IST FP6-004779

PYPY

Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community

STREP

IST Priority 2

D02.3 Automated Testing and Development support library

Due date of deliverable: March 2007

Actual Submission date: March 23rd, 2007

Start date of Project: 1st December 2004 **Duration: 28 months**

Lead Contractor of this WP: merlinux

Authors: Holger Krekel, Guido Wesdorp (merlinux)

Revision: Final

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level: PU (Public)

PyPy D02.3: Development and Testing library

Revision History

Date	Name	Reason of Change
Dec 01 2006	Carl Friedrich Bolz	Created a rough structure, outline
Jan 21 2007	Holger Krekel	Draft of exec summary, outline & abstract
Jan 28 2007	Carl Friedrich Bolz	Publishing of intermediate version on the web
Mar 10 2007	Holger Krekel	Filled and refined all sections
Mar 10 2007	Guido Wesdorp	More refinement, small additions
Mar 10 2007	Holger Krekel	Review, refactoring for Interim Version
Mar 12 2007	Carl Friedrich Bolz	Publish Interim Version on the website
Mar 18 2007	Holger Krekel	Incorporate feedback Grig Gheorghiu/Brian Dorsey
Mar 23 2007	Carl Friedrich Bolz	Publish Final Version on the website

Abstract

We describe PyPy's easy-to-use testing and development support tools, separately released as the "py lib". We discuss feature highlights that ease the writing of cross-platform applications and tests. The `py.test` tool pioneers new testing approaches aiming to ease and speed up the development process. Its development was driven by testing needs that arose in the development of the PyPy interpreter and Virtual Machine translation framework - a complex project that contains several thousand automated unit and functional tests, running in several languages and operating systems. We provide an overview on how the test tool and the py library are used for PyPy's agile development process.

Purpose, Scope and Related Documents

This document provides higher level information for developers and prospective users about PyPy's development support "py" library. Particularly, we highlight features and reference starting points for using the `py.test` testing tool, with a particular focus on its past and future uses in the PyPy project.

This document provides a conceptual overview of PyPy's testing tool and development support library. For detailed technical information, we refer to online entry points and API documentation.

Related Reports:

- D02.1 Configuration and Maintenance of Development Tools and website [D02.1]
- D12.1 High-Level Backends and Interpreter Feature Prototypes [D12.1]
- D13.1 Integration and Configuration [D13.1]
- D14.5 Development Process [D14.5]

PyPy D02.3: Development and Testing library

Contents

1 Executive Summary

Development and research within the PyPy project relies on automated testing and a test-driven development approach. PyPy Interpreters and its Virtual Machine translation framework have complex and diverse testing needs. For automated testing, the Python community traditionally uses the `unittest` standard library module and builds project specific extensions on top of its classes. Projects usually extend the test collection, test execution and test reporting process by means of subclassing. However, this approach often intermingles project specific integration code with these advanced features and thus prevents re-use from other projects.

Instead of writing such project specific extensions, we wrote and improved `py.test`, an highly flexible testing tool. Projects may modify and amend aspects of the testing process without requiring intrusive changes to the actual testing code. It may be used to run tests remotely or to perform documentation and reference integrity checks. `py.test` minimizes the overhead needed to implement project-specific processes for test writing, collection, execution or reporting.

For the developer, `py.test` emphasises a quick learning curve and provides detailed feedback in case of test failures. There are several facilities to automatically observe context and introspect a failing program. Usage is not limited to testing Python code, or tied to a particular PyPy interpreter implementation. In fact, it is possible to run compliance tests for the emerging PyPy JavaScript interpreter or to drive GUI and acceptance tests.

More recently, we pioneered ad-hoc distributed testing, which enables distribution of PyPy's thousands of tests to multiple computers. Since running tests relating to PyPy translation of code snippets down to C or .NET code is time consuming, ad-hoc distributed testing has helped to considerably speed up development cycles. Another more recent addition is a documentation generation tool [APIGEN] that incorporates information gathered while running tests into documentation.

Throughout 2004 to 2007 we incrementally developed the `py lib` using a test-driven approach with tests written in `py.test` itself. Today, many projects, developers and organisations use it. Talks about the library have been presented at multiple Python Community conferences worldwide. It is perceived as one of the major Python testing tools (see [TAXONOMY], [BLOG1]), its test collection and test state management concepts have been adopted by other projects such as nose [NOSE].

The experiences obtained during the execution and implementation of the PyPy EU project show that high quality testing tools can positively impact development speed and quality. Several individuals from the PyPy project and an emerging community are devoted to continue development and co-operate on further advancements of the py lib beyond the EU project duration. In addition to implementing features, plans include `py lib` education and implementing targeted commercial uses to enable quick automated cross-platform testing processes.

2 py lib overview

The py lib provides a consistent set of modular support tools and code for developing and testing application code. We discuss a few innovative areas below and list interesting entry points into online and API documentation.

2.1 py.test: the testing tool

A main benefit of using `py.test` is that it's easy to learn: it requires very little boilerplate code. A developer can immediately start to write and run tests for his application. As the number of tests grows, and new testing needs arise in a project, more features and extension possibilities can be used, usually without modifying previously written test code.

A main difference to other approaches is py.test's separation between test code and test configuration and extensions. For example, to write new reporters or to check for memory leakage around test methods, no intrusive changes to the actual test classes and functions are required.

Particular focus lies on reporting about and interacting with the application context when a test failure is encountered. One may enable additional reporting of variable values, or drop to an interactive Debugger to introspect variables and the execution call stack.

Using `py.test` does not require the programmer to hook test classes into framework class hierarchies. In fact, the test collection process discovers tests by recursively searching for test files, classes and methods, expecting certain naming conventions. If needed, these naming conventions are modifiable. Systematic hooks and temporary test directory management allow fine grained and easy control over test setup and state at the module, class or method level.

For testing Python programs, `py.test` conveniently modifies the `assert` statement so that it provides detailed information about the involved values which result in a failed Assertion. Most other approaches require the use of a set of methods to express assertions about values. With `py.test`, assertions about truth values are regular Python expressions.

When debugging and testing code, it is common practice to print variable values and progress information. In most testing frameworks the developer removes this output once the bug has been found to avoid cluttering up the output of running the test suite. This useful code can be kept in `py.test` based tests, since it automatically hides all printed output for successful tests. This allows the test code to print useful debugging output without distracting the developer until it is actually needed.

`py.test` allows modification of all of its test collection, execution and reporting behavior through the use of per-project or shared `conftest.py` files. Using this facility, we have implemented a wide range of project specific customizations, such as excluding test collection for certain directories and enabling remotely run Windows GUI tests from a Linux machine in an ad-hoc manner. Configuration and implementation aspects are documented online [CUST].

Recently we introduced ad-hoc distributed testing. By running tests on several hosts at once, overall test runs receive a considerable speed up. The only prerequisite for using distributed testing is a remote computer that allows access to a runnable Python interpreter, for example through the popular Secure Shell (SSH) server. For platforms such as Windows where SSH is not common, a light-weight server is provided.

A recently developed web application showcases both PyPy's and the py lib's flexibility. It displays test progress in real time and allows the developer to interactively view failures from distributed test runs via a web browser. The application extends py.test using test reporting hooks and uses PyPy's translation tool chain and JavaScript backend to produce client-side JavaScript from the python implementation.

For a more detailed feature list, architecture and extension possibilities, please refer to the online py.test documentation [PYTESTDOC].

2.2 Ad-hoc Distribution of Program Code

Also driven by requirements of the testing tool, the py library provides a new remote execution model which makes full use of Python as a very dynamic language to compile and execute program text at run time. Through its py.execnet namespace [PYEXEC] and lean API, it provides means to seamlessly instantiate and interact with both local and remote processes instantiated through Secure Shell or socket connections.

The data communication protocol between the sending and the receiving side is determined by the sender. There are no installation requirements on the remote side, except that of a runnable Python interpreter. Once set up, one may send program functions and fragments to the remote side and send and receive basic Python objects via **Channels**. Effectively, this allows the creation of a distributed application without the need to define global interfaces and without any installation overheads.

`py.test` itself uses this method of distributing programs to perform ad-hoc distributed testing. Within the PyPy project it is also used for co-ordinating translation build jobs and for system maintenance tasks such as replicating subversion repositories. It has received encouraging feedback at conferences and from individuals, and is at the verge of providing a practical model for distributing and controlling processes and applications within larger networks of heterogeneous platforms.

2.3 Rich API Documentation

With its public release, all of the specified exported API of the py library is documented online [PYDOC]. In addition to the information provided by other python documentation tools, information collected from test runs provides insights into the types of arguments, return values as well as call chains. This documentation is generated using `apigen`, a component recently added to the py lib.

In dynamic languages like Python it can be challenging to find out function argument or return types, and discover its call sites. In well-tested environments, most of this information can be automatically derived by monitoring test runs. Apigen monitors function calls during test runs to include such information in the documentation. This adds valuable information to the API documentation, and additionally serves as an indicator for the quality of the tests.

2.4 Other Features / Documentation Entry points

Besides `py.test`, the py lib provides support for higher level access to system, language and control flow aspects. The main online documentation entry point is here:

http://codespeak.net/py/dist/index.html

Below is a brief list of main features and entry points on the website:

- **Path objects** for uniformly accessing local file systems local and remote subversion file systems.
- **greenlets** provide advanced concurrent programming idioms, used by PyPy's distributed execution prototype as well as other projects. Greenlets provide a subset of [STACKLESS] Python's features using a standard CPython or PyPy interpreter.
- **py.code** for high level access to Python Code and Traceback information, substituted and extended by PyPy to connect it to its own Interpreter objects.
- **py.log** for producing and logging events to various channels.
- **py.xml** for generating and rendering XML objects, used for generating web pages.

2.5 Availability and Releases

The `py lib` is permanently publicly accessible through codespeak's Subversion repository (see D02.1 for more information). Several outside developers stated that they enjoy the ability to follow the development of the tool and have encountered few problems just using the latest repository version - the same approach taken for developing the PyPy interpreter and compiler code base. Since 2007 the py lib split "trunk" development from a more stable "dist" branch exposing stable development snapshots.

In February 2007, the 0.9.0 version was released [0.9.0] to the public under the MIT license, including extensive introductory and API documentation. It contains all of the features described in this report, including ad-hoc distributed testing reported through a PyPy generated AJAX web application. It can be used on Windows, Mac OS and Linux platforms and runs on top of CPython 2.3, 2.4. and 2.5.

3 Usage of py lib Facilities in PyPy

PyPy uses the py library for various purposes, such as accessing the file system, event logging, to distribute translation build jobs and to generate web pages and PDF Reports (including this one) using its py.rest tool [PYREST].

PyPy's architecture [ARCH] requires testing to occur at multiple levels. Project specific extensions instrument `py.test` to recognize and execute several test types:

- Interpreter level tests: being instrumented and set up with an Object Space, these tests run directly on CPython and test low level PyPy interpreter or translation aspects.
- Application level tests: these tests are instrumented and set up with particular Object Spaces and PyPy's own Python Interpreter, running itself on CPython. Additionally, these tests can run directly on a translated PyPy version (which is considerably faster as it avoids the double interpretation overhead).
- Compliance tests: the CPython regression tests are instrumented to run either through PyPy's interpreter, or directly on a translated PyPy version.
- JavaScript tests: ECMA regression tests (written in JavaScript) are instrumented to run on top of the emerging JavaScript PyPy interpreter.
- Documentation tests: web site generation and full link and reference integrity tests are executed from a special extension of `py.test`.

These test types co-exist in the PyPy project and can be uniformly driven by `py.test` invocations. The PyPy coding guide [PYPYGUIDE] introduces basic testing mechanisms. and the PyPy testing information on web pages [PYPYTEST] references the results of daily run tests and translation jobs.

To speed up running of a large number of tests, the above test types can be distributed to multiple hosts. This has encouraged developers to perform larger test runs before committing a change to the code base, effectively speeding up the development process.

References

[PYDOC] *py lib documentation*, http://codespeak.net/py/dist/apigen/api/index.html

[0.9.0] *py 0.9.0 release annoucnement*, http://codespeak.net/py/dist/release-0.9.0.html

[PYTESTDOC] *py.test documentation*, http://codespeak.net/py/dist/test.html

[APIGEN] *py API documentation generation*, http://codespeak.net/py/dist/apigen.html

[PYEXEC] *py.execnet documentation*: http://codespeak.net/py/dist/execnet.html, *py.execnet api documentation*: http://codespeak.net/py/dist/apigen/api/execnet.html

[CUST] *py.test Customization*, http://codespeak.net/py/dist/impl-test.html

[D02.1] *Configuration and Maintenance of Development Tools and Website*, PyPy EU-Report, 2007

[D14.5] *Documentation of the Development Process and Sprint Driven Development*, PyPy EU-Report, 2007

[D12.1] *High-Level Backends and Interpreter Feature Prototypes*, PyPy EU-Report, 2007

[D13.1] *Integration and Configuration*, PyPy EU-Report, 2007

[NOSE] *nose testing tool*, http://nose.python-hosting.com/

[STACKLESS] *Stackless CPython*, http://www.stackless.com/

[TAXONOMY] *Python testing tools*, http://pycheesecake.org/wiki/PythonTestingToolsTaxonomy

[BLOG1] *Blog post about py.test*, Grig Georghiu, http://agiletesting.blogspot.com/2005/01/python-unit-testing-part-3-pytest-tool.html

[PYREST] *py.rest document conversion tool*, http://codespeak.net/py/dist/bin.html#py-rest

[ARCH] *PyPy architecture*, http://codespeak.net/pypy/dist/pypy/doc/architecture.html

[PYPYGUIDE] *PyPy coding guide*, http://codespeak.net/pypy/dist/pypy/doc/coding-guide.html#id2

[PYPYTEST] *PyPy testing information on web pages*: http://codespeak.net/pypy/dist/pypy/doc/index.html#status