMyVBLRec.myVBLTask.vblAddr = (ProcPtr) DoVBL;/* address of task */
    gMyVBLRec.myVBLTask.vblCount = kInterval; /* Set the interval */
    gMyVBLRec.vblA5 = (long) CurrentA5; /* Save app's A5 in structure */

    return VInstall((QElemPtr) &gMyVBLRec.myVBLTask);
}


/* ============= the VBL task code itself  =============*/

void DoVBL()
{
    long     curA5;
    VBLRecPtr recPtr;

    recPtr = (VBLRecPtr) GetVBLRec ();
    curA5 = SetA5 (recPtr->vblA5);/* read app A5 from structure and save */
                                   /* current value of A5 */
    /* ...  now that it's OK to access application variables, do the task ... */

    /* Reset vblCount so that this procedure executes again */
    recPtr->myVBLTask.vblCount = kInterval;
    curA5 = SetA5(curA5);
}
```