Python on Windows

Mark Hammond

About Me

- Primary developer and maintainer for the pywin32 package.
  - Started in about 1993
- Primary author of *Python Programming on Win32*
- Co-author of *Programming in the .NET environment.*
- Core Python committer
- Currently working for Mozilla Messaging

Why not stick to cross-platform code?

Python has rich cross-platform support - why target Windows specific features?

- Provide good experience for administrators.
- Provide good experience for users.
- Integrate with other languages and development environments.

Provide a good experience for administrators

- Secure your network connections with NTLM/Kerberos
- Integrate with existing user roles - admins use Windows tools to grant access
- Implement 'daemon' style programs to run as a service to give administrators a single control point.
- Integrate with performance monitoring etc.
- Support local policies

Provide a good experience for users

- Single-signon for convenience (if not security!)
- Use native widgets for navigating the file-system and shell namespace
- Leverage metadata stored in files
- Access printers, network resources, USB devices, detect device removal/insertion, etc

Integrate with development tools

- Some shops have standardized on Windows, not Python
- Use Visual Studio as a development environment.
- Write reusable components for use in languages other than Python
- Consume reusable components written in languages other than Python

The 2 faces of Python on Windows

- "Classic" Python
  - A Win32 environment
  - Implemented in C
- IronPython
  - Runs in the .NET 'managed' environment
  - Implemented in CSharp

CPython on Windows

- Python has first-class support for Windows
  - Eg: Unicode APIs are used to offer seamless Unicode experience.
  - Registry used to configure *sys.path* etc
- Tends to avoid platform-specific functionality:
  - Native windows functions are used to create a cross-platform environment

Directly exposed features

- Not much Windows specific functionality is exposed
  - ctypes or pywin32 is often used to fill in the gaps
- In some cases, Windows features are directly exposed for fundamental application requirements
  - Access the windows registry
  - os.startfile() to open files based on file extensions
  - Distutils supports Windows specific

Windows Integration Options

- ctypes
- Python for Windows extensions
- IronPython

Python for Windows Extensions

- The pywin32 extensions expose Windows specific features
- Broad coverage of many aspects; native IO, services, networking, COM, UI, etc
- Stable and mature
- Downsides
  - fairly large,
  - not easy_install-able
  - A fair bit of C/C++

Broad Coverage

- Use COM objects from Python
- Write COM objects in Python
- Windows Shell integration
- Write Windows services
- Integrate with performance and event logging tools
- Access many Low-level APIs, IIS, etc
- Supports Python 2.x and 3.x

Basis for other toolkits

- Tim Golden's Windows Management Instrumentation (WMI) module
- Vernon Cole's adodbapi
  - standard dbapi to access Windows ADO components
- Mark Rees's ISAPI-WSGI
  - Python WSGI standard API running on IIS
- Used by other toolkits
  - twisted, mercurial, bzr etc all take advantage of pywin32 if found.

IronPython

- Python in the .NET environment.
- Lets us use our favourite language with .NET!
- But what exactly is .NET?

History of .NET:

- Microsoft had COM and ActiveX which provided good cross-language integration.
  - A COM object could be implemented in one language and called from another with relative ease.
  - COM provided "type information" so you could introspect available functions and the parameters they require.

History of .NET – beyond COM

COM worked well but had a number of problems:

- Largely defined around C++ concepts - support for dynamic languages was "bolted on" later.
- Could not avoid a buggy COM object from crashing your program.
- Could not avoid a malicious program from doing evil things.
- A real problem when trying to marry COM with the untrusted internet

.NET is born

- Microsoft decided COM couldn't evolve any further - revolution was needed!
- Research on Virtual Machine technology - particularly Java - offers promise.
- For largely political reasons (including Sun's reluctance to embrace languages other than Java on their VM), Microsoft research started building their own.

So what is .NET?

- A "managed" virtual machine environment.
- Programs compiled to .NET byte-code instead of native machine-language (eg, CPython) or custom byte-code (eg, Python programs).
- But fast JIT compilers means things usually *are* native when it matters

A safety .NET

- Environment manages "safety" of programs:
- Prevents accidental errors such as invalid memory references.
- Prevents malicious code by enforcing access control.
  - Think restricted environments such as your browser (eg, Silverlight)

Why use .NET?

- Windows itself is still largely "unmanaged" code
  - ie, you can integrate with Windows without .NET
- .NET interoperates with “legacy” systems, such as COM
- So why bother with .NET?

Why use .NET?

- The future of Windows?
- Integration with CSharp, VB.NET etc.
- Single development environment across all languages
  - Common debugger, step across language boundaries seamlessly.
- Ability to run in untrusted environments.
- Extensive library

.NET library

- Goes way beyond simply wrapping the Windows API.
- Extensive
  - Common cross-language GUI toolkit
  - Security, signature management, XML, databases, etc
- Stable
  - Part of .NET, so not maintained by each individual language.

Getting in on the action?

- Q) So how do I get some of this action while still using Python?
- A) IronPython!

IronPython

- A Python .NET implementation.
- Not a 'layer' or 'shim' around CPython - a new implementation written in CSharp.
- A first-class .NET citizen - no compromises!
- No real integration with CPython - that's the point!

IronPython history

- Started by Jim Hugunin in his spare time, but Microsoft liked it so much they took him and a team on.
  - No coincidence this sounds a lot like the history of Jython!
- Active community has evolved
- Books, websites, etc.

OpenSource friendly

- Released under a Microsoft Public License.
  - Certified by the OSI as meeting “open source” requirements.
- Works on Mono, an alternative .NET implementation.
- A 'community' version called Fepy is also available
  - Iron->Fe – geddit?

IronPython – Python compatible?

- Yes and No.
  - Language == Yes, environment == No
- Extension modules built for CPython don't work on IronPython.
  - IronPython has reimplements most extension modules from the stdlib.
- Tracks recent Python versions
  - Supports Python 2.6
  - Python 3.x is planned.

Why IronPython?

- Most Microsoft languages are statically typed
  - Not that there's anything wrong with that.
- IronPython appeals on .NET for the same reason Python appeals everywhere!
  - Simple built-in types
  - Interactive, supports introspection
  - etc.

IronPython introspection

- IronPython presents the same interactive session as CPython

```
IronPython 2.6.1 (2.6.10920.0) on .NET
2.0.50727.4927
Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

- Import any .NET namespace and peek inside.

```
>>> import System.Collections
>>> dir(System.Collections)
['ArrayList', 'BitArray',...]
>>>
```

IronPython introspection

- Create and introspect objects

```
>>> a=System.Collections.ArrayList()
>>> dir(a)
['Adapter', 'Add', …]
>>> a.Add("Hello")
0
>>> a.ToArray()
Array[object](('Hello'))
>>>
```

- This object is a native .NET object.

IronPython Tools

- Not many IronPython specific tools – .NET tools generally work!
- IronPython addon for Visual Studio
  - Provides syntax coloring, auto-complete, code browsers etc.
- Full debugger support
  - Full debugging of IronPython applications.
  - Seamless transitions between implementation languages.

Silverlight

- Development platform and environment for 'webby' .NET applications.
- Designed for web apps and mobile devices.
- Desktop applications can also be built
  - Think 'a slimmed down .NET environment'
- Has plugins for IE and Firefox
- Microsoft pushing Silverlight hard

IronPython and Silverlight

- IronPython in the browser
  - Use Python in the browser as well as the server
  - Avoid JavaScript!
- Obvious downside is requirement for Silverlight
  - Particularly useful for corporate intranets where silverlight can be mandated.
  - Less useful for public sites wanting to stick with web standards.
  - Microsoft hoping to make Silverlight as

Silverlight Security Model

- .NET and Silverlight implement a sand-box.
- Silverlight allows you to control the level of trust you apply to applications.
- .NET runtime prevents applications from doing what is allowed.
- No need to trust each individual application/language but instead trust the runtime.

Summary

- Python on Windows goes from strength to strength
  - Works well with the old world order
  - Works well with the new world order
- IronPython being directly supported by Microsoft bodes well for its future.

Questions?

- Any questions?
- Contact
  - mhammond@skippinet.com.au
- Thanks for coming!