RB-MPW Plug-ins Kit

Peter Robinson <peter@pmpfr.co.uk>

version 1.1b3 (January 14, 2003)

Introduction

This kit allows an MPW user to build plug-ins suitable for use with REALbasic. REAL Software has never supported the use of MPW to compile REALbasic plug-ins, but it's perfectly possible[1] to do so, and that's where this kit comes in.

In older versions of the official REALbasic Plug-ins SDK, it was necessary to modify some of the SDK files to use MPW. In version 4.5a1, REAL Software incorporated my changes and this is now no longer necessary. If, for some reason, you wish to compile an older version of the SDK, this kit includes a StreamEdit script and some header files in the folder '`Deprecated`' to allow this. However, it is recommended that you use SDK version 4.5a1 or newer,[2] in which case you can safely throw away the '`Deprecated`' folder.

With SDK version 5a1, REAL Software introduced a new file format for plug-ins. It uses a directory structure containing standard PEF shared libraries ('for development') and a proprietary data-fork based file ('for deployment'). This change permits the use of plug-ins in the new Windows version of the REALbasic IDE.

The 5a1 SDK includes an application for converting resource-based plug-ins to the new format. It should be straightforward to generate 'structured directory plug-ins' with MPW, and I hope to implement this in future versions of my RB-MPW Plug-ins Kit. Meanwhile, you can use REAL Software's supplied translator to convert MPW generated plug-ins to the new format.

Acknowledgements

My thanks go to Thomas Tempelmann, without whom I would not have been able to solve the problem of static constructors. Thomas has made available a *Plugin Starter* ([6]) which addresses some of the shortcomings of the official SDK for CodeWarrior users.

What you need

REALbasic You need a working copy of REALbasic with which to test the plug-ins. I am using 4.0.1, but other versions should work fine. I have successfully run MPW-built plug-ins under REALbasic 3.2.1, 3.5.1, 4.0.1 and 4.5fc4.

The official SDK This kit works in conjunction with the official REALbasic Plug-ins SDK; it does not replace it. If you have made changes to the official SDK (especially to its directory structure or file names) then it is best to use a fresh copy.

I recommend that you use REALbasic Plug-ins SDK version 4.5a1 or newer,[2] although it is possible to use older versions.

MPW You will need a working installation of MPW. It should include PPCLink version 1.5.2 or newer.

Getting started

Before using this kit, you need to download the REALbasic Plug-ins SDK and you need a working installation of REALbasic itself (to test the plug-ins). Most of this manual assumes that you are using SDK version 4.5a1 or newer; if you wish to use an older version of the SDK, see the section 'Older SDKs'.

You should fix the '`REALbasic`' alias in the RB-MPW Plug-ins Kit so that it points to your copy of the REALbasic application folder (at the moment, it points to this folders on *my* hard drive!).

You should also place the RB-MPW Plug-ins Kit folder into the official SDK folder. If you do these two things (and don't rename the alias!) you won't need to modify the MPW makefiles included in this kit.

[1]For a real working example of a REALbasic plug-in compiled with MPW, see my CompfacePlugin, [2].

[2]At the time of writing, the current REALbasic Plug-ins SDK is version 5a1. It is compatible with this Kit.

When you've fixed the alias and put the Kit in the right place, launch MPW, change its working directory to the RB-MPW Plug-ins Kit folder and execute the supplied script called '`BuildPlugins`'. That will build the four supplied example plug-ins from the SDK, together with my own example plug-in. It makes no attempt at building the obsolete plug-ins.

If the compilation was successful, you can load into REALbasic the test projects that came with the official REALbasic Plug-ins SDK, and see if everything worked. You can also try my own included test projects, named '`MyExample Project`' and '`BoxControl/TestPlugin Project`'.

The supplied project '`MyExample Project`' is provided for testing '`MyPlugin`', which is built from '`MyExample.cpp`'. It is for testing whether static constructors and destructors get called correctly, and it uses calls to `DebugStr()` to do so. If you don't have a MacsBug installed, this may cause a 'crash' when launching REALbasic, in which case there is a switch available in '`MyExample.cpp`' to turn off the debugger breaks. Alternatively, you can remove '`MyPlugin`' from the REALbasic '`Plugins`' folder.

Apart from dropping into MacsBug whenever anything happens in the plugin, '`MyPlugin`' should return the number 25 to '`MyExample Project`', indicating that everything's working.

The rest of this manual contains a detailed description of this kit, including its limitations, and how it works. While you don't have to read it in order to use this kit, it may come in handy if things go wrong.

Please send any comments, bug reports or suggestions by email to `<peter@pmpfr.co.uk>`.

Files in this Kit

The following files should have been included with this kit:

`BuildPlugins` An MPW Shell script that uses the two supplied makefiles '`Examples.make`' and '`Minimal.make`' to build all the example plug-ins supplied in the official SDK.

`Examples.make` A makefile for building all the example plug-ins supplied in the REALbasic Plug-ins SDK.

`Minimal.make` A simpler makefile for building my own minimal example.

`MyExample.cpp` The source code for my minimal example plug-in.

`MyExample Project` A very simple REALbasic project that tests my minimal example plug-in.

`BoxControl/TestPlugin Project` A REALbasic project that tests the two supplied plug-ins '`BoxPlugin`' and '`TestPlugin`'.

`Objects` A folder ready to receive the object files produced by the compiler.

`REALbasic` An alias which you should replace by one of the same name pointing to the REALbasic folder—the folder containing the REALbasic application itself.

`Read Me` A brief text file introducing this kit.

`Deprecated` A folder containing the following files for use with SDK versions 4a1 or older:

This kit must be able to find the files in the official SDK. Therefore, it is expected that the kit is inside the SDK folder.

This kit requires that the official SDK is unmodified. In particular, you should not rearrange the directory structure of the SDK, or this kit may not work without modification. If you have made changes to the official SDK, it is probably best to install a fresh copy.

In order to copy any built plug-ins into REALbasic's '`Plugins`' folder, the alias file '`REALbasic`' should be fixed or replaced by an alias of the same name so that it points to the folder containing the REALbasic application itself.

Using the Kit

Once the two alias files have been fixed as described above, the kit is ready to be used. The makefiles are designed to be run from MPW's current working directory, so you should execute a Directory command to set this to the RB-MPW Plug-ins Kit folder.

Using '`Examples.make`'

The makefile '`Examples.make`' is used to build the example plug-ins supplied in the official SDK, and my minimal example. It uses MakeDepend to generate much of its dependency information, but this can be automated by building the high level target '`Deps`'.

Because the five plug-ins have so much in common, the makefile is designed for a two-pass strategy. There are high level targets that themselves generate Make commands for the lower level targets, in the process defining Make variables that contain information relevant to the particular plug-in.

When these lower level Make commands are executed, they generate the actual build commands used to compile the plug-ins. The makefile itself includes extensive comments that describe how it works.

The following script may be used to build all the plug-ins:

```
Make Deps -f Examples.make > Cmds
Cmds
Make Plugins -f Examples.make > Cmds
Cmds > Cmds2; Cmds2
```

The file '`BuildPlugins`' is an MPW Shell script that does this, echoing its progress to the MPW Worksheet. Therefore, after changing directory to the RB-MPW Plug-ins Kit directory, simply execute BuildPlugins.

Using '`Minimal.make`'

The makefile '`Minimal.make`' is a much simpler affair than '`Examples.make`'. Since it only builds one plugin (my minimal example) it has no need of a two-pass strategy, and can therefore be used in the usual way. As before, much of the required dependency information is generated by MakeDepend from the target '`Deps`'.

The script

```
Make Deps -f Minimal.make > Cmds
Cmds
Make MyPlugin -f Minimal.make > Cmds
Cmds
```

will build my minimal example plug-in from the file '`Minimal.make`'.

The Build Process

REALbasic plug-ins are PEF containers stored in code resources (see the the REALbasic Plug-ins SDK documentation for more information). This is not a natural format produced by MPW (or CodeWarrior for that matter), and the build process is slightly complicated as a result. However, the MPW script MakePPCCodeRsrc comes in very useful.

This section does not attempt to describe the working of the makefiles themselves; instead, it describes the build commands that they generate, and the reasoning behind them. To understand how the actual makefiles work, you should see the extensive commenting included within them.

This kit does not attempt to build plug-ins for 68k or Windows (see the section 'Limitations'). It builds only PPC and Carbon plug-ins.

Compiling

There is nothing unusual about how plug-in source files are compiled, but the command line options passed to MrC and MrCpp are summarised here.

We suppress unused parameter warnings (`-w 35`) and define symbols on the command line that indicate whether it's a Carbon or PPC build (`-d ...`). When compiling C source files, we require that function prototypes are given and turn ANSI conformance on (`-proto strict -ansi on`).

The compiler must search for header files in folders in the official SDK, so we set this up, by passing appropriate pathnames with the `-i` option. From version 1.1b1 of this kit, the makefiles uses a precompiled headers. This is built as required by the makefile, and we tell MrCpp to load it using the `-loadc` option. There is a different precompiled header for Carbon and PowerPC builds.

As well as compiling the source code from each plug-in, '`PluginMain.cpp`' must also be compiled.

The convention used is that object files for PPC builds have the extension '`.x`', while those for Carbon builds use '`.y`'.

Linking

The process of linking is a little more delicate—there are a few things that can be done incorrectly, which would prevent REALbasic from loading a plug-in.

There is nothing unusual about the selection of libraries to link against. The usual warnings about linking only against Carbon safe libraries for Carbon builds and vice-versa still apply. To use the default PEF initialization and termination routines whenever possible, you should link against '`MrCPlusLib.o`' and '`PPCCRuntime.o`'.

In addition to the libraries, the object files coming from each source file must be linked (including that from the SDK's '`PluginMain.cpp`').

Options are passed to PPCLink allowing it to use temporary memory for linking and not to emit warnings about duplicate symbols (`-mf -d`). This is not so important.

What *is* important is to understand the limitations of the kind of target we're building. In the case of PPC plug-ins, we must build what is known as an accelerated resource with a usage constant that identifies it as a 'drop-in addition' (as opposed to an application, shared library, stub library etc.). Accelerated resources are PowerPC PEF containers stored in a resource, but with an extra header which allows it to be called as a direct replacement for an old style 68k code resource, without making changes to the caller.

Unfortunately, this kind of 'accelerated resource' is not allowed to use PEF termination routines, nor may it use 'packed data'. If you do accidentally allow it to specify a PEF termination routine, REALbasic will be unable to load the plug-in—it will produce a Code Fragment Manager error -2824 in MacsBug (or most likely crash if the debugger is not installed).

Carbon plug-ins are not called accelerated resources because they don't include the extra header (and so they don't have a usage type either). Therefore they do not have this limitation, and we can allow PPCLink to use the default initialisation and termination routines.

In practise, the way to achieve all this is to give PPCLink the following arguments `-m main -term none -packdata off` and `-xm dropin` for PPC plug-ins, and `-m main -packdata off` for Carbon plug-ins. Note that PPCLink version 1.5.2 or newer is required[3] to set the PEF usage constant to that of a drop-in addition (`-xm -dropin`), but should be able to achieve the same effect by using ModPEF and an older version of PPCLink instead.

For more information on the PPC termination routine problem, see the section 'Limitations'. For information about PEF containers and PPC code resources, see [8].

Making the Code Resources

The MPW script MakePPCCodeRsrc is used to convert data fork based PEF containers into PPC code resources. The resource types used are '`PLPC`' for PPC plug-ins and '`PLCN`' for Carbon plug-ins, and it is usual to give the code resources IDs of 128. The plug-ins should have a file type of '`RBPl`' and a creator of '`RBv2`'. All this is accomplished by specifying for instance `-rt PLCN=128 -t RBPl -c RBv2` as parameters to MakePPCCodeRsrc, to make a PPC plug-in resource.

The crucial difference between Carbon and PPC plug-in resources is that while PPC plug-ins should begin with a header describing the main entry point, Carbon plug-ins are just raw PEF containers. Therefore, we specify `-rawpef` as an option for Carbon plug-ins but instead `-procinfo 193` for PPC plug-ins. The given number, 193, describes the signature of the main entry point (which is `main()` in '`PluginMain.cpp`'). REALbasic will call `main()` with a single 4 byte function pointer to its 'resolver' function—this allows the plug-in to call routines provided by REALbasic—and expects no return value.

Examination of the header file `MixedMode.h` reveals how to calculate the 'magic' number 193 in the following C code snippet:

```
pluginMainProcInfo = kCStackBased
 | RESULT_SIZE(SIZE_CODE(0))
 | STACK_ROUTINE_PARAMETER(1, SIZE_CODE(4));
```

which (see '`MixedMode.h`'!) evaluates to

```
pluginMainProcInfo = 1
 | 0 << 4
 | 3 << ( (4+2) + (1-1)*4 );
/*  = 1 + 0 + 3<<6 = 193 */
```

All that remains is to combine the two architectures' plug-ins into one file using Rez, and including any plug-in specific resources if there are any.

Limitations

68k and Windows support

The official plug-in SDK is designed for CodeWarrior and it can build plug-ins for Windows and for 68k Macs. MPW will never be able to compile for Windows (at least not with the supplied tools!). Nor have I been able to get MPW to compile the SDK for 68k Macs, although this at least should be possible. In any case, REAL Software have now dropped support for 68k in the REALbasic application, so the point is moot.

[3]I believe that neither REALbasic nor the CFM actually check the usage constant, so it should be possible to omit the `-xm dropin`, and hence to use an older version of PPCLink.

Static Constructors and Destructors

Because the official SDK specifies no PEF initialization or termination routines for its plug-ins, it has difficulty ensuring that any static constructors or destructors get called at the appropriate times. In fact, it only manages to call static *con*structors, and then only for PPC plug-ins. It calls neither static constructors nor destructors for Carbon plug-ins and doesn't call static *de*structors for PPC plug-ins.

The situation with this kit is somewhat better. It uses PEF initialisation and termination routines where it can, so it manages to call both static constructors and destructors for Carbon plug-ins. Unfortunately it can only call static *con*structors for PPC plug-ins. As in the official SDK, no static destructors in PPC plug-ins get called.

These problems only affect constructors and destructors of objects stored *statically* (for instance global objects). Any other objects (e.g. objects in local variables, or objects that get created and destroyed by new and delete) are unaffected. Normal (i.e. non-object) global and local variables *do* get initialised appropriately.

Patching `ExitToShell()`

There is a solution that would ensure that static destructors do get called even in PPC plug-ins. The solution to patch `ExitToShell()`, is due to Thomas Tempelmann who has implemented it in his *Plug-in Starter* ([6]).

His solution is that when the plug-in starts up, he installs a patch on `ExitToShell()` so that when the host application (either REALbasic itself, or a compiled REALbasic application) exits, the plug-in gets a chance to do its clearing up.

Thomas also takes the sensible step of checking whether the static constructors have been called, and if not, doing it manually.

It should be possible to use both techniques in this kit in the future, however patching `ExitToShell()` does appear to have some drawbacks. It can cause problems when telling MacsBug to '`es`' after a crash in a plug-in—it seems that REALbasic itself also patches `ExitToShell()` when running applications, and so only the application exits, not the whole REALbasic IDE, which is left using a plug-in that has been destructed. As you might expect, this can cause more serious crashes.

The technique of patching `ExitToShell()` also causes some problems with Alfred Van Hoek's excellent *Plugin Plunger* ([7]) for examining REALbasic plug-ins.

Consistent behaviour

To get behaviour (with respect to static constructors and destructors) that while wrong, is at least consistent between architectures, and balanced (don't call a constructor without later calling the corresponding destructor) you should modify the makefiles so that they specify `-init none -term none` as arguments to PPCLink. It should be emphasised that the only shortcoming of this kit in this regard is that it fails to call static destructors for PPC plug-ins.

Older SDKs

With version 4.5a1, the REALbasic Plug-ins SDK became much more compatible with MPW 'out of the box'. This means that many of the tricks needed to get MPW to compile the old SDK are no longer necessary. However, if you still need to use an old SDK for some reason, the files in the '`Deprecated`' folder may be useful. They work with SDK 4a1, but will likely also be compatible with earlier versions. However, unless there is a compelling version to use an old SDK, you should update to the latest version (4.5a2 at the time of writing) and then you can safely throw away the '`Deprecated`' folder.

The '`Deprecated`' folder contains the old makefiles and scripts for use with older SDKs. You should fix the '`SDK`' alias so that it points to the official SDK folder, and ensure that '`REALbasic`' points to the REALbasic application folder.

In use, the deprecated makefiles work much the same as the newer versions: simply change MPW's working directory to the '`Deprecated`' folder and run the script BuildPlugins.

Modifications to the old SDK

When you first build the example plug-ins from the makefile '`Examples.make`' in the '`Deprecated`' folder, the kit automatically makes some small changes to the files in the old REALbasic Plug-ins SDK to make it compatible with MPW. This section will describe those changes.

The changes are made by MPW's StreamEdit, using the script file '`ModSDKScript`'. I chose to take a fairly 'non-invasive' approach, by making as few modifications as possible. Therefore, after adding a comment to the top of the modified files, the changes are only to `#include` my own header files.

These modifications will not have any effect on CodeWarrior because they are protected by `#ifdef` statements (and because the kit makes its own copy of the modified files, leaving the originals untouched). Each inserted line of code ends with a comment '`// --- mod PR`' which you can search for to see the changes.

The two modified files are placed in the '`SDK.mod`' folder together with the two unmodified files from the SDK, and this kit's own files. The files copied from the official REALbasic Plug-ins SDK are

`PluginMain.cpp` modified to `#include` the supplied header file '`ExtraMPWGubbins.h`'

`REALplugin.h` modified to `#include` the supplied header file '`ExtraMPWHeaders.h`'

`rb_plugin.h` unmodified

`RBCarbonHeaders.h` unmodified

Changes to '`REALplugin.h`'

'`REALplugin.h`' is the file that each of the user's plug-in source files will normally `#include`. The change made to it is to `#include` the header file '`ExtraMPWHeaders.h`', which in turn simply provides some standard headers from the Mac Toolbox. The official SDK uses a precompiled header in CodeWarrior for the same effect, but I chose not to use that technique here for reasons of transparency.

Changes to '`PluginMain.cpp`'

'`PluginMain.cpp`' is the file that should contain the `main()` routine for every REALbasic plug-in. Although the changes to this file are accomplished simply by `#include`ing '`ExtraMPWGubbins.h`', this is where the significant compatibility issues arise.

If you attempt to build the unmodified SDK in MPW, one of the first stumbling blacks you will find is that '`PluginMain.cpp`' `#include`s the CodeWarrior header files '`A4Stuff.h`' and '`SetupA4.h`'. However upon examination, these files do not actually have any effect for non 68k builds. For instance, they provide `EnterCodeResource()` and `ExitCodeResource()` but for PPC builds these routines do nothing at all!

Since this kit makes no attempt at building 68k plug-ins (see the section 'Limitations'), it gets round that problem by providing its own files '`A4Stuff.h`' and '`SetupA4.h`', but leaving them intentionally blank (other than a comment to that effect).

All of the interesting changes happen in the file '`ExtraMPWGubbins.h`'. First, it checks that it is being used in a PPC or Carbon build and flags up an error otherwise. Then, it uses the following code

```
#define EnterCodeResource()
#define ExitCodeResource()
```

to `#define` the routines `EnterCodeResource()` and `ExitCodeResource()` as nothing, but so that '`PluginMain.cpp`' can still call them.

Static constructors

The SDK file '`PluginMain.cpp`' calls CodeWarrior's `__sinit()` routine from `main()`. This is supposed to call any static (e.g. global) constructors that might be present, and it is provided by CodeWarrior's linker. Since we're not using CodeWarrior, we could use MPW's equivalent, which is `__init_lib()`. However, it is much better to make this the PEF initialization routine, which happens automatically as long as we link to the correct library (and don't override it!).

Even so, the SDK ('`PluginMain.cpp`') still calls `__sinit()` for PPC builds, so we must provide a 'stub' for it to call.

Although initialisation and termination routines work fine for Carbon builds, there are difficulties with termination routines for PPC plug-ins. See the section 'Limitations' for more information.

Distribution

This RB-MPW Plug-ins Kit is freeware, so it may be used without charge, however it remains copyright ©2002-3 by Peter Robinson. Please let me know if you find it useful.

I would also like to hear any bug reports, comments and suggestions, at `<peter@pmpfr.co.uk>`.

Version History

The current version at the time of writing is 1.1b3.

version 1.0b1 (4th May 2002) Initial release.

version 1.0b2 (5th May 2002)

- Slightly changed ‘`BuildPlugins`’, to stop it compiling ‘`PluginMain.cpp`’ repeatedly for every plug-in. This should reduce the full build time almost by half.
- Tidied up and fixed the list of libraries to link against. I had stupidly commented out the library providing the PEF init/term routines in Carbon builds.

version 1.1b1 (11 July 2002) Updated for SDK 4.5a1.

- Now include makefiles for the new 4.5a1 SDK which is much more compatible with MPW ‘out of the box’. Retained old makefiles in ‘`Deprecated`’ folder.
- Now uses precompiled headers, controlled by the makefiles, to get around an outstanding bug in the SDK.
- No longer necessary to modify the official SDK to compile, and no extra MPW headers are needed.
- Added a new project for the BoxControl and Test Plug-in examples in the SDK.
- Changed the makefiles to work from inside the SDK folder.
- Removed the PDF documentation from the main distribution to get its size down.
- Now uses a marker file to remember whether the dependency information has been generated yet.

version 1.1b2 (3 September 2002)

- Updated contact details.

version 1.1b3 (15 January 2003)

- Checked compatibility with SDK 5a1.
- Updated documentation describing the new plug-in file format. This kit still uses the old format.
- Tweaked the makefiles to generate slightly smaller files.

References

[1] This *RB-MPW Plug-ins Kit* can be found at `<www.pmpfr.co.uk>`.

[2] My own *CompfacePlugin*, built with MPW, `<www.pmpfr.co.uk/x-faces.html>`.

[3] *REALbasic*, `<www.realbasic.com>`

[4] The official *REALbasic Plug-ins SDK* is from `<www.realbasic.com/realbasic/about/plugins.html>`

[5] Rok’s memo on making PowerePC REALbasic plug-ins using MPW, `<homepage.mac.com/rok/TechMemo/RBPluginAndMPW.html>`

[6] Thomas Tempelmann’s own *Plugin Starter*, `<www.tempel.org/rb/index.html>`

[7] Alfred Van Hoek’s *Plugin Plunger* application, `<homepage.mac.com/vanhoek>`

[8] Apple’s *MacOS Runtime Architectures*, `<developer.apple.com/techpubs/mac/runtimehtml/RTArch-2.html>`