
ilabs Documentation

Release 1.2.1

thanos vassilakis

December 16, 2016

1	integrationLabs aka ilabs	3
2	Installation	5
3	Usage	7
4	Contributing	9
4.1	Types of Contributions	9
4.2	Get Started!	10
4.3	Pull Request Guidelines	10
4.4	Tips	11
5	Credits	13
5.1	Development Lead	13
5.2	Contributors	13
6	History	15
6.1	0.1.0 (2003-10-16)	15
7	Indices and tables	17

Contents:

integrationLabs aka ilabs

python enterprise integration framework project. Powerfull class library based on EAI patterns and a modeling and simulation tool.

In the last ten years, corporations have grown exponentially in complexity. This has been driven by the increase in sophistication and globalization of theirs markets. This growth has been matched by the growth of their IT infrastructure. The most important element in this growth of enterprise integration needs is the message, its creation, processing, transportation and storage. integrationLabs develops a simple but powerful integration framework to process this fundamental organizational element.

What is the message? The message is the fundamental unit of enterprise communication, it is an order, a trade, an email, a contact, an appointment, a line in a log file, critical system information, a document, it's in fact anything that needs to be communicated. Messages have to be created or extracted, translated, filtered, analyzed, routed, transmitted, stored, and displayed. All these actions form the integration circuitry of an enterprise. integrationLabs is the framework to build these circuit parts and is the tool to do wiring.

What is integrationLabs? It supplies developers with a framework to develop the needed integration circuitry.

Powerful Simplicity With integrationLabs integration, systems are far quicker to develop than with other more heavy and complex products. Because of the elegance and simplicity of its foundation, integration systems built with integrationLabs systems tend to be 3-5 times smaller than their equivalent built with systems such as webMethods. This results in a major reduction of TOC. Ilabs integration systems are easier to maintain. The simple and elegant methodology results in clean design. integrationLabs is an extremely versatile. It abstracts the underlying technology. Allowing a mix of programming languages, platforms and protocols to exists. This style of legacy leverage again reduces TOC and time to production. Implementation costs and infrastructure expenses are kept to a minimum. integrationLabs object-oriented paradigm is the most powerful and easy to use of any integration framework. It creates dynamic, resilient circuitry whose behavior can be adapted at run-time. This allows for unobtrusive releases and seamless integration. Again powerful simplicity allows for rapid development of more complex integration systems.

Portability integrationLabs is available on an incredibly wide range of hardware and software platforms. This includes the usual suspects: Sun, Intel, IBM, Microsoft Windows variants, Macintosh OS variants and all Unices. However, integrationLabs has also found its way into a wide range of less well-known platforms, including PDAs and set-top boxes. By abstracting the technology, integrationLabs plays well with all programming languages. integrationLabs systems can be extended using C, C++, Java, python, erlang, scheme, etc! It can also be embedded in programs written in these languages. integrationLabs comes with the usual barrage of adapters which support almost all standards, fix, swift, CORBA, COM, SOAP, XML, XML-RPC and so on. Due to the simple design, integrationLabs works well with other third-party adapters. integrationLabs provides the option for integration with low-level APIs for both the Windows and Unix platforms, allowing applications to be highly platform compatible.

- Free software: ISC license
- Documentation: <https://ilabs.readthedocs.org>.

Installation

At the command line:

```
$ easy_install ilabs
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv ilabs  
$ pip install ilabs
```

Usage

To use ilabs in a project:

```
import ilabs
```

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at <https://github.com/thanos/ilabs/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

ilabs could always use more documentation, whether as part of the official ilabs docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/thanos/ilabs/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *ilabs* for local development.

1. Fork the *ilabs* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ilabs.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ilabs
$ cd ilabs/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ilabs tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/thanos/ilabs/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_ilabs
```

Credits

5.1 Development Lead

- thanos vassilakis <thanosv@gmail.com>

5.2 Contributors

None yet. Why not be the first?

History

6.1 0.1.0 (2003-10-16)

- First release on PyPI.

Indices and tables

- `genindex`
- `modindex`
- `search`