Twilight II Generation 2 Module Format Reference

June 14, 1993 Revision
Featuring Twilight II v1.1
James R. Maricondo

Part of the Twilight II v1.1 Developer Kit

Introduction

If you feel some parts of this document are vague, let us know! We want to make everything as clear as possible for you! This document will be updated as time goes on to be clearer and more complete and to reflect any modifications to the format.

The Generation 2 Module Format (G2MF) represents a vast change from the way modules were previously called. Twilight I v1.0 brought a very simple module format (see the Twilight I developer kit for more information.) But along with that simplicity was lack of power, and lack of expansion ability. The G2MF, as implemented in Twilight II, represents a vast step foreward. It has been designed with all the present and potential future needs in mind. (Old modules are not directly compatible and need to be recompiled – sorry.)

At all times, try to keep your module changing the screen enough to prevent burn in! Use some descretion on this issue; when modules are automatically switched after a given amount of time (to be implemented in a future version of Twilight II) hopefully your module won't have to worry so much about protecting the screen (since the next module will probably change the screen in a different way).

In addition to the documentation contained within this document, please note that we also have provided sample sample module source code in C, Orca/M Assembly, and Merlin Assembly.

Very special thanks to the fantastic beta testers who contributed direct suggestions toward improving this document or the G2MF in general!

For the future we are looking into adding some IPC requests that modules can use to make modules featuring setup easier to write. This will be for the next significant revision of T2.

This documentation is provided only as interim specifications to serve you until the Twilight II Module Format filetype note (FTN) comes out from Apple Computer's Developer Technical Support center, hopefully this July.

Twilight II Module File Format

A Twilight II module is defined by a file with filetype $BC (Generic Load File — GLF) auxtype $4004 (suggested abbreviation: T2M) with a data fork containing a routine capable of handling at least a `BlankT2` message, and a resource fork containing a minimum of the following:

Resource Type	Resource ID	Description
`rLETextBox2` ($800B)	$0010DD01	Module specific message to be printed in the "About Module" dialog box.
`rIcon` ($8001)	$0010DD01	Module specific icon to be displayed in the "About Module" dialog box.
`rVersion` ($8029)	$00000001	Version resource for the module.
`rT2ModuleFlags` ($D001)	$0000001	Special resource similar to `rCDevFlags` in concept.

In addition, it is recommended the following are also present:

`rComment` ($802A)	$00000001	Comment for the user, for Finder 6.0.
`rComment` ($802A)	$00000002	Message to tell the user to access modules thru T2, not by double-click.

The about module window may eventually display your about LETextBox2 string in a textEdit control, so be prepared for this.

The Rez definition for the format of `rT2ModuleFlags` is as follows:

```
type rT2ModuleFlags {
      byte = $01;                  /* module flags version - use 1 */
      hex unsigned word;           /* module flags word */
      byte;                        /* enabled flag (unimplemented) */
      hex unsigned word;           /* minimum T2 version required */
      hex unsigned word;           /* reserved */
      pstring[25];                 /* module name */
};
```

Currently, these bits of the flag word of `rT2ModuleFlags` are defined and implemented:

Bit *0* ($0001) `fSetup`
The module supports setup. The module must be capable of receiving and doing something specific for `MakeT2`, `SaveT2`, `LoadSetupT2`, and `HitT2` (minimum.)

Bit *1* ($0002) `fFadeOut`
The module wants the previous screen to fade out before receiving a `BlankT2` message. After fading out, the SCBs, palettes, and pixel data will be set to NIL. There is no need for you to re-zero them if you have this bit set!

Bit *2* ($0004) `fFadeIn`
The module wants the saved screen to fade in after returning from a `BlankT2` message.

Bit *3* ($0008) `fGrafPort320`
This bit when set tells Twilight II to open a new port and then set all the SCBs to 320 mode before calling `BlankT2`. Twilight II will save the old port, open up a new port, set the current port to the new port, and then set all the SCBs to use palette $0, 320 mode, and then will set the `LocInfo` of the new port to have a `bounds` of (0,0,200,320) and a `visRgn` of the same size.

Bit *4* ($0010) `fGrafPort640`
This bit when set tells Twilight II to open a new port and then set all the SCBs to 640 mode before calling `BlankT2`. Twilight II will save the old port, open up a new port, set the current port to the new port, and then set all the SCBs to use palette $0, 640 mode, and then will set the `LocInfo` of the new port to have a `bounds` of (0,0,200,640) and a `visRgn` of the same size.

Bit *5* ($0020) `fLoadSetupBoot`
`LoadSetupT2` will be called right after the module is loaded (either at boot time, or when the CDev window is being closed), and `UnloadSetupT2` will be called only right before your module is being disposed of.

Bit *6* ($0040) `fLoadSetupBlank`
`LoadSetupT2` will be called right before `BlankT2`, and `UnloadSetupT2` will be called right after

`BlankT2`, *not* when the module has been just loaded.

Bit *7* ($0080) `fOpenRForkWriteEnabled`
Open the module's resource fork with read and write access instead of normal read access only before sending `MakeT2`. This bit is for special circumstances **only**. Usage is strongly discouraged whenever possible.

Bit *8* ($0100) `fMostCommonPalette`
Have Twilight II take a tally of which lines use which palette, and set all the SCBs to use the most commonly used palette.

Bit *9* ($0200) `fReqUsableScree`
The module requires a "usable" screen. (See discussion below.)

Bit *10* ($0400) `fLeavesUsableScreen`
The module leaves a "usable" screen.

Bit *11* ($0800) `fLeavesCycleScreen`
The module leaves a screen which can be color cycled by the next module,where applicable. Note: this is not implemented in Twilight II v1.1, but you should be ready for it.

Bit *12* ($1000) `fPrematureExit`
The module always exits *before* `movePtr` becomes true, when in random mode (e.g. Short Out, Color by Color.) This feature is not implemented in Twilight II v1.1 but is present to make switching modules after so many minutes much better in the future!

The minimum Twilight II version word of `T2ModuleFlags` is BCD, in the same format as toolbox version words in Apple IIgs technote 100. For example, $0101 represents Twilight II version 1.0.1. If the version of Twilight II is not great enough, Twilight II will display the module as dimmed. If the version of the `rT2ModuleFlags` resource is incorrect (i.e. not $01) or the Twilight II module has been inactivated (given an auxtype of $C004), then the module will not be displayed at all.

In addition, it should be noted that modules are free to put whatever else they want in their resource forks. Please put your setup controls in your resource fork if your module supports setup. Please use our defined resource types (i.e. `rT2ModuleWord`, `rByteArray`, etc.), where applicable, for consistency. Also, we suggest that you put as much of your data in resources as possible, for three reasons: 1) your module's data doesn't have to stay around in memory *all the time*, using valuable memory space (instead it can be loaded in LoadSetupT2, which can be called right before your module is called for super-memory-efficiency); 2) the advanced user can use a resource editor to modify your module data if necessary, and 3) resources should be used whenever and wherever possible because of the flexibility they offer.

Twilight II Module Messages

Twilight II modules are now sent "messages" to perform certain actions in the same way the Control Panel NDA sends action event codes to CDevs. (As a side effect, this makes modules a lot easier to write in "C" for you can just define them as CDevs for practical purposes — see the Orca/C sample module source for more information!) Currently there are seven defined action types. The only one modules are required to support is `BlankT2`. Support of the rest is optional, but recommended.

At *any* time, your module may call `MMStartUp` to get the ID it is running under. The ID returned from `MMStartUp` is what the data fork of the module was loaded with using the System Loader, so it is advisable to first create a few new modified auxID's to allocate all your memory with. Create as many auxID's as you wish, and do whatever you want with them, just don't delete the id returned from `MMStartUp` and don't use it to allocate extra memory.

When the data fork of a module is called and sent an action message, the stack is set up like this:

Inputs:

```
|previous contents|
|-----------------|
|    T2Result     |  Long - Result space.
|-----------------|
|    T2Message    |  Word - Action to perform.
|-----------------|
|     T2data1     |  Long - Action specific input.
|-----------------|
|     T2data2     |  Long - Action specific input.
|-----------------|
|     rtlAddr     |  3 bytes - Return address.
|-----------------|
```

The module must return control to Twilight II with the stack arranged as follows:

Outputs:

```
|previous contents|
|-----------------|
|    T2Result     |  Long - Action specific output.
|-----------------|
|     rtlAddr     |  3 bytes - Return address.
|-----------------|
```

Message 0: MakeT2

This message is sent only to modules which support setup, as indicated by their `T2ModuleFlags` resource. A `MakeT2` message is sent when the module's menu item is selected from the setup popup control in the setup window. It tells the module to create its setup-specific controls in the setup window. When the user selects the popup menu item for setup for your module, Twilight II loads the data fork of your module into memory (again if necessary) and calls `MakeT2`. It is the module's responsibility to position its controls below the setup window's psuedo-info bar.

Your module must return in `T2Result` (lo) the highest control ID (*not* resource ID!) of the controls it just created. This means that you must start numbering your control IDs with 1, going *consecutively through* the highest ID that must be returned in `T2Result` (lo). This highest ID number will be used by Twilight II when it is time to erase and dispose of the controls.

If you need to load the last saved setup configuration values of your module so you can set up your controls to reflect the current status of these flags, you must save the current resource file, set the current resource file to

T2Data2 (lo), read in or create, if necessary, your configuration flag resources, then restore the original resource file (probably that of your own module's resource fork.) Again, see our sample source if this sounds confusing. (You can create your config resources froms scratch at either MakeT2 or SaveT2; we recommend making new setup resources — if they don't already exist — during SaveT2.) You must create your setup resources the first time the user configures your module after installation or after deleting Twilight.Setup. The resource search path should be preset by Twilight II to: «Your Module », Twilight.II, ControlPanel, Sys.Resources.

Modules are free to use TextEdit controls in their setup dialogs without any problems or extra effort. Try to take care that the setup window may enlarge in the future when designing your control layout.

Just for your information, the horizontal line control always present in the setup window has QuickDraw II coordinates of (20,0,21,350).

All direct page space is **reserved** for use by Twilight II. The following parameters are passed on the stack:

T2Message = MakeT2 ($0000.)
T2Data1 = Window pointer of the setup modeless window.
T2Data2 (hi) = reserved (do *not* modify!)
T2Data2 (lo) = Resource file ID of the opened resource fork of Twilight.Setup.
T2Result (hi) = reserved (do *not* modify!)
T2Result (lo) = Highest control ID of module specific setup controls.

Control IDs in the range $07FEFFE0 through $07FEFFFF are currently reserved for use by the CDev and should *not* be used by modules.

Message 1: SaveT2

A SaveT2 message is passed to your module when your module is presently being configured, and the user clicks on the "update" control in the psuedo-info bar. Saving new configuration data was implemented in this fasion so that the user can choose not to save the new settings if a mistake is made somewhere. Typically in your SaveT2 handler you will first set the current resource file to Twilight.Setup, and then you will load in any existing configuration resources specific to your module and modify them to reflect the user's new changes. Don't forget to handle a first-case scenereo — the resources may not be there if your module was never configured before, so you might have to create them and then store the new values the user just chose. (See Twilight.Setup section below.) All parameters are reserved, but ones passed with MakeT2 are still valid for use. All direct page space is also **reserved** for use by Twilight II.

T2Message = SaveT2 ($0001.)
T2Data1 = reserved (do *not* modify!)
T2Data2 = reserved (do *not* modify!)
T2Result = reserved (do *not* modify!)

Message 2: BlankT2

BlankT2 is the one message modules are **required** to support. Modules are provided with a direct page of their own to be used in any manner. Do whatever you need to animate the screen! Note that the resource search path is undefined, and usually should be left that way (i.e. in most cases you should not be loading any resources in the BlankT2 handler!) Please see the special section "Special Notes on When Resources Are and Should Be Loaded" under "Setup Resources" for if your module has a valid reason to be loading data resources from your own resource fork during BlankT2 and not during LoadSetupT2.

Remember, that if you set the appropriate T2ModuleFlags bits, the screen might be already faded out when your BlankT2 handler gets control. Twilight II will automatically preserve and restore the user's original border

color for you and set the current color to black. Twilight II will also always open a new GrafPort for your use. `T2ModuleFlags` bits `fGrafPort320` and `fGrafPort640` will govern any special properties of this port. (See description above.) Otherwise you will just get whatever default port QuickDraw II gave to Twilight II when it called `OpenPort`.

To ease the creation of modules with high speed advanced animation effects that require the shadow screen to work their magic, Twilight II does all the work in securing shadow memory for all modules. Twilight II indicates that it was able to secure the shadowing screen for your module's use by turning on shadowing before calling your module. (So if you're drawing to the SHR screen directly, be sure to check if shadowing is on. If it is, **you must** use bank $01 SHR, else **you must** use bank $E1 SHR. If you're using QuickDraw II exclusively, you don't have to worry about checking which bank to use.) When shadowing is on and your `BlankT2` handler is called, $012000-A000 is guaranteed to be the same as $E12000-A000.

What if your module absolutely *requires* shadowing to function properly? This is okay – it is a tradeoff. What you should do in such a situation is first to check if shadowing is already on. If it is, do your stuff – modify any parts of $012000-A000 and $E12000-A000 and feel free to turn shadowing off and on if you need to for your animation (just make sure that the value of the `SHADOW` softswitch is the same when your module exits `BlankT2` as when it was first called.) However, if shadowing is not available (indicated by SHR shadowing turned off when your module gets control) you should exit back with an error string that the internal `DrawString` error module can show to the user.

Twilight II will also make sure the Font Manager is started before calling `BlankT2`.

The state of the system is as follows:

Port = A default new port, or a specific mode (see above.)
Pen/background color and pattern = Undefined, but the pen is hidden for you.
Resource file path = Undefined. (That set by the bottom application.)
Resource Application = Undefined except when the Twilight II main window is open and your module is called by the user moving to a blank now corner. In this case, the resource application is guaranteed to be that of the Control Panel NDA. (This supports modules storing resources that have to be loaded during blank in their own resource fork, using `T2ShareMemory` so they can be blanked while being configured.)
Module's resource fork = Not open and module not logged into Resource Manager, except if the module's setup window is open at the same time of blank.
Color tables = Table $0 set to default QuickDraw II palette, tables $1 through $F set to black ($000) if `fFadeOut` was set. If `fGrafPort320` or `fGrafPort640` is set then table $0 will be set to the default color table. Otherwise palettes are those set by the bottom application.
Current pixel data = Screen memory initialized to $00 if `fFadeOut` was set. Otherwise set to that of the bottom application.
Current screen mode = 320 (all SCBs ANDed with $7F) if `fGrafPort320` is set, 640 (all SCBs ORed with $80) if `fGrafPort640` is set, otherwise set to that of the bottom application.
Border color = Black.

`T2Message` = `BlankT2` ($0002.)
`T2Data1` = pointer to boolean movement flag (`movePtr`) indicating whether the module should return to T2 or not. The following are currently defined values for `movePtr`; all other values are reserved.

`$0000`	No movement has occurred; module should remain active unless returning to T2 with an error string.
`$0001`	The user has interacted with the computer; your module should now return normally to T2.
`$2-FFFE`	Reserved.
`$FFFF`	The module must exit because a specified number of minutes have elapsed in Random Mode, and Twilight II is moving onto the next selected random module. Note: this is not implemented in Twilight II v1.1 but you would be wise to make your modules aware of it for the future.

As soon as movePtr turns non-zero, the module is required to return to T2. (If Caps Lock "Lock" is on, and caps lock is down, your module can keep running forever, but movePtr also will not turn to 1 until caps lock is released.) If you don't return to T2 at least within 2 seconds after movePtr has become 1, then be sure you test your module well, as it has the potential to wreck havoc on Twilight II.
T2Data2 (hi) = reserved (do *not* modify!)
T2Data2 (lo) = bits defined as follows:

bmiBlankNow	$0001	Module is being called from "blank now"
bmiCycleColors	$0002	The previous module left a screen which can be color cycled. Note: this is not implemented in Twilight II v1.1, but you should be ready for it.

T2Result (lo 3 bytes) = handle to error c-string. If no errors occurred, pass NIL. Otherwise pass a handle (allocated using your memory ID) containing an error string that you'd like T2 to inform the user about. The error string must be a c-string. Up to one carriage return ($0D) may be embedded. This handle is passed to the internal DrawString error module – experiment with appropriate lengths of the string. Keep it as short as possible.
T2Result = optional flag byte (BlankFlag). Bits defined as follows (bmr = blank message result), with the rest reserved:

bmrNextModule $01000000
Skip to the next module. **Only** set this if you want to exit your module without movePtr becoming true. Also make sure that more than one module has been selected. If movePtr has become true or only one module is selected or you are being called from "blank now", do **not** set this.

bmrFadeIn $02000000
The SHR screen should be faded in after all. You don't have to set this bit if it is already set in the module flags word.

bmrLeavesUsableScreen $04000000
The module has left a "usable" screen after all. You don't have to set this bit if it is already set in the module flags word.

bmrLeavesCycleScreen $08000000
The module has left a screen which can be color cycled after all. You don't have to set this bit if it is already set in the module flags word. Note: this is not implemented in Twilight II v1.1, but you should be ready for it.

Message 3: LoadSetupT2

LoadSetupT2 tells your module to load any configuration or data resources. Remember that you are loading resources under someone else's memory ID, so be sure to DetachResource your resources immediately after LoadResource, and then to SetHandleID your detached resources to T2Data2 (hi) so that they will stay around long enough for you to use them in your BlankT2 handler. (Alternatively, you may also save the existing resource application, ResourceStartup (MMStartUp), load your resources, ResourceShutDown, and restore the old resource application. You still must detach your resources, but you don't have to bother with SetHandleID.)

Twilight II sets up the resource search path so that Twilight.Setup is on top. If you currently need to load *data* resources contained in your own resource fork, do an LGetPathname2 to find out your pathname (use fileNum = $0001), and open (and close) your resource fork when appropriate by yourself. Be sure to save and restore the previous current resource file. C code that does this follows. ID is your module's memory ID, as returned from MMStartUp.

```
word MyResFile, OldResFile;

OldResFile=GetCurResourceFile();
MyResFile=OpenResourceFile(1 /* read only */, NULL, LGetPathname2(ID, 0x0001));
CloseResourceFile(MyResFile);
SetCurResourceFile(OldResFile);
```

In most cases, all resources should be loaded at this time! This includes both configuration resources of yours in the Twilight.Setup file, and static (unchanging) data resources of yours in your own resource fork. If your module has sound effects, then it would be a good idea to store them as rSound resources and also load them at

this time. Keep track of your allocated memory handles yourself; be sure to dispose of them in the `UnloadSetupT2` handler. It is guaranteed that after `LoadSetupT2` is called, your module will remain in memory, in the same location, through the time `UnloadSetupT2` is called.

Please see the special section "Special Notes on When Resources Are and Should Be Loaded" under "Setup Resources" for when `LoadSetupT2` will get called, and if you should indeed load all your data resources (from your own resource fork) during it, or if your module qualifies as a special case that should load data resources during `BlankT2`. It probably doesn't, but read and be sure.

`T2Message` = `LoadSetupT2` ($0003.)
`T2Data1` = reserved (do *not* modify!)
`T2Data2` (hi) = reserved (do *not* modify!)
`T2Data2` (lo) = flag word. Presently only bit 0 is defined (lmi = load message input):
`lmiOverrideSound` $0001
1 = override sound, 0 = sound okay. If sound is overriden, you should not load any of your sounds into memory (to conserve memory.)
`T2Result` = bits defined as follows (lmr = load message result):
`lmrReqUsableScreen` $0001
Requires usable screen after all. You don't have to set this bit if it is already set in your module flags word.
`lmrFadeOut` $0002
Fade out after all. You don't have to set this bit if it is already set in your module flags word.
`lmrMostCommonPalette` $0004
Do most common palette (mcp) after all. You don't have to set this bit if it's already set in your module flags word.
`lmrPrematureExit` $0008
While not implemented in Twilight II v1.1, this bit is very important and you must support it. This bit must be set if your module plans to exit *before* `movePtr` becomes true. For instance, Mountains, Plasma, String Art, etc. should set this bit only when their "Quit After One" option is selected. You don't have to set this bit if it's already set in your module flags word (as it is for modules like Short Out and Color by Color, which always exit early in random mode.)

Message 4: UnloadSetupT2

`UnloadSetupT2` gives you the chance to dispose of any old memory handles that you previously had in memory for the entire time your module was selected. When you receive an `UnloadSetupT2` message, your module is about to be unloaded and disposed of, so make sure you don't leave any handles behind!

The resource search path is undefined.

`T2Message` = `UnloadSetupT2` ($0004.)
`T2Data1` = reserved (do *not* modify!)
`T2Data2` (hi) = reserved (do *not* modify!)
`T2Data2` (lo) = reserved (do *not* modify!)
`T2Result` = reserved (do *not* modify!)

Message 5: KillT2

`KillT2` gives you the chance to dispose of any memory handles that you previously had in memory during the time your module was being configured by the user. When you receive an `KillT2` message, your module is about to be unloaded and disposed of, so make sure you don't leave any handles behind!

The resource search path will be set to: « Your Module », `Twilight.Setup`, `Twilight.II`, `ControlPanel`, `Sys.Resources`.

T2Message = KillT2 ($0005.)
T2Data1 = reserved (do *not* modify!)
T2Data2 = reserved (do *not* modify!)
T2Result = reserved (do *not* modify!)

Message 6: HitT2

HitT2 gives you the chance to react immediately when the user clicks in any one of the controls in the setup window. It also gives Twilight II the chance to enable the save (called update in Twilight II v1.0) button.

T2Message = HitT2 ($0006.)
T2Data1 = handle to control in question.
T2Data2 = ID of the control.
T2Result (hi) = reserved (must currently return $0000.)
T2Result (lo) = boolean result value indicating if save control should be enabled based on the control hit.

Setup Resources

Twilight II also features a new way of saving module preferences. Each module can have its own custom preferences and the preferences from all modules can all exist simultaneously! In addition, the new preference manager was fully designed with multi-user AppleShare networks in mind as well. Preferences are now stored in `Twilight.Setup`, saved in the modules folder which exists in the same directory as the CDev when the CDev runs its `BootCDev` message handler.

Twilight II first checks for the `Twilight.Setup` file on boot. If the setup file can't be found, it is created and initialized with some default resource values.

Storage

Predefined resource type assigment for `Twilight.Setup` file:

Resource Type	Description
`rT2ExtSetup1` ($1001)	Reserved for internal CDev use. (Internal integer flags.)
`rT2ModuleWord` ($1002)	Available for any word-sized setup resources. (Unsigned word.)
`rT2String` ($1010)	Reserved to save pathnames of currently selected modules. (Psuedo-WStrings.)
`rByteArray` ($1012)	Available for any size setup resources. (unsigned char array.)

If your module supports user-configurable setup (as defined by the `fSetup` bit of the `T2ModuleFlags` word,) then it must store the user's currently selected module options in the `Twilight.Setup` file at the appropriate time. All custom resource types other than those above are **reserved.** All existing resource IDs are also **reserved.** This means that your module may use any resource IDs in any of the above resource types or any of the Apple defined resource types that is not already taken when your module receives a `SaveT2` message. If your module needs to store information that is suited by a predefined Apple resource type, then use the Apple type. For instance, YouDrawIt! and Movie Theater store the pathname of their currently selected files as `rWString` resources. If your module needs to store any word-sized configuration resources, then please use the `rT2ModuleFlags` resource type. If none of the above resource types or the Apple defined resource types suits your use, than please contact us and we will assign a new custom resource type that all modules will be able to take advantage of. It is imperative that you use resource names to keep track of your configuration resources. Load and save them by name, **not** by ID! When creating a new resource from scratch, use `UniqueResourceID` and then `SetResourceName`. The new resource name System 6 Resource Manager calls make this pretty easy.

Special Notes on When Resources Are and Should Be Loaded

`LoadSetupT2` is not always called on boot, and `UnloadSetupT2` is not always called right before your module is about to be shut down and disposed. The new logic governing when each of these messages is called depends on whether the boot disk is a SCSI hard disk or not. If the boot disk is not a SCSI hard disk, `LoadSetupT2` will be called during boot, and `UnloadSetupT2` will be called only before your module is about to be shut down and disposed (i.e. when Twilight II has been purged, or when the user has selected a new module.) If the boot disk is a SCSI hard disk, `LoadSetupT2` will be called when it is time to blank, right before calling `BlankT2`, and `UnloadSetupT2` will be called right after your `BlankT2` routine. This has the advantage of saving precious memory, while still making users without hard drives pretty happy. However, there are cases where this logic doesn't work the best it could. For these cases, two new bits of T2ModuleFlags have been defined: `fLoadSetupBoot`, and `fLoadSetupBlank`. You should use these bits in the situation where you think the internal logic described above isn't best for your module. For instance, the Tiler module only needs to load six bytes of configuration resources. Since this is very minimal, it keeps them in memory all the time the module is loaded, by setting `fLoadSetupBoot`. The YouDrawIt module, on the other hand, loads the active animation template file in `LoadSetupT2`. This file can use anywhere from $7D00 to $9A00 bytes of memory,

so it is not wise to make it stay in memory all the time under any situation. As such, the file is only loaded right before blanking, by setting `fLoadSetupBlank`.

It should be noted that when your module is called as a result of the user clicking the "blank now" button, Twilight II forces your setup to be called at blank (obviously.)

Likewise, there also are cases where it may be unwise for your module to load all its data resources from its resource fork at `LoadSetupT2` time. Say, for instance, you have a module that displays a random `rPString` from your resource fork. And say you have 1000 `pString` resources in your resource fork, but only several random ones will be used each time your module is called. It would be a waste to load in all 1000 when you're only going to use several random ones that change each time your module is called. In rare cases like this, I suggest you load the few resources you need in your `BlankT2` handler (and dispose of them before returning.) Note this process is **not** encouraged except in rare cases like the one above. For this reason, Twilight II does not have things all nice and spiffy for you to load your resources at `BlankT2` time. You must do all the dirty work. This isn't too much of a big deal – all you really have to do is:

```
OldRezFile = GetCurResourceFile
ResourceStartup(MyID)
RezFileID = OpenResourceFile(LGetPathname2)
load your resources …
blank your stuff until MovePtr = TRUE …
release/dispose your resources …
CloseResourceFile(RezFileID)
ResourceShutDown
SetCurResourceFile(oldFile)
return to Twilight II.
```

Miscellaneous Notes

Using Sound in Modules

Several guidelines have been established for modules using digitized sound effects played through the GS's Ensoniq sound chip. By following these, sound can be implemented with a minimal amount of effort, and in a consistent fashion that the user can control and understand. The following points comprise the first recommended way for using sound effects:

1. Make sure you have an option allowing the user to turn the sound effects off. This is important, even if your module has no other setup options. Preferably this option should allow the user to change the sound volume as well. For instance, have a control where the volume can be changed from 0 to 15. At zero, there are no sound effects. Users want a feature like this!
2. Note the newly defined `T2Data2` (lo) flag word passed to your module at `LoadSetupT2` time. If bit 0 of this flag is on, then you also should not play any sound effects. This is the global sound shutoff boolean flag, and you must honor it.
3. Keep your sounds stored as `rSound` resources. These can be loaded and detached at `LoadSetupT2` time and disposed of at `UnloadSetupT2`. If the user has requested not to have sound effects, or if the global sound shutoff flag is true, then you should not waste memory with loading in your sound resources from disk. You might want to consider allowing use of `rSound` resources in the Sounds folder, but that currently involves a lot of extra coding work!
4. You can use Apple's Sound control panel to play the sounds. In C:

```
SendRequest(6 /* srqPlaySoundSample */, stopAfterOne, NULL, TheRSoundHandle, NULL);
```

If you need to play several sounds at once or require greater flexibility than the above method offers, you may play the sounds yourself using the Sound Manager. Twilight II has several requests that make this easier for you. When calling Twilight II, send your request to "DYA~Twilight II~". We also have sample source code available that illustrates this method. Three Twilight II IPC requests were designed for sound:

`$9005 - t2StartupTools`

Note: if any errors occur during startup, no tools will be started and no memory will be allocated.

`dataIn` (lo)	Integer bit flags specifying which tools to start up. The following bits currently defined in Twilight II v1.1 ($0110) bit 0 = start SANE bit 1 = start Sound Manager
`dataIn` (hi)	Word UserID to allocate tool direct page memory with.
`dataOut`	Pointer to the following 4-byte structure: +00 (word) – receive count (used by Tool Locator) +02 (word) – any errors incurred in the startup

`$9006 - t2ShutdownTools`

Any tools started by `t2StartupTools` should be shutdown by this procedure. Their direct page memory will also be disposed.

`dataIn` (lo)	Integer bit flags specifying which tools to shut down. The following bits currently defined in T2 v1.1 ($0110) bit 0 = shutdown SANE bit 1 = shutdown Sound Manager
`dataIn` (hi)	Reserved (Pass zero)
`dataOut`	Reserved.

`$900D - t2CalcFreqOffset`

This request will take a `relPitch` value from an `rSound` header and convert it into a corresponding `freqOffset` to be used with the Sound Manager.

`dataIn` (lo)	`relPitch` value from `rSound` header.
`dataIn` (hi)	Reserved (Pass zero).
`dataOut`	Pointer to the following 4-byte structure: +00 (word) – received count (used by Tool Locator) +02 (word) – `freqOffset`

These three request should make handling your own sound effects with the Sound Manager much easier. Just remember to not use sound if the Sound Manager is already in use when you gain control!

Using Fonts in Modules

Twilight II now starts up the Font Manager before calling `BlankT2`. This means it is okay for your module to make Font Manager calls such as `InstallFont` and `SetPurgeStat`. However, you should be aware that some users may not have their system disk online at all times, and if you call `InstallFont` with the boot disk offline, the Font Manager will probably not be very complying. Be sure to react accordingly. In the future, Twilight II may start up the Font Manager at `LoadSetupT2` and `UnloadSetupT2` time to be more friendly to users with limited volumes, but at this time such action is not supported.

"Usable" Screens

A usable screen is defined as a screen which contains enough content to make it worth modifying by another module. For instance, Tiler leaves the screen in a state which Color by Color can work with. Short Out does not leave a usable screen, so if the screen is shorted out (to black) and another module is run directly afterward that requires a usable screen (like Tiler), then Twilight II will restore the screen before running Tiler.

If your module always requires or always leaves a usable screen, set the appropriate bit in the module flag word. If your module only sometimes requires or leaves a usable screen (such as snow, due to the clear screen option), then you can return this information at `LoadSetupT2` and/or `BlankT2` time.

We request that you follow the above guidelines so Random Mode can be enhanced now and even more in the future.

Low Memory Mode and What it Means to You

Low memory mode saves users 32k of memory. LMM affects one situation: when $012000-A000 has been allocated (in a handle exactly $8000 bytes in size) but SHR shadowing is off when it is time to blank the screen. Twilight II will *not* allocate shadowing in this situation if LMM is on, because both banks $01 and $E1 of SHR must be preserved.

LMM will be significantly changed in the future (and made more or less automatic.)

"Quit After One:" When You Need It, and How to Implement It

Some modules (e.g. Mountains, Plasma, String Art, Headlines, etc.) have options to quit to the next module in random mode after they have generated a one screen display. This type of option is to tie users over until a definite "In Random Mode, Exchange Modules After X Minutes" is implemented for the future.

If you would like to implement this option, make sure you set the appropriate usable screen bits for your modules. Then code your module like this:

1) Call `t2GetInfo` (request $9004 to "DYA~Twilight II~", `stopAfterOne+sendToName`) around the start of your module's `BlankT2` message to find the `count_selected_modules` word. Use a `DataIn` of NIL and a `DataOut` such as the following. (You also can use the `DataOut` supplied in our `T2.H` header file, but this one here is more efficient for reading only this one word.)

```
Word recvCount;
Word start_offset = $0002;      /* copy from this byte of the buffer */
Word end_offset = $0004;        /* to this byte of the buffer */
Word count_selected_modules;    /* # of selected modules */
```

2) When you think you should return to T2 to quit to the next module, you may only return with `bmrNextModule` in `T2Result` if `bmiBlankNow` (passed as an input at `BlankT2` time) is clear and `count_selected_modules` is equal to more than one.

Restoring or Saving the Original Screen

Twilight II has (obviously) saved the contents of the screen before your module was run. What if your module needs to restore the original screen? Or what if your module needs to reference the original screen? (For instance, Meltdown takes the original screen and reverses it after a few minutes; a few minutes later it restores the original screen back. Dissolve needs a copy of the original screen to reference in order to do its effect.) In these cases, you can make the Twilight II `t2GetBuffers` IPC request ($900B).

You may **not** dispose of the handles this call returns, nor may you modify the pixel data! Ignore the auxiliary buffer handle (which is normally not used), and the palette buffer handle (which is used only for background blanking.) Also, please note that this call returns the *original* screen. The original screen is not necessarily what was on screen before your module was called! For instance, in random mode, your module can be called after another one. Thus on screen is the previous module's display and in the buffer is the original user's screen. React and plan accordingly.

Here is C source code defining the structure returned by `t2GetBuffers`. `DataOut` should be set to a pointer to this structure. `DataIn` should be set to NIL.

```
#define t2GetBuffers    0x900Bu

/* DataOut structure for t2GetBuffers */
typedef struct getBuffersOut {
   Word recvCount;
   void ** shr_main_bufferH;              /* handle to main SHR buffer */
   void ** shr_aux_bufferH;               /* handle to aux SHR buffer */
   void ** palette_bufferH;               /* handle to palette buffer */
};
```

Conclusion

Well, that wasn't so bad, was it? The format is quite complex, but after you've dealt with it for a bit, it makes much more sense. Be sure to make full use of the sample source code we provide; it can help a lot! And if you run into any problems, remember that we can be reached on America Online (in our company support board available from the AUT utilities forum and AGS graphics and sound forum) and GEnie (in A2Pro category 29.) We'd be happy to help you write your modules! Your feedback is also welcome for extensions and modifications to the format, and to this documentation!

Coming in July: the Twilight II IPC documentation, listing all the details on the Twilight II inter-process-communication routines you can use in your programs and modules!

You also can contact us by phone (203.375.0837) and by USMail:

DigiSoft Innovations
P.O. Box 380
Trumbull, CT 06611-0380
Attn: Twilight II Tech Support

Or by electronic mail at the addresses listed in the Twilight II manual (e.g. digisoft@aol.com, etc.)

"Why Do I Need Twilight II?"

That's a very good question. If you're still not convinced that Twilight II will protect your valuable monitor, dazzle your friends and family, and not interfere with your work, listen to what some existing owners have said.

"It's fantastic, it's amazing, it's totally awesome!"

Andy Kress
North Kingstown, RI

"Got Twilight II last week—really nice! Thanks for a great product."

Bob Fischer
Jesup, GA

"Congratulations on developing a great piece of software ... Good work and I look forward to your future projects."

Sam King
Aurora, Ontario, CANADA

"MiniFireworks is really elegant, and Fireworks glorious ... The documentation and flexibility of the program is impressive!"

Kirk Hollingsworth
New York, NY

"Works like a charm."

Donald McIntosh
Lexington, MA

"Twilight II v1.1 is absolutely astounding ... I am very *impressed and proud to own it! Keep up your good work."*

Evan Trent
Norwich, VT

Order your copy today, and find out what everyone else is raving about!

DigiSoft Innovations

P.O. Box 380
Trumbull, CT 06611
Phone 203.375.0837