ONE LANGUAGE VERSION, ALL PLATFORMS:

PREVENTING INCOMPATIBILITY WITH CROSS-PLATFORM PYTHON, PERL AND TCL

ActiveState

ONE LANGUAGE VERSION, ALL PLATFORMS

QUALITY INGREDIENTS FOR SOFTWARE:

IT departments like to standardize. To be efficient rolling out updates and security patches, installing software, or providing user support, you need to limit the number of variables as much as possible.

Workstations will be set up to run the same version of one operating system on similar hardware. Likewise, servers on the network, though using different hardware and operating systems than the workstations, will be clones of one another wherever possible. Though there are dangers in creating an electronic monoculture (one bad update could take down all systems) the uniformity helps limit the number and type of things that can go wrong. A sysadmin or IT team can become more deeply knowledgeable about those systems, and have a good idea what needs to be done when something goes wrong.

Likewise, desktop software should be kept consistent and up-to-date across the entire company. When someone in one division sends a document or spreadsheet to someone in another division, they should be reasonably confident that the recipient will have the same office software to view it.

However, there is a class of software that is often exempted from this strict but sensible policy. Programs that require an interpreter to run, mostly dynamic languages such as Perl, Python, Ruby, and Tcl, will often be set up with the required runtime in an ad-hoc manner, which can have consequences:

- Subtle breakages in the application. Backwards compatibility is a design goal of Perl and Python releases, but there are sometimes changes which are not backwards compatible.
- Testing needs to be redone. If the software is business or mission critical, a new round of testing may need to be undertaken to verify the software.
- Lock-in to a specific operating system version. You may end up tied to a particular version of an OS because upgrading would change the language version.

These problems can be avoided by using a single language version on all platforms. This approach lets you:

- use the very latest release, or a specific version that is known to work with your software
- use the same version of the language on any operating system
- upgrade—or even change—your operating system while keeping your interpreter version the same - or the other way around
- manage language libraries using language-specific package managers, rather than the operating system's tools or ad-hoc installers

De-coupling the language version from the operating system version and the hardware platform gives system administrators more flexibility or more consistency, whichever is needed.

LANGUAGE VERSION COMPATIBILITY

Point releases are to Perl, Python and Tcl what major version numbers are to less mature technologies. New features are added, bugs are fixed, and often a great deal of code is refactored. But, with so much software depending on these interpreters, compatibility between these point releases is not treated lightly.

ONE LANGUAGE VERSION, ALL PLATFORMS

Perl 6 and Python 3 are beyond the scope of this paper. Python 3 breaks backward compatibility intentionally (though there are tools for partially converting Python 2.x code to Python 3) and Perl 6 should be considered a whole new language.

Perl in particular has an excellent track record of making new releases which are compatible with older code, and the Perl community has codified policies[1] on introducing experimental features and deprecating old ones.

Perl's development team has taken a very safe approach to handling incompatible changes:

In other words, the default behavior is the existing behavior from the previous version. This policy gives developers confidence that the code they write now will still be able to run in future releases of Perl. System administrators will also be more likely to trust that older code will run with a current release.

Python similarly tries to maintain backwards compatibility in point releases.[2]

> *"Any language change which breaks backward-compatibility should be able to be enabled or disabled lexically. Unless code at a given scope declares that it wants the new behavior, that new behavior should be disabled."*
>
> *- "Social Contract about Contributed Modules"*
> RUSS ALLBERY AND THE PERL5-PORTERS.

Like Perl, any incompatible changes have to be introduced gradually. The change has to be described, discussed, and documented. The next release will not implement the actual change, but warn users about the deprecated construct and provide an alternative. After at least one year has passed, the change can then be introduced in the next point release.

The stability ensured by these development policies provides some assurance that code can be migrated to new versions of the language interpreter and core libraries without major breakage, but there are situations where caution is necessary.

CURRENT AND LEGACY APPLICATIONS

Software under active development can keep pace with this systematic and slow language evolution, but legacy software will still be susceptible to breakage when upgrading code that has not been touched in a long time. The approach to maintaining systems hosting these two types of software will necessarily differ.

Software under active development will often be using the most recently available versions of the language and requisite modules simply because the project itself is new. There's no penalty in using the latest releases. The code base will be well known by the developers, so incremental patch and version updates can be applied with some degree of confidence that the application will continue to work. At the very least, they will have a good idea where it might break.

In this situation, developers can take advantage of new features in the language, and the application can run on an up-to-date interpreter.

Legacy software on the other hand will often need to use the language version it was originally built and tested with. If nobody in the organization is familiar with the code, updating the software to work with the

ONE LANGUAGE VERSION, ALL PLATFORMS

newest language release could be risky, expensive, or just not worth the hassle. Reliability is more important than keeping up with the language.

In either case, being tied to the operating system's bundled distribution of Python, Perl or Tcl is not optimal. It limits your ability to set up older or newer versions as necessary to match the application.

INTERPRETER VERSIONS

Operating system vendors have a lot of software packages to worry about. Keeping their versions of Perl, Tcl and Python completely up to date is fairly low on their priority list. Most will provide updates for critical security problems, but generally they're a bit behind the current language releases. Some are better than others:

Ubuntu 14.04.3 LTS has Perl 5.18.2 and Python 2.7.5 [3]

RedHat Software Collections 2.0 has Perl 5.20.0 and Python 2.7.5 [4]

Even though these are reasonably modern releases at the time of this writing, in two or three years they will be substantially obsolete. Furthermore, there is a "what you get is what we have" aspect to these systems: if you are running Perl 5.24.1 on Windows, it makes sense to run the same version on Ubuntu or RedHat or OSX, not the one the vendor provides.

Security updates are one thing that drive updates to language interpreters, as vulnerabilities in OpenSSL and other libraries are found and patched. The ability to update all interpreters across all platforms to a new version with the latest security updates is a big advantage to the modern IT organization.

"DOWNGRADE-ABILITY"

Though keeping the dynamic language interpreters fully patched and up to date is the ideal scenario, reality sometimes dictates that older interpreters are required. Crucial legacy applications may require an older version of the language, whose bugs have become features.

If the system interpreter is the only one available, this could mean holding off on a system update to prevent breaking the application. This in turn might mean leaving the system with unpatched security or stability issues. Upgrading the bundled interpreters on their own is generally not an option, as they are often deeply integrated with the OS package manager or other system administration tools.

A preferable option would be to allow the system interpreters to stay in sync with the OS and run the target application with a completely separate installation of the language. This keeps the OS upgrade schedule independent of application requirements, and allows for much finer grained control of the language environment. Perlbrew and virtualenv are useful tools in this regard.

PLATFORM INDEPENDENCE

Despite striving for homogeneous IT environments, most organizations deal with a mix of server operating systems. Over time, any server room will accumulate hardware and software that reflect the preferences of the IT managers and the demands or prerequisites of the applications being hosted. Even when efforts are made to standardize, there's often one or more "odd ones out" in an otherwise homogeneous network. Sometimes desktop systems are pressed into service as servers.

1. http://perldoc.perl.org/perlpolicy.html#BACKWARD-COMPATIBILITY-AND-DEPRECATION

2. http://www.python.org/dev/peps/pep-0005/

ONE LANGUAGE VERSION, ALL PLATFORMS

Running business-critical applications across disparate operating systems is not ideal, but it often happens when one or more pieces of the infrastructure puzzle requires specific hardware or software. In this situation, minimizing the variables by standardizing on a particular language distribution (and version) is essential.

Though most applications will not need to operate on multiple platforms or operating systems, backup scripts, monitoring software, and other network tools often will. To ensure reliable operation, these applications should run on the same version.

DEVELOPMENT TO PRODUCTION

A more common case of language environment mismatch is the discrepancy between developer workstations and production servers - even when both sets of machines are running Linux.

Corporate production servers tend to be dominated by Red Hat Enterprise Linux and Suse Linux installations, with major version upgrades happening much less frequently than in typical desktop distros. Applications developed on Ubuntu using the default interpreters may encounter language-level or module availability problems when migrating to the production Linux environment. The problem is even worse when moving from Linux to Solaris, HP-UX or AIX.

PACKAGE MANAGEMENT

Any discussion of Python, Perl or Tcl version compatibility needs to discuss their extensive corpus of third- party public modules. These languages are popular in part because so much additional special-purpose code has been written by and given back to the community at large.

Distributions of these language needs to give easy access to as many of these as possible. The CPAN shell for Perl and easy_install or pip for Python are excellent, but these are not binary package managers (i.e. any C/ C++ code needs to be compiled at install time) so module installation is more error prone. The Tcl community has no centralized system for publishing public modules, so packages need to downloaded from a variety of sites in a variety of formats.

Most Linux and UNIX systems have their own package managers for installing software. They are tremendously useful for installing system software, but they are limited in their ability to deliver language modules. Red Hat Enterprise Linux 5 has access to 47 Perl packages and 40 Python packages through its package manager.

As with the language packages themselves, these modules are often quite out of date compared to what is available in CPAN or PyPI.

Compare this to more 13,000 modules for Perl 5.20 and above, available from the ActiveState repositories via PPM.

As in the language itself, there is a strong tradition in the Perl module author community of backwards compatibility. Code that works with an older version of a particular module should generally work with the latest version.

3. *http://distrowatch.com/table.php?distribution=Ubuntu*
4. *http://distrowatch.com/table.php?distribution=redhat*

ONE LANGUAGE VERSION, ALL PLATFORMS

With Python packages we recommend downloading your favourites from PYPI using pip and the binary wheel format. Most popular packages have wheels now available so you don't have to worry about building these modules yourself. In addition, ActiveState's Python distribution already ships with many of the most common packages already built.

ACTIVEPYTHON, ACTIVEPERL, AND ACTIVETCL

ActiveState's dynamic language distributions are uniquely positioned to solve cross-platform, version and module compatibility problems. Since development and build maintenance efforts are focused on the languages themselves rather than a specific operating system, customers can take advantage of:

- a choice of up-to-date versions from the main production branches of:
  - Python - 3.5, 2.7
  - Perl - 5.24, 5.22
  - Tcl - 8.4, 8.5, 8.6
- builds that are consistent across all available platforms
- access to older builds for supporting legacy software
- binary packages for thousands of modules via PPM

ActivePython, ActivePerl, and **ActiveTcl** are the only commercially supported distributions available for Windows, Linux, Mac OS X, Solaris, HP-UX and AIX.

Community Edition is a free distribution intended for non-commercial use available for Windows, Mac OS X and Linux.

Business Edition adds access to 77 archival versions of Perl and 50 versions of Python, so if your software has very specific compatibility requirements, a build is almost certainly available that will fit.

Enterprise Edition offers additional levels of technical support, optional IP indemnification, and can include custom language or module builds.

Talking to a Dynamic Language Expert

If your organization is facing challenges with Python, Perl, or Tcl, the specialists at ActiveState can go through the specific requirements of your projects with you to see which dynamic language distribution will work for you.

ActiveState Software Inc.
sales@activestate.com

Phone: **+1.778.786.1100**
Fax: **+1.778.786.1133**

Toll-free in North America:
1.866.631.4581

ActiveState®

ABOUT ACTIVESTATE

ActiveState believes that enterprises gain a competitive advantage when they are able to quickly create, deploy and efficiently manage software solutions that immediately create business value, but they face many challenges that prevent them from doing so. The company is uniquely positioned to help address these challenges through our experience with enterprises, people and technology. ActiveState is proven for the enterprise: more than two million developers and 97 percent of Fortune 1000 companies use ActiveState's end-to-end solutions to develop, distribute, and manage their software applications written in Java, Perl, Python, Node.js, PHP, Tcl and other dynamic languages. Global customers like Cisco, CA, HP, Bank of America, Siemens and Lockheed Martin trust ActiveState to save time, save money, minimize risk, ensure compliance and reduce time to market.