Einsteinium Engine Subscriber's Development Guide

A Guide for Using the Einsteinium Engine

http://esforhaiku.sourceforge.net/

Introduction

This guide is for programmers who wish to develop an application for Haiku that uses the Einsteinium Engine. The Engine can provide ranked lists of applications which continually update based upon when applications launch and quit. This function greatly enhances the ability to provide more than just the standard Haiku “Recent Applications” list.

What Is the Einsteinium Engine?

The Engine is a background application that monitors all applications for when they launch and quit, and records these times into a database. The Engine is primarily called upon by other applications to provide a ranked list of applications given certain parameters.

Using the data collected on application launch and quit times, the Engine is able to create ordered lists of applications based on weighing certain statistics over others. For example, the Engine can determine which applications have been launched the most and the least number of times, which have the largest and the smallest total lifetime running time, which have the largest and smallest time intervals between the last two launches, etc.

Haiku has a standard “Recent applications” list of the most recently launched applications. With the Engine, you can create similar lists but using one or a combination of any of the following criteria:

- Total number of times the application has been launched
- The date and time of the very first launch
- The date and time of the most recent launch
- The time that elapsed between the two most recent launches
- The total running time of the application over its lifetime

How does the Engine collect the data? The Engine runs in the background and watches the be_roster, and receives a message whenever an app is launched or quit. Each app has an associated file located in ~/config/settings/Einsteinium/Applications which is a simple sqlite database. The launch and quit times stored in each application's database are analyzed by the Engine to produce various statistics on each application's usage.

How Can I Use The Engine?

In order to query the Engine for a ranked application list, a third party application must become a "subscriber" to the Engine. As a subscriber, the application will tell the Engine what values to use when calculating the ranked list, and the Engine will automatically push updates to the subscriber as the ranked list changes. The subscribing application may tell the Engine to change the values used to calculate the ranked list at any time.

All interaction with the Engine is implemented through a class called the "EngineSubscriber". This class is available from the Einsteinium svn repository (in /trunk/src/Engine/). The subscribing application must create a class that implements the EngineSubscriber class. For example, a simple application can inherit the EngineSubscriber class:

```
class SubscriberApp : public BApplication, EngineSubscriber {
```

There are 3 functions within the the EngineSubscriber class that are virtual function and MUST be implemented by your inheriting class. These functions are:

```
// virtual functions inherited from the EngineSubscriber class
virtual void            _SubscribeFailed();
virtual void            _SubscribeConfirmed();
virtual void            _UpdateReceived(BMessage *message);
```

These functions are called when a message is received from the Engine. The first two functions are called after a subscription message is sent to the Engine. These inform your application if the subscription failed or was confirmed (succeeded), and only one or the other will be called as a result of your subscription. The _UpdateReceived function is called whenever the Engine sends your application a new updated ranked list. The details of these functions will be described later, but for now if you are creating an inheriting class these functions must be defined even if they are left empty.

The first thing to do is to check to make sure the Engine is running, since the Engine must be running in order to subscribe and receive messages from it. To check this use the _IsEngineRunning() function, which returns a boolean. If the Engine is not running, call the _LaunchEngine() function to launch it. For example:

```
// Launch the engine if it is not running
if(!_IsEngineRunning())
        _LaunchEngine();
```

Now that the Engine is running, we can subscribe to it by defining the number of applications we want in the list, each weighted scale value, and applications to exclude from the list. First we want to reset all values to their defaults:

```
_ResetSubscriberValues();
```

Next we define the maximum number of applications in the list we want return by the Engine:

```
_SetCount(COUNT_VALUE);
```

Next define the values to use for each weighted scale:

```
_SetTotalLaunchesScale(LAUNCH_VALUE);
_SetFirstLaunchScale(FIRST_VALUE);
_SetLastLaunchScale(LAST_VALUE);
_SetLastIntervalScale(INTERVAL_VALUE);
_SetTotalRuntimeScale(RUNTIME_VALUE);
```

If there are any applications you would like excluded from the list, you can do so using the application's signature like so:

```
_AddExclusion("application/x-vnd.Be-TSKB");
```

And finally, call this function to send the subscription message to the Engine:

```
_SubscribeToEngine();
```

At this point you should quickly receive a reply from the Engine, and either the _SubscribeFailed() or _SubscribeConfirmed() function will be called. You can program these functions to take any action you need to depending on the confirmation or failure of the subscription. Note that the _SubscribeConfirmed function does not provide you any information about your ranked list, it is ONLY a confirmation indicator. In this function you can perform and steps necessary to prepare for your first ranked list to arrive, but you do not have this information yet.

After returning from your _SubscribeConfirmed function, the _UpdateReceived(BMessage *message) function will be called with information on your ranked list. The message passed to your function contains a list of entry_ref objects under the name “refs”, with the highest ranking application being the first entry_ref object, and each successively lower ranked object being the next entry_ref. You can extract each entry_ref from the message with something like the following:

```
// Get any refs found
int32 fSubscriptionRefCount = 0;
if(message)
{
	type_code typeFound;
	message->GetInfo("refs", &typeFound, &fSubscriptionRefCount);
	entry_ref newref;
	for(int i=0; i<fSubscriptionRefCount; i++)
	{
		message->FindRef("refs", i, &newref);
		// Do what you need to with your newref object here
	}
}
```

The Engine will continue to monitor all application launches and quits, and if your ranked application list changes at all, the Engine will push that updated list to your application which will call the _UpdateReceived(BMessage *message) function with the new list. Therefore your application must be ready for the _UpdateReceived function to be called at any time with an update from the Engine.

If your program must update any of the values used to create the ranked list, simply set the values like

you did with your first subscription, and call the _SubscribeToEngine() function. The Engine will recognize that you have a current subscription and will update the values used for the rank list. Usually this results in the Engine pushing a new update to your application immediately because the new list will have a different application rank order.

If your application will quit or it does not need to receive updates any longer, call the _UnsubscribeFromEngine() function and the Engine will stop sending updates.

Here is a very simple example of using the EngineSubscriber class to print out a list of the most recently used applications. This example can be found included with the Einsteinium source files. Run the application from the Terminal with the command 'subscriber_example &' then open a few applications to see the updated rank list print out. The run the command 'subscriber_example -q' to unsubscribe and quit the application.

subscriber_example source:

```
/* SubscriberExample.cpp
 * Copyright 2012 Brian Hill
 * All rights reserved. Distributed under the terms of the BSD License.
 */
#include <AppKit.h>
#include "EngineSubscriber.h"

class SubscriberApp : public BApplication, EngineSubscriber {
public:
                                SubscriberApp();
                                ~SubscriberApp();
        virtual void            ArgvReceived(int32, char**);
        virtual void            ReadyToRun();
private:
        // virtual functions inherited from the EngineSubscriber class
        virtual void            _SubscribeFailed();
        virtual void            _SubscribeConfirmed();
        virtual void            _UpdateReceived(BMessage *message);
};

SubscriberApp::SubscriberApp()
: BApplication("application/x-vnd.Einsteinium_Subscriber_Example")
{ }

SubscriberApp::~SubscriberApp()
{
        printf("Unsubscribing from Engine...\n");
        _UnsubscribeFromEngine();
        printf("Quitting Einsteinium Subscriber Example.\n");
}

void SubscriberApp::ArgvReceived(int32 argc, char** argv)
{
        if(strcmp(argv[1],"-q")==0 || strcmp(argv[1],"--quit")==0)//option to quit
                PostMessage(B_QUIT_REQUESTED);
}

void SubscriberApp::ReadyToRun()
{
```

```
	printf("Example of the Einsteinium Engine Subscriber\n");
	// Launch the Engine if it is not running
	if(!_IsEngineRunning())
		_LaunchEngine();

	// Subscribe to the Einsteinium Engine- simulate the Recent Applications list
	printf("Subscribing to Engine...\n");
	_ResetSubscriberValues();
	_SetCount(20);
	_SetTotalLaunchesScale(0);
	_SetFirstLaunchScale(0);
	_SetLastLaunchScale(1);
	_SetLastIntervalScale(0);
	_SetTotalRuntimeScale(0);
	_AddExclusion("application/x-vnd.Be-TSKB");
	_SubscribeToEngine();
}

void SubscriberApp::_SubscribeFailed()
{
	printf("Subscribe Failed.\n");
}

void SubscriberApp::_SubscribeConfirmed()
{
	printf("Subscribe Confirmed.\n");
}

void SubscriberApp::_UpdateReceived(BMessage *message)
{
	// Print the updated applications list
	int32 fSubscriptionRefCount = 0;
	if(message)
	{
		type_code typeFound;
		message->GetInfo("refs", &typeFound, &fSubscriptionRefCount);
		printf("Updated application list:\n");
		entry_ref newref;
		for(int i=0; i<fSubscriptionRefCount; i++)
		{
			message->FindRef("refs", i, &newref);
			printf("%i- %s\n", i+1, newref.name);
		}
	}
}

int main()
{
	SubscriberApp myApp;
	myApp.Run();
	return 0;
}
```