Win32Utils Support
Daniel Berger

== Win32Utils Overview

The Win32Utils Project provides a series of Ruby libraries that are designed specifically for the Windows operating system to make life easier and more enjoyable for Ruby programmers on MS Windows. These libraries provide interfaces to various parts of the Windows OS, and extend the Ruby core classes so that they're better suited for Windows.

The goal of this project was to continue the transition of several of the existing libraries from Win32API library to FFI or OLE in order to make the code more robust and compatible with other implementations, such as JRuby.

== win32-file-stat

gem: win32-file-stat
url: https://github.com/djberg96/win32-file-stat

The win32-file-stat library has been fully converted to FFI, with several new methods added from the previous version. This includes analogues for the #ino, #gid, #grpowned, #owned?, #rdev and #uid methods. All tests pass with both MRI and JRuby. The gem was released on December 16th, 2013.

Issues: In the process of testing this with JRuby, two bugs were discovered. First, JRuby returns incorrect values for the address of unsigned integers. This mainly affected the INVALID_HANDLE_VALUE constant, which resulted in false positives for certain function. Second, JRuby deviated from MRI by returning numeric values instead of nil for the File::Stat members #dev_major, #dev_minor, #rdev_major and #rdev_minor. Both issues have been reported.

== win32-file

gem: win32-file

url: https://github.com/djberg96/win32-file

The win32-file library has been fully converted to FFI. Most of the custom methods were removed, having been moved into separate gems like win32-file-attributes or win32-file-security. The File.long_path and File.short_path methods remain, however, as do the overridden methods. Several new methods were overridden in this release, owing to the changes in win32-file-stat. These include the .group_owned?, .readable?, .owned? and .writable? singleton methods. In addition the .realpath and .realdirpath methods have been overridden to handle symlinks. The gem was released December 16th, 2013.

Issues: I had to handle in-out string arguments differently than I had originally planned. Internally certain Windows functions modify the input argument by adding a null byte at a specific index. However, JRuby doesn't work that way. Instead, I used string pointers (instead of a string buffer) and then read bytes from the pointer as needed. This worked for both MRI and JRuby. It's a safer approach for other possible implementations as well.

I did find one bug on JRuby. Specifically, it was unable to delete a dangling symlink on Windows, raising an Errno::ENOENT instead. This issue has been reported.

== win32-eventlog

gem: win32-eventlog
url: https://github.com/djberg96/win32-eventlog

The win32-eventlog library has been fully converted to FFI with the original API intact. One bug (a potentially undersized buffer) was corrected along the way. The gem was released on January 13th, 2014.

Issues: My original plans to use OLE and/or a mix of FFI and OLE were dropped because it's not possible to create an event source, nor write to it, using the OLE/WMI interface that Microsoft provides. Consequently, I decided it was easier to simply use FFI throughout the entire codebase.

Another issue that came up was JRuby's heap size. JRuby users will probably need to

increase the heap size to use the win32-eventlog library properly.

== win32-clipboard

gem: win32-clipboard
url: https://github.com/djberg96/win32-clipboard

The win32-clipboard library has been fully converted to FFI. The original API has been kept intact, with a couple enhancements along the way. First, the ability to write bitmap data to the clipboard has been added thanks to a patch from Tadashi Kba. Second, the ability to copy and retrieve raw html data was added by Park Heesob's HtmlClipboard class. Two methods that wrap HtmlClipboard, get_html_data and set_html_data, were conseqently added to the core Clipboard class. The win32-clipboard gem was released on January 12th, 2014.

In other news, I decided to drop support for Ruby 1.8. I also dropped support for Windows XP since Microsoft will end its support in April 2014.

Issues: Shortly after the first release I realized there was a bug with certain functions that were only exported on 64-bit platforms. I consequently updated the code to work for both 32-bit and 64-bit platforms, and pushed out an updated release on January 20th, 2014. There still were some issues with change notifications and another release was pushed out on 7-Feb-2014.

== win32-taskscheduler

gem: win32-taskscheduler
url: https://github.com/djberg96/win32-taskscheduler

The win32-taskscheduler library has been fully converted to use OLE. The original APi was mostly kept intact, though additional arguments were added to the constructor. These arguments allow the user to specify a folder to look in, and whether or not to create the folder if it doesn't exist. By default the root folder is used (where tasks are kept on the system) and new folders are not created.

Issues: Certain methods are no longer necessary, such as TaskScheduler#save. Rather

than remove them I have made them deprecated methods for now. They will be removed in the next major release. In some cases, changes made to an existing task are not applied immediately, and the user must re-activate the task to ensure the changes are applied. On JRuby I did notice that it raised a native exception in one place instead of a Ruby exception, but this should not have any real affect in practice.

== win32-changejournal

gem: win32-dirmonitor
url: https://github.com/djberg96/win32-dirmonitor

I decided that porting the win32-changejournal library was both unnecessary and too difficult. Instead, I created a new library called win32-dirmonitor that accomplishes the same functionality, but uses OLE instead of FFI. The API is very similar, though the struct used to hold file changes did have some member changes, so new users will need to update their code.

Issues: The win32-dirmonitor library does not currently monitor files in subdirectories, only the files in the specified directory. I plan to allow this option if possible.

== win32-job

gem: win32-job
url: https://github.com/djberg96/win32-job

This is a new library written using FFI. It provides an API for "jobs" on Windows, which is how Windows manages process groups. Although there is some minor duplication of functionality with win32-process, this library provides much more granular configuration of process groups if the user so chooses. It also provides the ability to gather account or limit information on jobs, kill jobs, and wait for all processes in a job to complete.

Issues: No serious issues. Some enhancements and changes were made after the initial release. The current release is 0.1.2, released on 7-Feb-2014.

== win32-api

gem: win32-api
url: https://github.com/djberg96/win32-api

One potential initialization bug was patched courtesy of Kevin Huene and version 1.5.1 was pushed on 14-Feb-2014.

Although deprecated there are some people (notably Puppet) that still rely on it, so I continue to support it for now.

BONUS ACCOMPLISHMENTS

== file-find

gem: file-find
url: https://github.com/djberg96/file-find

As a side effect of converting the sys-admin gem to use FFI, I was able to add support for the :uid, :gid and :inum options for the file-find gem. I also fixed support (however limited) for the :perm option on Windows. Version 0.3.8 of the file-find gem was released on 12-Feb-2014.

== file-temp

gem: file-temp
url: https://github.com/djberg96/file-temp

For the file-temp gem I cleaned up and simplified the source code for both MS Windows and Unix systems, and included separate source code for JRuby. Version 1.2.1 of the file-temp gem was released on 17-Feb-2014.

CONCLUSION

All goals were met, and some additional work above and beyond was completed. The project was a success.

== Lessons Learned

Although I'm an old hand at FFI by this point, there were some lessons learned and some experiences that I can pass along.

=== Compatibility with JRuby

Although using FFI usually means compatibility with JRuby, there were a few snags that had to be overcome.

One issue that came up was casting negative values. For example, the constant INVALID_HANDLE_VALUE is defined as (DWORD)-1 in the windows.h header file. This translates to 0xFFFFFFFF on 32-bit platforms and 0xFFFFFFFFFFFFFFFF on 64-bit platforms. With that in mind we originally defined it like so:

INVALID_HANDLE_VALUE = FFI::Pointer.new(-1).address

However, it turns out that JRuby does not cast negative integers in a way we would expect, as per https://github.com/jruby/jruby/issues/1315. So, we ended up using this approach instead:

INVALID_HANDLE_VALUE = (1<<FFI::Platform::ADDRESS_SIZE)-1

This returns the expected value for both MRI and JRuby.

Another issue that came up was string handling. There are a couple different ways you can deal with strings and function prototypes in FFI. One is to use string buffers, e.g. buffer_in or buffer_out. Another is to use pointers. I found that it was generally better to use pointers because by using pointers you could read a specific length from the FFI pointer, as needed, and often the exact length you needed was known. With string buffers, we often found ourselves manually parsing out null characters, which was susceptible to errors.

== 32-bit vs 64-bit Issues

In one instance we needed to specifically define different functions depending on

whether or not we were using a 32-bit or 64-bit version of Ruby. This came up with the GetWindowLong and GetWindowLongPtr functions for the win32-clipboard gem. We handled it by assuming a 32-bit version, and defaulting to the 64-bit version if the 32-bit function definition failed.

== Overall Productivity and Maintainability

Despite a few quirks with JRuby, overall I found FFI to be a fantastic library for interfacing with C libraries on Windows. Although it does require some additional up-front work, such as manually defining structs and constants from header files, the payoff is worth it. The structs are effectively self-documenting and constants, once defined, are familiar to people coming from a Windows API background.

Using FFI let me leverage Ruby constructs that make coding much easier than if I were writing C code. For example, using the begin/rescue/ensure paradigm, I can make sure an open handle is closed no matter where an error occurs within that block. If I were using C code I would have to explicitly close that handle if any function in between failed. I've also found that retry works as an excellent way to reallocate memory as needed.

Lastly, I'll say that FFI makes the code highly maintainable. Not only is it easy to read, it's easy to modify. Even people who are not necessarily C programmers can easily read the code base, spot bugs, and even provide patches in simple cases. I receive far more merge requests using FFI than I ever did when I was using C code.