Using Application Global Variables in Tasks

When a **Time Manager** task executes, the A5 world of the application that installed the corresponding task record into the **Time Manager** queue might not be valid (for example, the task might execute at interrupt time when that application is not the current application). If so, an attempt to read the application's global variables would return erroneous results because the A5 register would point to the application global variables of some other application. When a **Time Manager** task uses an application's global variables, it must therefore ensure that register A5 contains the address of the boundary between the application global variables and the application parameters of the application that launched it. The task must also restore register A5 to its original value before exiting.

It is relatively straightforward to read the current value of the A5 register when a **Time Manager** task begins to execute (using the **SetCurrentA5** function) and to restore it before exiting (using the **SetA5** function). It is more complicated, however, to pass to a **Time Manager** task the value to which it should set A5 before accessing its application's global variables. The reason for this is quite simple: neither the original nor the extended **Time Manager** task record contains an unused field in which the application could pass this information to the task. The situation here is unlike the situation with **Notification Manager** tasks or **Sound Manager** callback routines (both of which provide an easy way to pass the address of the application's A5 world to the task), but it is similar to the situation with VBL tasks.

One way to gain access to the global variables of the application that launched a **Time Manager** task from within that task is to pass to **InsTime** (or **InsXTime**) and **PrimeTime** the address of a structure, the first segment of which is simply the corresponding **Time Manager** task record and the remaining segment of which contains the address of the application's A5 world. For example, you can define a new data structure, a **Time Manager** information record, as follows:

```
typedef struct TMInfo {
    TMTask      atmTask;      //original and revised TM task record
    long        tmWakeUp;     //tmWakeUp in extended task record
    long        tmReserved;   //tmReserved in extended task record
    long        tmRefCon;     //space to pass address of A5 world
    } TMInfo, *TmInfoPtr;
```

Then you can install and activate your **Time Manager** task as illustrated in the following program.

```
// Listing: Passing address of the application's A5 world to Time Manager task
// Assuming inclusion of <MacHeaders>

void InstallTMTask (void);
pascal void MyTask (void);

typedef struct {                    // Time Manager information record
    TMTask      atmTask;            // original and revised TMtask record
    long        tmWakeUp;           // tmWakeUp in extended task record
    long        tmReserved;         // tmReserved in extended task record
```

```
    long        tmRefCon;       // space to pass address of A5 world
} TMInfo;

typedef TMInfo *TmInfoPtr;

void InstallTMTask (void)
{
    TMInfo myTMInfo;// a TM information record
    long   myDelay;  // delay value

    myDelay   = 2000;   // milliseconds to delay
    myTMInfo.atmTask.tmAddr    = MyTask; // get task address
    myTMInfo.tmWakeUp = 0;    // initialize tmWakeUp
    myTMInfo.tmReserved    = 0;    // initialize tmReserved
    myTMInfo.tmRefcon   = SetCurrentA5(); // store A5 world address
    InsTime((QElemPtr) &myTMInfo);     // install the info record
    PrimeTime((QElemPtr),
              &myTMTask,  // activate the task record
    myDelay);//
}
```

With the revised and extended Time Managers, the task is called with register A1 containing the address passed to **InsTime** (or **InsXTime**) and **PrimeTime**. So the **Time Manager** task simply needs to retrieve the TMInfo record and extract the appropriate value of the application's A5 world. The following program illustrates a sample task definition that does this.

```
// Listing: Defining a Time Manager task that can manipulate global variables
// Assuming inclusion of <MacHeaders>

typedef struct {                // Time Manager information record
    TMTask  atmTask;        // original and revised TMtask task record
    long    tmWakeUp;       // tmWakeUp in extended task record
    long    tmReserved;     // tmReserved in extended task record
    long    tmRefcon;       // space to pass address of A5 world
} TMInfo;

typedef TMInfo *TmInfoPtr;

pascal TmInfoPtr GetTMInfo (void)
    = 0x2E89;                           // MOVE.L A1,(SP)

pascal void MyTask (void);

pascal void MyTask() // for revised and extended TMs
{
    long        oldA5;                  // A5 when task is called
    TmInfoPtr   recPtr;

    recPtr = GetTMInfo();               // first get your record
    oldA5 = SetA5(recPtr->tmRefcon);    //set A5 to app's A5 world

    // do something with the application's globals in here
```

```
    oldA5 = SetA5(oldA5);              // restore original A5
                                       // and ignore result
}
```

The main reason that this technique works is that the revised and extended Time Managers do not care if the record whose address is passed to **InsTime** (or **InsXTime**) and **PrimeTime** is bigger than what they are expecting. If you use this technique, however, you should take care to retrieve the address of the task record from register A1 as soon as you enter the **Time Manager** task (because some compilers generate code that uses registers A0 and A1 to dereference structures).

Note: The technique illustrated in the previous program cannot be used with the original Time Manager because that version of the **Time Manager** does not pass the address of the task record in register A1. To gain access to your application's global variables when using the original Time Manager, you would need to store your application's A5 in one of the application's code segments (in particular, in the code segment that contains the **Time Manager** task). This technique involves the use of self-modifying code segments and is not recommended. Applications that attempt to modify their own 'CODE' resources may crash in operating environments that restrict an application's access to its own code segments (as, for example, in A/UX).