Embedded DefProc

```
/*
 * EmbeddedDefProc *
 * This unit contains "InstallDefProc", a procedure that allows a program to
 * use definition routines imbedded in the program's source, as opposed to
 * requiring a separate resource. This mechanism is useful for debugging most
 * defprocs, and for "hiding" them from the user.
 */

/* USING THE EMBEDDED DEF PROC:
 *
 *    To use the embedded def proc, you need to do the following steps:
 *
 *    1) Create a new project that you are going to use to test the CDEF, MDEF, LDEF
 *            etc. you are debugging
 *
 *    2) Create a resource file for this project.  Now create a resource of the type that
 *            you are trying to debug and give it id 128.
 *
 *    3) Using the resedit hex editor, fill this resource with a jump instruction,
 *            followed by 8 0's. (See comments on for InstallDefProc() below.) Save
 *            the file.
 *
 *    4) In the project file, add the source file that contains your code for your
 *            resource, the code to check test your resource, and of course the install
 *            def proc procedure.  A good thing to note here is that to make sure you
 *            rename the main of your resource to something like myresource_main
 *            etc.
 *
 *    5)  Now add the following as the first line in your main.
 *            InstallDefProc(CurResFile(),
 *                    'Myrtype',128,(Ptr)&myresource_main);
 *            where 'Myrtype' is actually the type of your resource, i.e. 'LDEF', 'CDEF'
 *             etc. and myresource_main is the main function of your code resource.
 *
 *
 *    Now when you run your test, you should be able to single step through the code
 *    using Think C's source level debugger.
 */

 // Assumes inclusion of <MacHeaders>
#include <Traps.h>

void InstallDefProc(short dpPath, ResType dpType, short dpID, Ptr dpAddr);
Boolean TrapAvailable( short theTrap);
void FlushCache(void);

typedef struct {
    short   jmpInstr;
    Ptr     jmpAddr;
} JmpRecord, *JmpPtr, **JmpHandle;


/*
 * TrapAvailable
 * Check whether a certain trap exists on this machine.
 */

Boolean TrapAvailable( short theTrap)
{
	TrapType	tType;
	short		numToolBoxTraps;

	// first determine the trap type

	tType = (theTrap & 0x800) > 0 ? ToolTrap : OSTrap;

	// next find out how many traps there are

	if (NGetTrapAddress( _InitGraf, ToolTrap) == NGetTrapAddress( 0xAA6E,
		 ToolTrap))
		numToolBoxTraps = 0x200;
	else
		numToolBoxTraps = 0x400;

	// check if the trap number is too big for the
	// current trap table

	if (tType == ToolTrap)
	{
		theTrap &= 0x7FF;
		if (theTrap >= numToolBoxTraps)
			theTrap = _Unimplemented;
	}

	// the trap is implemented if its address is
	// different from the unimplemented trap

	return (NGetTrapAddress( theTrap, tType) !=
			NGetTrapAddress(_Unimplemented, ToolTrap));
}


/*
 * FlushCache
 * Flush the CPU cache(s). This is required on the 68040 after modifying
 *	code.
 */

#define _CacheFlushTrap		0xA0BD

void FlushCache(void)
{
	if (TrapAvailable( _CacheFlushTrap))
		asm
		{
			dc.w _CacheFlushTrap
		}
}
```

```
/*
 * InstallDefProc works by looking for a resource of the desired resource
 * type and ID in the resource file specified by "dpPath"; if you're installing
 * definition routines at program startup, you can pass CurResFile() as the
 * first argument.  The defproc resource is then patched to point to the
 * procedure address given in "dpAddr".
 *
 * If the resource is not found in the resource file, a debugger trap is
 * executed.  To avoid this, you should create a 6-byte resource (it can be all
 * zeros) of the defproc's type and ID, and place it in your program's
 * resource file.
 *
 * Caveats:
 *
 * You probably don't want to install stub defprocs in the system resource file.
 * Also remember that there are reserved resource ID's from 0 to 127; you should
 * use resource ID's 128 or higher for your defprocs.
 *
 * Procedures that are installed should be in the main segment, and
 * InstallDefProc only need be called once.
 *
 * HOW IT WORKS:
 *
 * In the normal case, the system loads a procedure resource (an LDEF, for
 * example), and jumps to its beginning.  InstallDefProc provides this
 * functionality for procedures in your program by providing a 6-byte stub
 * resource, which contains $4EF9, followed by a long word.  The $4EF9 is a
 * 68000 long jump instruction, and the long word is the address to which to
 * jump.  So the system calls this dummy defproc, which in turn jumps to the
 * address passed.
 */

void InstallDefProc(short dpPath, ResType dpType, short dpID, Ptr dpAddr)
{
    JmpHandle    jH;
    short        savePath;

    savePath = CurResFile();
    UseResFile(dpPath);

    jH = (JmpHandle)GetResource(dpType, dpID);
    UseResFile(savePath);

    if (!jH)                        /* is there no defproc resource? */
        DebugStr("\pStub Defproc Not Found!");

    (**jH).jmpAddr = dpAddr;
    (**jH).jmpInstr = 0x4EF9;
    FlushCache();

    HUnlock((Handle)jH);
    MoveHHi((Handle)jH);
    HNoPurge((Handle)jH);    /* make this resource nonpurgeable */
}
```