Programming with Haiku

Lesson 18

Written by DarkWyrm

Application Scripting

One of the most complicated – and least understood – parts of the Haiku operating system is its system for scripting applications. This is mostly because it looks like a complicated mess to the uninitiated, and compared to the C++ API, it is. Unfortunately, this is the price paid for the incredible flexibility it provides. The official documentation on the subject in the Be Book doesn't exactly make it any easier, either. By the end of this lesson, you should have a good grasp of the way that application scripting is done with Haiku.

Application scripting abstracts away all of the C++ details of the API and provides a way to interact with an application using nothing but messages to manipulate objects and properties. While any message can be technically sent to an application to script it, here is a list of the official scripting message constants which can be used to describe most of the interface.

Command	Description
`B_COUNT_PROPERTIES`	Get the number of instances of a property.
`B_CREATE_PROPERTY`	Create a new instance of a property. Note that this can't add new properties to a suite – it can only create new instances of an existing property.
`B_DELETE_PROPERTY`	Delete an instance of a property.
`B_EXECUTE_PROPERTY`	Run a property like a method.
`B_GET_PROPERTY`	Get the value of a property.
`B_SET_PROPERTY`	Set the value of a property.

Scripting Suites

These commands don't help us very much without one other key piece of the puzzle: getting the list of properties for a particular object, i.e. the list of scripting interfaces an object supports. The BHandler class is the foundation of Haiku's scripting support. Each scriptable object inherits from the BHandler class, so every one of them has the properties `Suites`, `Messenger`, and `InternalName`. This interface, known in the lingo as a **suite**, has the name `"vnd.Be-handler"`, which is not a MIME type even though it bears a striking resemblance to one.

The `Suites` property is of particular interest to us. If we ask an object to `GET` it, it will reply with a message which contains a list of the suites and properties that it supports. We can test this out using the Terminal command `hey`, which comes bundled with Haiku. Run this command from the Terminal:

```
hey Tracker getsuites
```

This certainly looks simple enough. `hey` hides some of the complexity for us. The output from the above command reminds us of the beast with which we are dealing.

```
Reply BMessage(B_REPLY):
   "suites" (B_STRING_TYPE) : "suite/x-vnd.Be-TRAK"
   "suites" (B_STRING_TYPE) : "suite/vnd.Be-application"
   "suites" (B_STRING_TYPE) : "suite/vnd.Be-looper"
```

```
   "suites" (B_STRING_TYPE) : "suite/vnd.Be-handler"
   "messages" (B_PROPERTY_INFO_TYPE) :
        property   commands                                specifiers             types
-------------------------------------------------------------------------------
           Trash   B_DELETE_PROPERTY                       DIRECT
                   Usage: delete Trash # Empties the Trash
          Folder   B_CREATE_PROPERTY                       DIRECT                 RREF
                   Usage: create Folder to path # creates a new folder
     Preferences   B_EXECUTE_PROPERTY                      DIRECT
                   Usage: shows Tracker preferences

   "messages" (B_PROPERTY_INFO_TYPE) :
        property   commands                                specifiers             types
-------------------------------------------------------------------------------
          Window                                           INDEX REV.INDEX
                   Usage:
          Window                                           NAME
                   Usage:
          Looper                                           INDEX REV.INDEX
                   Usage:
          Looper                                           ID
                   Usage:
          Looper                                           NAME
                   Usage:
            Name   B_GET_PROPERTY                          DIRECT                 CSTR
                   Usage:
          Window   B_COUNT_PROPERTIES                      DIRECT                 LONG
                   Usage:
         Loopers   B_GET_PROPERTY                          DIRECT                 MSNG
                   Usage:
         Windows   B_GET_PROPERTY                          DIRECT                 MSNG
                   Usage:
          Looper   B_COUNT_PROPERTIES                      DIRECT                 LONG
                   Usage:

   "messages" (B_PROPERTY_INFO_TYPE) :
        property   commands                                specifiers             types
-------------------------------------------------------------------------------
         Handler                                           INDEX REV.INDEX
                   Usage:
        Handlers   B_GET_PROPERTY                          DIRECT                 MSNG
                   Usage:
         Handler   B_COUNT_PROPERTIES                      DIRECT                 LONG
                   Usage:

   "messages" (B_PROPERTY_INFO_TYPE) :
        property   commands                                specifiers             types
-------------------------------------------------------------------------------
          Suites   B_GET_PROPERTY                          DIRECT                 (suites CSTR)
(messages SCTD)
                   Usage:
       Messenger   B_GET_PROPERTY                          DIRECT                 MSNG
                   Usage:
    InternalName   B_GET_PROPERTY                          DIRECT                 CSTR
                   Usage:

   "error" (B_INT32_TYPE) : 0 (0x00000000)
```

What a mess! There is a way to make sense of it, though. We are looking at a dump of the different suites that the main part of Tracker provides. There are four of them:

```
"suite/x-vnd.Be-TRAK"
"suite/vnd.Be-application"
"suite/vnd.Be-looper"
"suite/vnd.Be-handler"
```

The rest of the information printed is a dump of each suite in the same order as the list at the top. Instead of trying to figure all four of them out at the same time, let's just look at the last one and pick it apart.

A better way to describe this information would be like this:

Property Name	Command	Specifier	Return Type
Suites	Get	Direct	String array, BPropertyInfo array
Messenger	Get	Direct	BMessenger
InternalName	Get	Direct	String

Specifiers

The word `Direct` used in the third column of the table is the specifier for a scripting command. Specifiers are the way that properties are referenced. Here is a list of the available specifiers:

Specifier	Description
`B_DIRECT_SPECIFIER`	Used by properties where no additional information is needed to reference the property. This is the specifier used by non-array properties, like `Name` or `Frame`.
`B_NAME_SPECIFIER`	A string is used to determine which property instance is to be used.
`B_ID_SPECIFIER`	A unique 32-bit integer is used to reference a particular property instance.
`B_INDEX_SPECIFIER`	A 32-bit integer specifies a property instance the same way that you would an index for an array.
`B_REVERSE_INDEX_SPECIFIER`	This works the same way, but counts from the end of the array. A reverse index of -1 is the last element in the list.
`B_RANGE_SPECIFIER`	One or more consecutive property instances can be referenced at one time.
`B_REVERSE_RANGE_SPECIFIER`	This works starting from the end of the array instead of the beginning. More on both range specifiers can be found below in the section on using the scripting API from C++.

Most – but not all – properties can be referenced using only one kind of specifier. When the `hey` command prints out all of the properties in a suite, it lists each property by specifier type. The `"vnd.Be-application"` suite used by BApplication objects could be described using our simplified form like this:

Property Name	Command	Specifier	Return Type
Window	Any*	Index, Reverse Index	Object
Window	Any*	Name	Object

Property Name	Command	Specifier	Return Type
Window	Count	Direct	int32
Looper		Index, Reverse Index	Object
Looper		Name	Object
Looper		ID	Object
Looper	Count	Direct	int32
Name	Get	Direct	String
Windows	Get	Direct	BMessage
Loopers	Get	Direct	BMessage

A window, for example, can be referenced by index, reverse-order index, or by name. Getting the window itself isn't very useful, but accessing a window's properties is. Here are a few of the more useful properties in the `"vnd.Be-window"` window suite:

Property Name	Command	Specifier	Return Type
Title	Get, Set	Direct	String
Frame	Get, Set	Direct	BRect
MenuBar	Any	Direct	Object
View	Count	Direct	Int32
View	Any*	Name, Index, Reverse Index	Object

Let's play around a bit with StyledEdit using these Terminal commands:

```
open /system/apps/StyledEdit
hey StyledEdit set Title of Window "Untitled 1" to "Haiku Rocks"
hey StyledEdit set Frame of Window 1 to "BRect(100,100,500,400)"
hey StyledEdit get Title of Window -1
```

First, we open StyledEdit, then we change the title of the first blank document window to "Haiku Rocks" and both move and resize the window. StyledEdit's Window 0 happens to be the Open File dialog window, which is not really what we want. Lastly, we obtain the title of the last window.

Any time a property returns an object, it means that an additional specifier must be used to get at the subobject's properties. We can home in on a Window's views or menus this way.

```
hey StyledEdit get InternalName of View -1 of View 1 of Window -1
```

The name returned is `textview`. Very interesting! We just homed in on the BTextView used to hold the text in a document window! Perhaps the most amazing part about it all is that we didn't use one bit of C or C++ to do it. `hey` provides an interface to the scripting API which can be used easily from any language.

Concluding Thoughts

The scripting API that Haiku provides is very deep and powerful, but far underutilized or understood. With the ground we have covered here, you should be starting to get an idea of its potential. Any Haiku application can be "remotely controlled" in a limited way without any extra effort from the developer. An obscure program from the days of BeOS called DogWhistle used scripting with Tracker to create a unique file management application, for example. With a little work, your programs can be leveraged by others to do something which was previously not possible.

Going Further

- Fire up the accompanying demo application *Scripting Explorer* and use it to tinker around with some of the programs bundled with Haiku. See what interfaces some of the programs provide. How could they be used by others?
- What programs bundled with Haiku are currently of limited scripting use and could be expanded to be much more powerful? For what could some of them be used?
- For a real thinking challenge, what might go into a library designed to program the API purely from the scripting API? How could it be implemented?