# Mac OS 8 Transitional Application

This paper describes a Mac OS 8 Transitional Application.

## 1.0 Definition

A Mac OS 8 Transitional Application is a well-written application that explicitly takes advantage of Mac OS 8 technologies. It comes in two flavors: binary compatible with System 7.x and source compatible with System 7.x

In this paper, it is assumed that both types of transitional applications comply with the guidelines presented in the accompanying document, *Mac OS 8 Compatible Application,* and, are thus, **Mac OS 8 Compatible**.

Furthermore, it is also assumed that developers considering this type of application are primarily concerned with preserving an existing source base and/or producing one binary that runs on System 7.x and Mac OS 8.

If you are **not** concerned with binary or source compatibility (i.e you are starting from scratch), see the *Mac OS 8 Only Application* document.

**Binary Compatible**

This type of application carefully checks its runtime environment for the availability of Mac OS 8 features. When the Mac OS 8 feature is not available, it falls back on System 7.x or disables the feature.

**Source Compatible**

This type of application is a Mac OS 8-Only binary which is built from a shared System 7.x - Mac OS 8 source base. This type of application does **not** have to dynamically check its runtime environment with the exception of truly optional features. It can assume Mac OS 8 features because it was explicitly built for Mac OS 8.

## 1.1 Binary Compatible

This type of application must not call any Mac OS 8 services on System 7.x. It is the responsibility of the application to check what features are available at runtime.

**Service Attribute Function**

The service attribute function is a new mechanism in Mac OS 8 which replaces one aspect of Gestalt -- finding out if particular features of a given technology are available. By convention, all Apple supplied technologies (and libraries) will have a well known service attribute function.

### 1.1.1 Dynamic Feature Determination

In Mac OS 8, applications can use a combination of CFM weak-linking and the service attribute function to determine the availability of features. By weak-linking against a well-known symbol (the service attribute function) to see if it has been resolved, the application can determine if the general service is available. By calling the service attribute function, it can further determine what specific features are available from this service. The following sample code demonstrates both mechanisms:

```
// GetMyEnvironment sets the gFMStatus global
// variable to one of the following values:
//
// kFMNotAvailable - Feature Mgr is not present.
// kFMWithoutSOM - Feature Mgr is present but
//     it does not support SOM-based extensions.
// kFMWithSOM - Feature Mgr is present and it does
//              support SOM-based extensions.

enum FMStatus {
    kFMNotAvailable,
    kFMWithoutSOM,
    kFMWithSOM
};

FMStatus gFMStatus = kFMNotAvailable;

void GetMyEnvironment( void )
{
  // weak-link check: see if symbol is resolved
```

```
  if (&FeatureMgrLibrary != kUnresolvedSymbolAddr)
  {
      gFMStatus = kFMWithoutSOM;

      // call service attribute function to see
      // if service and/or feature is available
      if (FeatureMgrLibrary(kFMSupportsSOM))
          gFMStatus = kFMWithSOM;
  }
}
```

### 1.1.2 Temporary Solution

The abovementioned dynamic checking mechanism is not generally available in the Mac OS 8 Developers Release: *Compatibility Edition* release. This mechanism is planned to be fully available in the subsequent developer release. The following mechanism is being provided as a temporary workaround to assist development only.

**TEMPORARY SOLUTION FOR Mac OS 8 Developers Release: *Compatibility Edition***

Call Gestalt and check the System Version. If it is not at least 8.0, then the application must fall back on System 7.x.

A temporary alternative to dynamic feature determination:

```
void GetMyEnvironment(void)
{
    ...
   Gestalt(gestaltSystemVersion, &sysVersion);
   if (sysVersion < 0x800)
      gFMStatus = kFMNotAvailable;
   else
      // initially assume SOM is not available
      gFMStatus = kFMWithoutSOM;
    ...
    ...
}
```

## 1.2 Source-Code Compatible

This type of application has at its disposal all of the preferred technologies available in System 7.x and Mac OS 8.

## 1.3 Transitional Technologies

Not all technologies are adoptable in a manner which does not impact a System 7.x source-code base. Some new Mac OS 8 technologies can be adopted in a stand-alone fashion with minimal impact on a preexisting code base. Some, however, depend on other new technologies, requiring more work and understanding to adopt in a preexisting code base. There are also some technologies which require a fundamentally different programming model -- these require significant restructuring of a preexisting code base.

**Developer Assumptions**

The basic assumption about both kinds of Transitional Applications is that the developer:

1. Would like to take advantage of some Mac OS 8 specific features.
2. Can not make pervasive changes to the source base at this time.
3. Generally, the more impact the change has on the source base the less likely it is to be considered.

**Technology List**

The following technologies “fit in” to the assumptions listed above:

1. Tasking
2. Timing Services
3. Synchronization Services
4. File Manager
5. Memory Performance Improvement

### 1.3.1 Tasking

Tasks are a fundamental building-block for Mac OS 8. Developers can take advantage of tasking for computation, file system calls, networking and general I/O. By following some guidelines, a System 7.x structured application can be adopted to use tasking in a straightforward manner.

**Benefits**

Simpler programming model for computation. Better perceived performance due to responsive user interface. Infrastructure for other Mac OS 8 transitional technologies. In addition, tasking allows parallel execution on multiprocessor systems.

**Limitations**

Tasking is not really very useful without some form of communications. At the very least, the creator of the task typically wants to know when the task is done. There are many technologies which facilitate communications under Mac OS 8. The guidelines and sample-code below describe some of the possible choices for communications and the impact to a System 7.x source code base.

Fortunately, the System 7.x API `WakeupProcess` allows a task to "communicate" with a cooperative task with very little modification to the traditional System 7.x event-loop.

**Adoption Choices**

1. **`WakeupProcess`** allows a task to unblock a cooperative task (the cooperative task will receive a NULL event as a result). The cooperative task could then inspect a shared global or a kernel synchronization primitive (a kernel `EventGroupID`, or a `KernelQueueID`) which indicated that the task has done some work.
2. **`WakeupProcess`** can be used with kernel synchronization services (e.g. Event Groups, etc.) to convey more information about why the cooperative task was unblocked. The cooperative task, in its nullEvent handling, could inspect the Event Group to determine what work was accomplished.
3. **Apple Events** allow the caller to unblock the cooperative task and send information (presumably the work the task accomplished) at the same time.

**Sample Code**

The following sample code demonstrates how a System 7.x structured application could use tasking and `WakeupProcess()`:

```
GetCurrentProcess(&myparameters.psn);
myparameters.pi_result = 0;

WaitNextEvent(...);

switch (event.what) {
    case kHighLevelEvent:
```

```
            AEProcessAppleEvent(&event);
            break;

        case nullEvent:
            // check to see if my task is done
            if (myparameters.pi_result)
                displayPI();
            break;
    }

    OSErr ComputePIHandler(const AppleEvent *theAE,
                           AppleEvent       *reply,
                           UInt32    handlerRefcon)
    {
           // do some stuff with the Apple Event
           .
           .
    #if BUILDING_FOR_SYSTEM7
           doPIcalculation();
    #else
           // this call returns immediately; the
           // computation occurs concurrently with
           // the main task
           CreateTask('pi  ',
                      CurrentKernelProcessID,
                     (TaskProc) &computePI,
                      &myparameters,
                     NULL,   // stackBase (default)
                     NULL,   // stackSize (default)
                     NULL,   // KernelNotification
                     NULL,   // TaskOptions
                     NULL);  // TaskID
    #endif
           .
           .
           return(noErr);
    }

    OSStatus computePI(void *param)
    {
       UInt32 result = 0;

       // do the computation
       result = doPIcalculation();
```

```
    // store the results
    ((struct myparms *)param)->pi_result = result;

    // wakeup application task with NULL event
    WakeUpProcess(&((struct myparms *)param)->psn);

    // returning will terminate task
    return(noErr);
}
```

### 1.3.2 Timing Services

**Benefits**

Mac OS 8 Timing services provide truly drift-free timing and necessary alternatives to spin looping in tasks.

**Limitations**

Several Mac OS 8 Timing Services (e.g. `DelayFor, UpTime`), were introduced in System 7.5.1 for PCI drivers. In Mac OS 8, they are available for all software.

**Adoption Choices**

1. `DelayFor` can be used to directly replace any explicitly coded delay or spin loop.
2. `DelayUntil` can be used to provide drift-free timing.

**Sample Code**

The following sample code shows how a function that's called from the main event loop of a System 7.x application and a preemptive task in a Mac OS 8 application can delay 100 milliseconds between iterations. In the System 7 version, the function just returns, expecting to be called from the main event loop repeatedly until time for another iteration. In the Mac OS 8 version, a `DelayUntil` is used between iterations, causing the task to block until time for the next iteration (instead of polling, as in the System 7 version). Note that the timing in both cases is drift-free.

```
#if BUILDING_FOR_SYSTEM7

    static UnsignedWide lastTime = {0, 0};
    UnsignedWide currentTime;
```

```
    Microseconds(&currentTime);
    if (U64Compare(U64Subtract(
                    currentTime,lastTime),
                   100000)) > 0)
    {
        DoTheWork();
        lastTime = currentTime;
    }
#else
    AbsoluteTime nextTime;
    UpTime(&nextTime);
    while (1) {
        DoTheWork();
        nextTime = U64Add(nextTime,
                          DurationToAbsolute(100 *
                          kDurationMillisecond));
        DelayUntil (&nextTime);
    }
#endif
```

### 1.3.3 Synchronization Services

**Benefits**

Synchronization services provide a way to signal that an event has occurred or to synchronize access to data shared by one or more tasks. If you don't create any additional tasks (“Tasking”, section 1.3.1.), you don't need these services. If two tasks update even a single shared word, you need to use synchronization.

For example, two different tasks execute the statement "`counter += 1`" simultaneously will produce incorrect results because of preemptive scheduling. This C statement could be compiled as three instructions:

1. load the value of counter into a register
2. increment the register
3. store the result back into counter.

If counter starts with value 0, and task A executes instructions 1, 2, 3 and then task B executes instructions 1, 2, 3, then counter ends with a value of 2. If counter starts with a value of 0 and task A executes instructions 1 and 2, then task B executes instructions 1, 2, and 3, then task A executes instruction 3, then counter ends with the value of 1. However, If both tasks use the `IncrementAtomic` call, correct results are guaranteed.

**Limitations**

The association of a synchronization service with the data it protects is up to you; the services don't enforce it. If a particular data structure is protected by a lock, nothing prevents you from modifying the data structure without first acquiring the lock — if you do that, the data structure will (probably) become corrupt.

You should avoid calling any of the blocking operations such as `BeginLockedSection` or `BeginWriteLockedSection` from your cooperative task. These calls can block for an arbitrary amount of time (until the task holding the lock releases it), and all user interaction is suspended during that time. If the task holding the lock has a bug and never releases it, the user interface will be completely hung unless the user aborts the currently running application. Instead, use the normal `WaitNextEvent` call to block your main task and use the `WakeupProcess` call from another task to wake up your cooperative task. The cooperative task can then use a non blocking call to check for availability of the lock or completion of the synchronized operation.

**Adoption Choices**

If you need to synchronize access to shared data or signal completion of events between tasks, you may not need to use synchronization services at all. Both Apple Events and the `WakeupProcess` call are available in both System 7 and Mac OS 8. Apple Events may be sent or received from any task, while `WakeupProcess` may be called from any task to wake up a cooperative task.

There are several sets of synchronization services, but they fall into three groups:

1. **Atomic operations** (such as `CompareAndSwapAligned`, `IncrementAtomicAligned`, and `BitXorAtomicAligned`) These perform an arithmetic or logical operation on a single word of memory atomically. They are very fast and never block, so they are safe to use from cooperative tasks as well as additional tasks. (**Note**: For compatibility with System 7.5.1 drivers, there are also versions of these

routines that operate on non-aligned data. These routines are much less efficient and should be avoided. All stack, heap, and static variables are aligned by default in Mac OS 8). Under System 7, these can be replaced with normal non-atomic operations.

2. **Simple locks, read/write locks, and counting semaphores**. You can use these in the usual way in preemptive tasks: acquire the lock, access the shared data, release the lock. If one task attempts to acquire the lock while another task holds it, the first task will block until the lock is available. Non-blocking variations of some of these calls are available; the non-blocking versions are safe to call from a cooperative task (but of course you must be prepared to deal with the possibility that you didn't get the lock). Under System 7, where the data is being accessed sequentially instead of being shared with multiple preemptively scheduled tasks, these calls can be omitted.

3. **Event flags**. These are used to signal the occurrence of one or more events. They are roughly analogous to `WakeupProcess`, but they can be used by any task to wake up a preemptive task (cooperative tasks should never block waiting for an event flag for the same reason that a cooperative task should never block waiting for a lock). Under System 7, these calls should be omitted.

**Sample Code**

```
// This routine can be called once to process all the chunks of data
// sequentially (on System 7) or from any number of tasks simultaneously
// to process chunks in parallel (on Mac OS 8). The variable
// "gChunkCount" is a global SInt32 that's initialized to zero.
// Each task will process chunks until all the chunks have been processed,
// and then SafeProcessChunks will exit. If multiple tasks are executing
//SafeProcessChunks, some will process some chunks, and some will
// process others. On a multiprocessor system, they could be processing
// chunks at the same time. The actual chunks of data aren't shown, but it's
// assumed they don't overlap and so no synchronization is needed for the
// chunk data itself.

#if BUILDING_FOR_SYSTEM7
   #define IncrementAtomicAligned(s) (*s)++
#endif

void SafeProcessChunks() {
```

```
   SInt32 chunkToProcess;

   while ((chunkToProcess =
         IncrementAtomicAligned (&gChunkCount))
               < kChunkCount)
      ProcessOneChunk(chunkToProcess);
   }
```

```
// The following code shows a function accessing a data structure. Under
// System 7, this function can only be called sequentially, from the main
// event loop. Under Mac OS 8, this function can be called from several
// preemptive tasks simultaneously, so the data structure must be protected
// by a lock. It's assumed that the lock has been initialized elsewhere.

struct MyStruct {
#if BUILDING_FOR_SYSTEM8
   Lock   theLock;
#endif
   ....
   }
typedef struct MyStruct MyStruct;

OSErr SafeUpdateMyStruct(MyStruct s) {
#if BUILDING_FOR_SYSTEM8
   if (BeginLockedSection(s.theLock) != noErr)
      return kInternalError; // structure corrupted
#endif
   UpdateMyStruct(s);
#if BUILDING_FOR_SYSTEM8
   if (EndLockedSection(s.theLock) != noErr)
      return kInternalError; // structure corrupted
#endif
   }
```

### 1.3.4 File Manager

**Benefits**

The File Manager has been completely rewritten for Mac OS 8. New features include object iteration, notification, and memory-mapped file access. Those applications which desire maximum performance should transition from the System 7.x calls to the new APIs.

**Limitations**

The Mac OS 8 File Manager uses a new data type, `FSObjectRef`, to refer to file system objects. The Mac OS 8 File Manager provides a call, `FSSpecGetFSObjectRef`, to convert a System 7.x `FSSpec` to a Mac OS 8 `FSObjectRef`. `FSSpecGetFSObjectRef`, however, requires that the passed `FSSpec` be well-formed; that is, that it contain no empty fields, no working directories, and no partial or full path names. Applications wishing to use `FSSpecGetFSObjectRef`should call `FSMakeFSSpec`to create well-formed `FSSpecs` prior to calling `FSSpecGetFSObjectRef`.

While `FSSpecs` can be readily translated to `FSObjectRefs`, other data types cannot necessarily be shared or translated between the two API's. For example, a file which has been opened with `FSpOpenDF`, a System 7.x API call, cannot be read using `FSStreamSimpleRead`, a Mac OS 8 API call, because the open file reference data types are different and no translation facilities are provided.

The System 7.x compatibility API is only available to cooperative tasks, so those applications wishing to take advantage of tasking (“Tasking”, section 1.3.1.) for file operations will need to use the new Mac OS 8 File Manager APIs.

**Adoption Guidelines**

There is no requirement that an application entirely adopt the Mac OS 8 File Manager. The best places to adopt the new Mac OS 8 File Manager without pervasive source code changes are those areas where File Manager use is relatively contained. Examples include completely reading/writing a document from/to disk, iterating through a folder, and searching. Those applications which maintain open files across several subroutine calls will require more fundamental changes because of the data type discrepancy issue previously mentioned in the Limitations section.

Many of the System 7.x API calls have direct equivalents in the Mac OS 8 File Manager. These simplified calls, located in FileManager.h, have been separated from the full Mac OS 8 File Manager implementation to give developers a starting point for converting their applications. Long-time developers of System 7.x will recognize the similarity of these calls to the System 7.x FSp(ec) API set.

**Sample Code**

The following code demonstrates how a document might be saved to disk using the new Mac OS 8 File Manager. This code assumes a rather simplistic saving model (open, write, close), but it is, after all, sample code. This code, with the proper addition of data synchronization, could be used from a preemptive task to save a document to disk while the cooperative

task is otherwise engaged. See the Tasking description in 1.3.1 for more details.

```
// It is assumed that the file fileSpec exists
OSStatus SaveDocument(const FSSpec *fileSpec,
LogicalAddress data, ByteCount dataSize)
{
     OSStatus status;
     FSObjectRef fileRef;

     // Convert the FSSpec to an FSObjectRef

     status = FSSpecGetFSObjectRef( fileSpec,
                                    &fileRef );
     if( status == noErr )
     {
     FSStreamObjID fileStream;

     // FSStreamOpen opes the file for exclusive
     // read/write access
     // Use FSStreamOpenWithConstraints() to
     // specify different access permissions.

     status = FSStreamOpen( fileRef, kFSDataFork,
                                    fileStream );
       if( status == noErr )
       {
          FSOffset  eof64;

         // The Mac OS 8 File Manager uses 64-bit
         // data types for files sizes and offsets.
         eof64.hi = eof64.lo = 0;

         // Clear the file of any existing data
         status = FSStreamSetAbsoluteEOF(
                                     fileStream,
                                         &eof );
         if( status == noErr )
         {
            FSForkPositionDescriptor writePos;
            ByteCount actualBytesWritten;
             FSOffset currentMark;
```

```
            writePos.positionOffset = 0;
            writePos.positionMode = kFSFromStart;

            // Write out our data. Note that the
            // last two arguments, like most
            // Mac OS 8 File Manager outputs, are
            // optional

            status = FSStreamSimpleWrite(
                                fileStream,
                                dataSize,
                                data,
                                &writePos,
                               &actualBytesWritten,
                                &currentMark );
        }
        (void) FSStreamClose( fileStream );

    }


    // All FSObjectRefs returned by the File
     // Manager need to be disposed of. Note that
     // this does _not_ delete the object on the
     // disk.


    (void) FSObjectRefDispose( fileRef );
   }

   return( status );

}
```

---

The following code demonstrates how a use an iterator to “walk” a folder and it's contained folders and tabulate the total sizes of all enclosed files.

```
OSStatus SumFileSizes(const FSSpec *folderSpec,
                      FSSize *totalFilesSize)
{
    OSStatus status;
    FSObjectRef folderRef;
```

```
totalFilesSize->hi = totalFilesSize->lo = 0;
status = FSSpecGetFSObjectRef( folderSpec,
                                &folderRef );
if( status == noErr )
{
    FSObjectIteratorObjID iterator;

    // Create an iterator which returns only
    // file objects and will automatically
    // traverse contained folders.
    status = FSObjectIteratorCreate( folderRef,
                            kFSIncludeFiles |
                kFSTraverseEmbeddedContainers,
                                    &iterator );
    if( status == noErr )
    {
      while( status == noErr )
      {
        FSObjectInformation objectInfo;

        // Iterate to the next file and get
        // aggregate property information.
        // The last two parameters to
        // FSObjectIterateOnce() are pointers
        // to the name and object ref for the
        // file, neither of which are
        // needed for this exercise.

           status = FSObjectIterateOnce(
                                    iterator,
                 kFSInfoCurrentReleasedVersion,
                    &objectInfo, NULL, NULL );
           if( status == noErr )
           {
             FSSize fileSize;

              fileSize = S64Add(
  objectInfo.info.fileInfo.dataForkSize,
  objectInfo.info.fileInfo.resourceForkSize );

           *totalFilesSize = S64Add(
                            *totalFilesSize,
```

```
                                                fileSize );
                    }
                }

                // E_EndOfIteration is an exception
                // returned when an iterator has
                // finished iterating with a folder.
                // This is similar to indexed
                // PBGetCatInfo() calls returning
                // fnfErr at the end of a folder.

                if( status == E_EndOfIteration )
                    status == noErr;

            (void) FSObjectIteratorDispose( iterator );
        }

        // All FSObjectRefs returned by the File
        // Manager need to be disposed of. Note that
        // this does _not_ delete the object on the
        // disk.

        (void) FSObjectRefDispose( folderRef );
    }

    return( status );
}
```

### 1.3.5 Memory Performance Improvement

The current Memory Manager data structures rely on handles to achieve efficiency in using the available memory, but at the cost of slower speeds. The Transitional Memory API, on the other hand, is designed for high performance on a virtual memory system, potentially at the risk of less space efficiency. The Transitional Memory API allows your application to continue using the familiar Memory Manager API (i.e. NewHandle, NewPtr, etc), but with a faster implementation.

**Note**: The Transitional Memory API is a subset of the Memory Mgr API and is not binary compatible with System 7.x. This transitional technology is **not** appropriate for binary-compatible applications that do dynamic feature checking.

**Benefits**

Your application will gain speed in its memory allocations.

**Limitations**

For compatibility reasons, the System Heap is always a traditional Memory Manager heap. Because there is not a perfect translation between the current data structures and the new data structures, there are some Memory Manager calls that are not supported, and should be removed from your code. These include:

- `InitZone`
- `GetApplLimit`
- `SetApplLimit`
- `MaxApplZone`
- `MoreMasters`
- `HandleZone`
- `PtrZone`
- `FreeMem`
- `MaxMem`
- `CompactMem`
- `ReserveMem`
- `PurgeMem`
- `SetGrowZone`
- `GZSaveHnd`
- `PurgeProcs`
- `MoveHHi`
- `HLockHi`
- `MaxBlock`
- `PurgeSpace`
- `SetApplBase`
- `InitApplZone`

These calls are No-ops (therefore, can be safely left in your code),

- `HLock`
- `HUnlock`
- `HPurge`
- `HNoPurge`
- `HSetRBit`
- `HClrRBit`
- `HGetState (returns 0)`
- `HSetState`

These calls access the system or temp mem heap, and are therefore not part of the Transitional Memory API. Your application can still use these calls, but they will not get the benefits of the Transitional Memory API.

- `NewHandleSys`
- `NewHandleSysClear`
- `NewEmptyHandleSys`
- `NewPtrSys`
- `NewPtrSysClear`
- `MaxBlockSys`
- `TempNewHandle`
- `TempMaxMem`
- `TempFreeMem`
- `ReallocateHandleSys`

**Note**: The Transitional Memory API data structures are completely different from the Memory Manager data structures. Your code MUST NOT depend on any of them.

The following two calls are supported in the Transitional Memory Manger API, but behave slightly differently than their Memory Manager counterparts:

**SetPtrSize**: this call does not necessarily just expand the data block in place. You will probably get a different pointer returned from this call.

**SetHandleSize**: This is the only call that can move a handle data block. Since there is no concept of a lock bit, be careful of dangling pointers when using this call.

**Adoption Guidelines**

**TEMPORARY SOLUTION FOR Mac OS 8 Developers Release: *Compatibility Edition***

Specifying that your application uses the Transitional Memory Manager API is accomplished by using a bit in the 'SIZE' resource:

The Transitional Memory Manager uses one of the bits in the 'SIZE' resource to determine if your application requires the traditional Memory Manager API vs the Transitional API. Set bit #2 (i.e. mask of 0x00000002) if your application uses the Transitional Memory Manager API. Note: in Types.r this bit is still "reserved".

Note: This is a workaround for **Mac OS 8 Developers Release**: ***Compatibility Edition*** only. Since adopting this technology is not binary

compatible with System 7.x, it is very likely that this type of application will have its own unique file type to designate it as such.

### 1.3.6 Navigation Services

Navigation Services replaces the traditional Standard File dialogs with a greatly improved user experience. The view presented is identical to that of the Finder, allowing users to more easily make the connection between files in the Finder and files in Open and Save dialogs. Searching and history mechanisms are also directly integrated for ease of navigation.

Navigation Services presents a standard dialog for confirmation of saving changes. Apple has long provided guidelines for these dialogs, but not a procedure for the dialog.

Both dialog versions and a stand-alone panel are provided.

**Benefits**

Significant user benefit is provided by adoption of Navigation Services due to its improved user interface. Since most features which have required dialog customization in the past are built in to Navigation Services, there is less need for the developer to customize, and less confusion for the user resulting from different dialog appearances in different applications.

The NavAskSaveChanges procedure removes the need to implement this dialog yourself.

Because of the NavigationPanel HIObject, it is possible to embed file system browsers in any kind of window or dialog. The methods of NavigationPanel may be overridden to modify or restrict its behavior as desired.

**Limitations**

Navigation Services in its current incarnation is not generalizable to the point of adding new columns or browsing data spaces outside the file system. This is a possible future direction. Navigation Services procedures use Apple Event descriptors to allow for later generalization.

**Adoption Guidelines**

Transitional applications can retain their current Standard File code bases and switch to Navigation Services when a run-time check indicates that it is present. Transitional applications should weak-link to the Navigation code fragment and test for the presence of its entry points.

Developers should customize as little as possible, in order to enhance the consistency of the user interface. Apple's technical support receives many questions relating to Standard File dialogs and it is likely that yours does as well. These support costs can be reduced by removing customizations which do not add sufficient user benefit to justify their support cost.

**Sample Code**

The following 3 samples demonstrate how a System 7.x structured application could use Navigation Services with a runtime check :

```
// Opening a File
// To let the user select a file to open,
// use the NavGetObject() routine:

OSErr err;
NavReplyRecord reply;
NavDialogOptions options;
NavTypeList types;

// default position
SetPt(&options.location, -1, -1);
GetIndTextObject(&options.defaultButtonLabel,
                 kMyTextObjectListID,
                 kMyOpenButtonLabelIndex);
GetIndTextObject(&options.banner,
                 kMyTextObjectListID,
                 kMyOpenBannerIndex);

options.customPanel = NULL;// for no customization
types[0] = typeFSS;        // return only FSSpecs
types[1] = 0;

err = NavGetObject(NULL,    // no default object
                &reply,
                &options,
                kMyOpenResourceID,
                types,
                NULL,       // no object filter
                NULL,       // no event procedure
                NULL);      // no context pointer

if (err == noErr)
{
```

```
   FSSpec target;
   AEDesc fileDesc;
   err = AECoerceDesc(&reply.selection, typeFSS,
                      &fileDesc);
   if (err == noErr)
   {
      err = AEGetDescData(&fileDesc, NULL, &target,
                          sizeof(FSSpec), NULL);
      AEDisposeDesc(&fileDesc);
      if (err == noErr)
         MyOpenFile(&target);
   }
}
```

---

```
// Saving a File
// To allow the user to select a location in which
// to save a file, call NavPutObject(). The setup
// code is similar to that for NavGetObject().

// The first parameter (fileToSave) is an AEDesc
// which specifies the file to be saved; it may be
// NULL for a first-time save, but the normal case
// in Mac OS 8 is that the file already exists.

err = NavPutObject(fileToSave,
                   &reply,
                   &options,
                   NULL,    // no event procedure
                   NULL);   // no context pointer

if (err == noErr)
{
   FSSpec target;
   AEDesc fileDesc;
   err = AECoerceDesc(&reply.selection, typeFSS,
                      &fileDesc);
   if (err == noErr)
   {
      err = AEGetDescData(&fileDesc, NULL, &target,
                          sizeof(FSSpec), NULL);
      AEDisposeDesc(&fileDesc);
      if (err == noErr)
```

```
            MySaveFile(fileToSave, &target);
      }
}
```

---

```
// Confirming Changes
// To ask the user whether to save changes to a
// document when it is closed or the application
// is quit, you can use the NavAskSaveChanges()
// procedure.

NavAskSaveChangesResult reply;
Point location;

SetPt(&location, -1, -1);

// applicationName & documentName are text objects
err = NavAskSaveChanges(applicationName,
                        documentName,
                        kNavSaveChangesClosingDocument,
                        &reply,
                        location,
                        NULL,    // no event procedure
                        0);      // no context pointer

if (err == noErr)
{
   switch (reply)
   {

      case askSaveChangesSave:
         MySaveFile(fileToSave);
         MyCloseFile(fileToSave);
         break;
      case askSaveChangesCancel:
         // don't do anything to the document
         break;
      case askSaveChangesDontSave:
         MyCloseFile(fileToSave);
         break;
   }
}
```

### 1.3.7 Appearances

The Appearance Manager provides services which enable customization of the Mac OS 8 system user interface. Developers can use the Appearance Manager to make their applications adopt a compatible appearance under different system user interface themes.

Some applications provide their own non-Macintosh "environment", such as some games and multimedia programs which take over the screen completely. These applications don't necessarily need to use the Appearance Manager. Such applications need only worry about UI pieces they bring up through the toolbox, such as dialogs and menus.

**Benefits**

By using the Appearance Manager, you can tailor your application to coordinate with the current system appearance. As we move to more variations of the user interface, applications which fail to use the Appearance Manager will be very obvious to the user, since their color and pattern choices will not match the current system theme. For example, most System 7.x applications with custom MDEFs and WDEFs will display menus with white background and black text, and System 7-style rectangular windows, while the rest of the world will have colored menus and non-square windows. By utilizing the Appearance Manager, you can help your application blend into the new world, which will get you noticed by your customers as a more "modern" application.

**Limitations**

The Appearance Manager cannot provide geometry information for use by custom definition procedures - it can only provide color and pattern support. Your application may still have different shaped windows or menus than the system if you use custom definition procs. You can however minimize the differences in color using the Appearance Manager calls. Once again, it is best to use standard defprocs whenever possible to ensure your application will have the same appearance as the system.

**Adoption Choices**

Every application which uses standard windows, menus or controls will receive some level of Appearance Manager support without modification. Beyond that, there are three main areas where you can add Appearance Manager support to your existing System 7 application:

- Outside of your application window content area, which includes any custom definition procedures (MDEFs, WDEFs, CDEFs, MBDFs.
- Within dialogs, wherever you might now have user items to draw grouping rectangles, default button rings or custom popup menu items.

- Within your application content area itself, if you use window headers, status areas, rulers, and palettes or if you wish to have document content mimic system appearance.

The implementation cost associated with these options varies widely. Each of these options is discussed in more detail below. For the typical application which just wants to use system colors and patterns or draw dialog groups, you should only have to make a few simple calls.

NOTE : If you must ship a hard-coded visual appearance and don't want to transition to adopting the Appearance Manager yet, use the guidelines defined in the *Apple Gray-Scale Appearance* document provided on the WWDC ’96 CD. Doing so will make your application compatible with the standard Mac OS 8 appearance. While your interface will not adapt automatically to other themes, the gray levels used in the default appearance will blend in well with most other themes because they lack strong colors which might clash. In general, you should avoid the use of strong colors in your interface, except where relevant to the content such as in games or graphics programs. Doing so will help prevent palette clashes with more colorful system themes in the future.

The range of adoption choices to transition to the Appearance Manager includes :

1. Do Nothing.
   If you use only standard definition procedures (no custom MDEFs, WDEFs, CDEFs, MBDFs), your application will automatically inherit the correct system appearance under Mac OS 8. If you do have custom defprocs and do nothing, they will not adopt the current system appearance automatically, and may not look good with some themes.
2. Update your custom definition procedures.
   Applications which do use custom definition procs (MDEFs, WDEFs, CDEFs or MBDFs) may use the Appearance Manager color and pattern query functions to modify their behavior so they appear more standard. For example, you can modify your custom MDEFs to use the current menu background pattern and text color from the Appearance Manager, instead of white backgrounds and black text when running under Mac OS 8.
3. Update your application's dialog window items.
   If you use a dialog `useritem` to draw dialog group rectangles, you may use the Appearance Manager `DrawThemePrimaryGroup` and

`DrawThemeSecondaryGroup` primitives to draw these groups with the current system appearance when Mac OS 8 is present. `DrawThemePrimaryGroup` is intended for top level grouping and may surround other sub-groups. Sub-groups should be drawn with `DrawThemeSecondaryGroup`

If you use a dialog `useritem` to draw separator lines in your dialog, you may use the `DrawThemeSeparator` primitive, rather than drawing a gray line.

If you use lists, you may use the `DrawThemeListBoxFrame` primitive to frame them, rather than simply using `FrameRect`.

You can draw or erase keyboard focus rings for your text boxes, list boxes and selection areas using `DrawThemeTextBoxFocus`, `DrawThemeListBoxFocus` and `DrawThemeGenericFocus` respectively. Pass `isActive` of true to draw the focus and false to erase it.

Do not assume the background pattern of your dialogs is white. It will not always be. If you have a `useritem` which does an erase when it really wants pixels to explicitly go to white, you should explicitly set the pen color to white and do a fill or paint operation. Likewise, don't erase to the background pattern by painting with white. You may end up with a white blotch in the middle of your dialog.

If you have PICTs which appear in dialogs, you should note that they will sometimes appear against a non-white background. If you wish your pict to appear correctly against a color dialog background, you should either use a paint program which can generate PICTs with lassoed areas or use a program like PictDetective to decode your pict and rebuild it with no background fill opcode. Alternatively, if you do wish your PICT to have a square white background, use a dark border pixel around the PICT so that the white background appears intentional, not like a white blotch in the dialog behind the PICT.

**4.** Adapt your document content areas to blend with the system.
If you have areas of your `contentRgn` where you wish to coordinate with the system patterns and colors, you may use the query functions `GetThemeColor` and `GetThemePixPat` to do so. For example, an application which provides a ruler could fill its area with the window header background pattern instead of white or gray.

If your application uses a window header, you may use the `DrawThemeWindowHeader` primitive to draw it. This will coordinate your window header with the current Finder window header area (where # items and disk space appear).

If you provide a status area adjacent to the scrollbars in your document windows, you may use the `DrawThemePlacard` primitive to draw the background of this area.

## 1.4 How To Build a Mac OS 8 Transitional Application

A Mac OS 8 Transitional Application requires the interfaces and libraries on the **Mac OS 8 Developers Release**: ***Compatibility Edition*** CD. This release will provide helpful feedback while you are compiling, linking and running your application.

### Interfaces

The Mac OS 8 version of the interfaces, like all Apple interfaces, are universal to all Apple software. These are the interfaces that Apple engineers use to write their software. The **Mac OS 8 Developers Release**: ***Compatibility Edition*** CD will include the latest version of our interfaces.

### Libraries

In addition to the interfaces on the CD, we will include stub libraries on the CD to link your application against. These libraries correspond to the different types of products you might build. They allow you to link against one library without having to know what specific library the service (and symbol) in question came from.

### Compiling Your Application

To link a Mac OS 8 Transitional application, use the **`BUILDING_FOR_SYSTEM7_AND_SYSTEM8`** compiler flag to indicate to the system that you are building an application which runs on both System 7.x and Mac OS 8. Compiling with this build flag ensure that you are not using System7 only interfaces.

### Linking Your Application

To link a Mac OS 8 Transitional application, use the **`AppSystem7orMacOS8.stubs`** library in your development environment.

**Running Your Application**

When running your application against the debug version of the system release (on the **Mac OS 8 Developers Release: *Compatibility Edition***), you may also encounter debugger breaks which detect unsupported or discouraged use patterns. This will help you determine how well your application will run.