Using Win32Utils, Part I

by Daniel Berger

Quick introduction

My work on the Win32Utils project originally stemmed from a need to make a TCP/IP client-server application, which myself and a colleague had written for work, cross platform. It was then that I learned about Windows services. It was also then that I realized how woefully few libraries there were for Ruby on Windows.

I decided to look at what both Perl and Python had in terms of Windows libraries and began porting them, learning the Windows API as I went. I created the Win32Utils project on RubyForge, recruited both Park Heesob and Shashank Date, and starting working on getting Ruby up to speed with regards to Ruby libraries for Windows, including a library for Windows services.

There are now over a dozen packages forming the Win32Utils project. They include interfaces for native mutexes, semaphores and events, as well as eventlog and service information, user and group information and administration, process information, native file methods, the clipboard, pipes, the taskscheduler, shortcuts and the SAPI library. We'll look over a few of these packages today.

A closer look at a few libraries

WIN32-CLIPBOARD

This package provides an interface to the Windows clipboard. With this you can retrieve data copied to the clipboard, insert data into the clipboard, or just clear it.

Example:

```
require “win32/clipboard”
include Win32

# Get the data on the clipboard
puts “Data on clipboard was: “ + Clipboard.data

# Set the data on the clipboard
Clipboard.data = “test”

# Printing the data again, we get “test”
puts “Data on clipboard is now: “ + Clipboard.data

# Clear the clipboard
Clipboard.empty
```

Note that the current release does not support any non-textual formats, i.e. You can't copy and paste images with this package. Support for this will be added in a future release.

WIN32-ETC

This package allows you to retrieve information about current users and groups

on a local or remote system. You can also administer users and groups using the Admin module.

The API for win32-etc was modelled on the 'etc' package that comes as part of the standard library. The API is, in fact, nearly identical to the version that you would use on Unix systems.

Example:

```
require “win32/etc”
include Win32

# Print users on a remote system
Etc.passwd('\\some_machine'){ |pwstruct|
	p pwstruct
}

# Print groups on the system
Etc.group{ |gstruct|
	p gstruct
}
```

There are two differences between this version and the version that comes with the standard library. First, you can specify a host as an argument to most of the Etc methods to retrieve information from remote machines. Second, the members of the struct yielded to Etc.passwd and Etc.group will be different, though similar, to its Unix counterpart.

In order to add users or groups, or do any sort of adminstration in general, use the Admin module.

Example:

```
require “win32/etc”
include Win32

# Add a user
Etc::Admin.add_user(
	:user_name	=> "some_user",
	:password	=> "abc123def",
	:home_dir	=> "C:\\foo_user",
	:description	=> "random text",
	:host		=> "\\some_machine",
	:flags		=> Admin::ACCOUNTDISABLE
)

# Add a group
Etc::Admin.add_group(
	:group_name	=> "foo",
	:description	=> "Test Group",
	:local?		=> true,
	:users		=> %w/matz guido larry/
)
```

With the Admin module you can also configure and delete existing users and groups, as well as set passwords.

WIN32-FILE

This package provides additional, win32-specific methods to Ruby's builtin File class, and replaces a couple of the existing methods.

One of the features of files on Win32 systems is that they have attributes. If you were to use your Explorer window, right click on a file and select "properties", you would see the additional properties for that file. You can get and set these using Ruby by using the win32-file package.

Example:

```
require "win32/file"

# Find out information about 'foo.txt'
my_file = "C:\\Program Files\\foo.txt"
File.compressed?(my_file)       # true or false
File.read_only?(my_file)        # true or false

# Set some attributes for 'foo.txt'
File.open(my_file,"w+"){ |fh|
        fh.read_only = true
        fh.compressed = true
}
```

In addition, you can get and set the security settings of files.[1] Use File.get_permissions to retrieve a hash of key value pairs that describe the security settings. However, the values are integer values. To turn those into human readable strings, we need to pass the flag to the File.securities method.[2]

Example:

```
require "win32/file"

# Get the current security settings for 'foo.txt'
File.get_permissions("foo.txt").each{ |acct,perms|
        puts "Account: #{acct}:"
        puts File.securities(perm).join("\n\t")
}
```

It is also possible to set file permissions using the File.set_permissions method.

Other capabilities of the win32-file package include encrypting and decrypting files, native IO read and write methods, and many more attribute related methods.

On a final note, a few of the core File methods have been reimplemented in the win32-file package, either because they weren't implemented for Win32 systems, or they didn't work properly. At the time of this writing this included File.blockdev?, File.chardev? and File.size[3].

---

1 This is only possible on NTFS filesystems.
2 I may alter this in a future release to simply return an array.
3 The core File.size method works fine so long as the file size is less than 2 GB.

WIN32-SOUND

This package allows you to play sounds on Win32 systems, including .wav files and system sounds.

Example:

```
require “win32/sound”
include Win32

# Play a .wav file
Sound.play(“foo.wav”)

# Play a system sound
Sound.play(“SystemAsterisk”,Sound::ALIAS)
```

It is also possible to play a simple beep, retrieve information about sound devices on your system, check the volume or stop any currently playing sounds.

WIN32-SHORTCUT

This package allows you to create and configure shortcuts on Win32 systems. Shortcuts are links (.lnk files) to other files or programs, typically represented by an icon on your Desktop or in an Explorer window, with a little arrow in the lower left hand corner.[4]

Example:

```
require “win32/shortcut”
include Win32

s = Shortcut.new('c:\test.lnk')
s.description = "test link"
s.path = 'c:\winnt\notepad.exe'
s.show_cmd = Shortcut::SHOWNORMAL
s.working_directory = "C:\\"
s.save
```

In the above example, we've created a shortcut called “test.lnk” in the 'C:\' directory and gives it a description of “test link”.[5] The shortcut is a link to the Notepad executable. The Shortcut#working_directory method tells it to consider 'C:\' the starting directory for that executable when started. The Shortcut#show_cmd method indicates how GUI applications should start – normal, minimized or maximized.

There are also methods for setting and retrieving information about associated icons, arguments, hotkeys, and more.

WIN32-TASKSCHEDULER

This package allows you to manage and create scheduled tasks on a local or

4 Note that shortcuts on Win32 are not the same thing as symlinks.
5 You can see this if you right-click on the .lnk file and select “properties”.

remote system. Scheduled tasks are similar to the Unix 'cron' utility, in that they allow you to execute programs at specified intervals.

Example:

```
require “win32/taskscheduler”
include Win32

t = TaskScheduler.new

# Enumerate over all existing tasks
t.enum.each{ |task|
        puts “Task name: #{task}”
}
```

Note that creating a new TaskScheduler object doesn't actually create a new task. It merely creates an object used to interface with the TaskScheduler program. If we want to create a task, we need to add a work item and a trigger, then save it.

Example:

```
trigger = {
      "start_month"     => 4,
      "start_day"       => 11,
      "start_hour"      => 7,
      "start_minute"    => 14,
      "trigger_type"    => TaskScheduler::DAILY,
      "type"            => { "days_interval" => 2 }
}
```

A trigger is written as a hash[6]. The above trigger sets a task to start on April 11th (of the current year if not specified), starting at 7:14am and will run daily, every other day. With a trigger prepared, a new work item can be created.

Example:

```
t.new_work_item(“foo”,trigger)
t.application_name = “ruby foo.exe”
t.save
```

This sample actually creates the work item in the Task Scheduler[7]. Note that we have to specify the actual program to execute, as well as save the work item. The work item is **not**created until TaskScheduler#save is called. For convenience, the samples I have shown you here could have been condensed:

```
t = TaskScheduler.new(“foo”,trigger)
t.save
```

In addition to creating new work items, you can configure or delete existing items, set priority, find out information regarding the task such as the most recent run time, or terminate a task as it runs. You can even add multiple triggers to the same work item.

Other Libraries

6 See the docs for all possible valid keys. Note that the “type” key takes a hash as its argument.
7 If you want visual evidence, you can select Start -> Control Panel -> Scheduled Tasks.

There are also packages for enumerating, configuring and creating Windows services (win32-service), reading from and writing to the Event Log (win32-eventlog), native pipes (win32-pipe), a series of packages related to interprocess communication and/or events (win32-ipc, win32-event, win32-semaphore, win32-mutex), extended process handling, including a version of fork for Win32 (win32-process), a native memory mapped file interface (win32-mmap), a library for watching directories or files for changes (win32-changenotify), an library for interfacing with the MS Sound API that allows you to do speech creation and recognition using Ruby (win32-sapi), and a version of popen3 that works on Win32 as it does on Unix systems (win32-open3).

There is also an experimental native thread library (win32-thread), with a fiber interface in the works.

Conclusion

The goal of this article was to demonstrate that there are several useful Ruby libraries for Win32 systems at your disposal that you may not have known about previously. With the help of these libraries I hope you can now use Ruby in production code on Win32 systems.

SHORT BIOGRAPHY

Daniel Berger has been a programmer since 1996, and a Ruby programmer since 2000. He currently uses Ruby for everything from client-server applications to report generation, plus whatever other uses he can think of. He is also the author of the upcoming Pragmatic Bookshelf title “Programming Ruby on Win32”, courtesy of the Pragmatic Programmers.