Chapter6
Debugging a plug-in

6.1 Debugging standalone code

Plug-ins are “standalone code”: an independent code resource that is not linked into an application. The process of debugging a plug-in is therefore somewhat different than the normal process of debugging applications. The Macintosh uses many other types of standalone code resources, such as MDEFs, WDEFs, CDEFs, cdevs and INITs. Debugging a plug-in is similar to debugging these other types of standalone code resources, although Arrange provides some special features to assist you in debugging your plug-in.

6.1.1 Debugging using Macsbug

Macsbug debugging commands should work normally for plug-in modules. You can disassemble routines, set breakpoints, etc. Depending on what debugging features are enabled in the copy of Arrange you are using, you may see a large number of routines from the Arrange application itself show up in the Command-: procedure list.

When debugging a plug-in using Macsbug, it is probably a good idea to set the mfKeepLocked flag in the ‘MDdf’ resource, so that your code doesn’t move around after being loaded.

6.1.2 Debugging using Jasik’s Debugger

It is possible to do full source-level debugging for plug-ins using Steve Jasik’s Debugger. To do this, you should generate a .sym file for your plugin (e.g. when building under MPW, specify “-sym on” on the compiler and linker command lines), and then follow these steps:

1.Open the “ROM.dsi” file in the Debugger folder and make the following changes:

- Set the “Dbg_Rsrcs” flag to 1. This allows debugging of standalone code resources. You may want to turn the flag back off when you are not debugging plug-ins

- Add the two lines “=G’ and “MDcd” to the file. This tells the Debugger that “MDcd” is a resource type which contains executable code.

2. Reboot to allow the new ROM.dsi settings to take effect. (You only need to perform steps 1 and 2 the first time you debug a plug-in.)

3. Rename your .sym file to “Select Subnotes/MDcd_8000.sym”, where “Select Subnotes” is replaced by the filename of your plugin, and “8000” is the resource ID of your MDcd resource, as four unsigned hex digits. Place the .sym file in the same folder as your plug-in.

When Arrange starts up and loads your plug-in, the Debugger reads your .sym file and adds an entry to the task list for the plug-in. You can then use all of the normal debugging commands. You may wish to create a .dsi file for your plugin (named similarly to the .sym file) that specifies the location of your source code.

6.2 Special debugging features in Arrange

You might want to use a special version of the Arrange application which includes extra support for debugging plug-in modules. This extra support includes error checking in callback functions, and a special “debugging window” in which you can print diagnostic information and examine the internal state of your Arrange document.

For information on receiving the Arrange plug-in debug version of Arrange, contact Developer relations at Common Knowledge.

6.2.1 Callback error checking

All versions of Arrange, including the standard shipping version, do error checking on all parameters passed to a callback function. Some of these checks are mentioned in the API documentation using statements of the form “under such-and-such a condition, log an error and return nil” (or something similar). In the shipping version of Arrange, when the described condition occurs, the function will simply return nil; there is no mechanism for “logging” an error. In a debugging version, Arrange will drop into the debugger with an error message along the lines of “Error in callback GetSelEntry, called from Auto-completion: index out of range”. The message includes the name of the callback function which detected the error (“GetSelEntry”), the name of the plugin module which called the callback function (“Auto-completion”), and a message specific to this error (“index out of range”). In this example, the index parameter was out of range, i.e. less than zero or greater-than-or-equal-to the number of objects in the selection.

If you continue execution from one of these “Error in callback” messages, the program will proceed as it would in a non-debugging version, i.e. it will return control to the offending plug-in with a default result (typically nil).

6.3 The Arrange debugging window

Debugging versions of Arrange have an extra menu at the end of the menu bar; the name of this menu is a space character, so it is normally invisible. To select items in this menu, click just to the right of the last visible menu in the main menu bar. Select the “Show Debug Window” command to bring up the debugging window. (Depending on which debugging version of Arrange you have, there may be other commands in this menu as well. Exploration is strongly discouraged; these other commands are designed for internal use at CKI and are likely to crash your machine or corrupt your document.)

6.3.1 Logging to the debug window

The PrintToLog callback function, defined in Module.h, appends a string to the end of the debug window. You can use this function to print a record of actions taken by your plug-in, or to output other internal information for debugging purposes. In non-debugging versions of Arrange, PrintToLog does nothing (however, it is a bad practice to leave debugging code in a shipping copy of your plug-in).

The debugging window is written using TextEdit, and therefore is not designed to handle large amounts of text. If the window contains more than 32,000 characters of text, text will be discarded at the top to make room for new information at the bottom. Performance will drop noticably long before this limit is reached; you may want to periodically clear the contents of the debugging window using the “Clear Debug Window” command in the debugging menu, or by triple-clicking on the text in the window and hitting backspace.

Logging messages to the debugging window is particularly handy for understanding when and how Arrange calls your hook functions. If you printa message each time your hook function is called, then you can watch the messages go by in the debugging window as you perfom actions in your Arrange document.

6.3.2 Debug window commands

In addition to displaying information in the debug window using the PrintToLog function, you can also use the debug window to enter debugging commands for Arrange itself. Commands are entered in a manner similar to the MPW worksheet: you type a command on a single line and press Enter, or select some text and press Enter to execute the selected text.

The debug window supports a number of commands, many of which are for CKI internal use only. One command of general interest is “PrintCurState”. This will output a block of text looking something like this:

```
Current window = $1b2406c, document = $197de58:
```

```
windowSettings = i1047

curTopic       = i1046

curRootTopic   = i1045

selection      = vfe00034
```

The first line gives the memory address of certain internal data structures and should be ignored. The next three lines give the arNoteID for the frontmost document window, the current topic or view in that window, and the current "root topic" in that window. (If the window is displaying a topic, then the "root topic" is that topic; if the window is displaying a view, then the root topic is the view's parent topic.) The last line gives the ID of a data structure listing the contents of the selection. In some cases, the third line may look like this: "`selTriple = (i1056,i0,i1010)`". In this case, the three numbers are the parentNote, field, and note values (respectively) which would be returned by the GetSelEntry callback function. See the documentation for GetSelEntry for more details.

To display the contents of a particular note, execute a command of the form "iNNN", where NNN is the decimal value of the note's arNoteID. For example, if you double-click on the ID of the current window ("i1047" in the above example) and hit enter, you will get output which looks something like this:

```
LocalItemID 1047:

{ "selArray2"->(array v0FCC0034),

."createDate"->2/18/94,

."modDate"->2/18/94,

."type"->i1007,

."subItems"->(array v0FE00034),

."fieldOrder"->(array v0FA40068),

."creatorName"->"Steve",

."editorName"->"Steve",

."bodyText"->nil,
```

```
."bodyTextAux"->nil,

."name"->"",

."selectedPlace"->i1046,

."placesScroll"->nil,

."windowSize"->40108506,

."windowLoc"->3015207,

."shelfList"->(array v0FBC006C),

."selectionArray"->nil,

."shelfHeight"->100,

."placeListWidth"->-120,

."shelfScroll"->nil,

."displayCat"->i1024,

."visMode"->1,

."iconBar"->true,

."tlState"->0,

."untimedViewHeight"->80}
```

A full explanation of this output is beyond the scope of this document; much of the information displayed is strictly for internal use by Arrange. However, scattered about will be the all of the note's fields and other information. For example, this note was created by "Steve" on "2/18/94", and its list of subnotes is "`array v0FE00034`". Data structures such as arrays and sets are represented by a hexadecimal value prefixed by the letter v; you can double-click on one of these "v values" and type enter to display the contents of the array.

Don't get too carried away looking at internal data structures in the debugging window. These facilities are mainly intended for internal use at CKI. However, it can sometimes be handy to

peek at the raw data when you are having a confusing problem with your plug-in.