Isomer Documentation

Release 1.0.10

Isomer Contributors

Oct 06, 2020

Contents

Version 1.0

Release 1.0.10

Date Oct 06, 2020

CHAPTER 1

About

1.1 Hackerfleet

1.1.1 Who we are

The Hackerfleet is a research & development venture founded by some friends who decided to revolutionize the maritime technology sector.

We develop opensource hardware and software for unmanned and manned vessels on all waters, we do this publicly and transparent, our repositorys are open for you.

1.1.2 Our goals

One of our primary goals is to establish a better communication network between compatible hardware, for automatic data exchange between ships.

We want to aggregate all the information that is currently thrown away on ship-bridges all over the world and make the best free map of the ocean.

This vast resource of currently nearly unused data will also help scientists understand our oceans better.

1.1.3 Timeline

- 2011 Founding -> CCC Camp MS 0x00
- 2012 Hackathon for Android App 'Social Bearing'
- 2013 Mariner's code: Computer hackers conquering the high seas
- 2014 EuroPython Hackathons
- 2015 Oh, camp again! We did some crowdsourced management

And now, we're here!

You can check all this out on the intertubes. Youtube, CNN, etc. Just search for 'Hackerfleet' - hmm.. succinct name, eh?

1.1.4 Founders

- Heiko 'riot' Weinen (riot@c-base.org, @__r107__, ri0t@github)
- Johannes 'ijon' Rundfeldt (ijon@c-base.org, @aegrereminiscen, ij0n@github)

Meet us at c-base, the spacestation below Berlin Mitte!

1.1.5 Communication

- github.com/Hackerfleet
- Twitter.com/hackerfleet
- Facebook/hackerfleet
- hackerfleet.soup.io
- Also on G+
- irc #hackerfleet on freenode

Note: Please be patient when using IRC, responses might take a few hours!

1.2 Isomer

1.2.1 About Isomer

The Isomer framework is being developed specifically to target a handful of properties and challenges, that are unique to the projected use of the system:

- Locally offline and undisruptable operation (True Internet!)
- Extremely low energy profile
- Must work on embedded systems with low memory, storage and computing capacity
- Realtime handling and federation of incoming and outgoing data
- Many, many different bus and sensor systems as well as configurations
- Clients should not be limited in any way

To master all these challenges, a rather radical approach was chosen after evaluation of most of the currently available frameworks and libraries.

1.2.2 Isomer System overview

Architecture

The system consists of two parts:

1. A backend written in Python. It handles communications, data handling and other general services provided by independent modules.
2. To communicate with users, a HTML5 based Frontend is deployed to most modern web browser capable clients.

Meshed Operation

The cloud/server-less mesh operation enables local independence and adaptability regarding network environments.

No platform specifics

It also eliminates the need to write platform specific applications (e.g. native Android or other mobile platform applications)

1.2.3 What's new here?

At first glance, Isomer looks like just another web application platform.

In contrast to most available systems though, Isomer works using a component based frontend and backend architecture.

This enables every installation to install, activate and use only the modules relevant for the local group of users.

Also, communication between clients and the backend has been streamlined and minimized by relying on Websockets.

CHAPTER 2

User's Manual

2.1 Getting Started

This part shall guide you to a quick installation and setup of your own Isomer instance.

2.1.1 Quick Start Guide

Docker

We're providing Docker images and composer files for quick and easy installation.

The command to get the latest isomer is:

```
docker-compose -f docker/docker-compose-hub.yml up
```

This will spin up a database and Isomer itself. If that worked, you should *head over to the setup*

If you run into trouble, check out the *docker section of the developers manual* or try the manual installation:

2.1.2 Manual Installation

If you run into trouble or get any unexpected errors, *try the complex installation procedure*, which details all the automated bits and steps.

Note: We're working on a detailed error handling system that includes links to online documentation and ad-hoc advice on how to fix problems.

Concepts

To run an Isomer instance, it makes sense to get familiar with some terms:

Term	Definition
Local Management	executing local commands to manage isomer systems
Management Tool	*iso* or *isomer* is the core application which handles instance management and general setup
Instance	A single Isomer platform definition, providing environments to run
Environment	The working parts of a single Isomer platform i.e. the installed backend, modules and user data
Module	Plug-In functionality for Isomer platforms
Remote Management	Using a local management tool to configure and maintain remote hosted Isomer systems and instances

Install minimum dependency set

Please make sure, you have python3 as well as python3-setuptools installed.

Get Isomer

Currently, getting Isomer via git is recommended. We are working on Python packages, packages for multiple distributions as well as ready made images for various embedded systems.

```
git clone https://github.com/isomeric/isomer
cd isomer
git submodule update --init
```

Install Management Tool

The management tool's automatic installation currently only supports Debian based systems.

Tip: Feel free to contribute installation steps for other distros - that is mostly adapting the package manager and package names in isomer/tool/defaults.py

First, install the local management system:

```
cd ~/src/isomer
python3 setup.py install
```

Test the Tool

Now run

```
iso version
```

to see if the tool installed correctly. It should print a few lines detailing its version number and invocation place.

Set up the system

To run securely and provide a robust upgrade and backup mechanism, your system needs a few things set up:

- a user account for running instances
- some paths in */var/lib/isomer*, */var/local/isomer*, */var/cache/isomer* and
- a configuration skeleton in */etc/isomer*

Setting these up is done automatically by invoking

```
iso system all
```

Create an Instance

Now you should be able to create and install your instance:

```
iso instance create
iso instance install
```

If that runs through successfully , you should *head over to the*.

Planned Installations

- We're planning to offer ready-made SD card images for various embedded systems.
- A custom NixOS system is planned as well.

2.1.3 Requirements and Dependencies

Backend

Isomer' backend has a few dependencies:

- Python: >= 3.3 (or possibly pypy >= 2.0)
- Database: MongoDb

Note: We have phased out Python 2.7 support.

A few more dependencies like nginx, and some python packages provided per distribution are recommended, but not strictly necessary.

The Isomer Python package additionally installs a few pure Python libraries:

- Circuits
- Click and a few supporting packages
- PyMongo
- PyOpenSSL
- PyStache
- JSONSchema
- DPath
- DeepDiff

  Supported Platforms Linux

  Supported Python Versions 3.3, 3.4, 3.5, 3.6, 3.7

Frontend

The frontend is built with

- node
- npm

and others. The detailed list can be found in frontend/package.json after pulling the frontend git submodule.

2.1.4 Downloading

Latest Stable Release

By design, there is currently no stable release planned.

The latest stable releases (if there should ever be one) could be downloaded from the Releases page (*specifically the Tags tab*).

Latest Development Source Code

We use Git for source control and code sharing.

The latest development branch can be cloned using the following command:

```
git clone https://github.com/isomeric/isomer.git
cd isomer
git submodule update --init
```

For further instructions on how to use Git, please refer to the Git Website.

2.1.5 Installing

First of all: The manual installation procedure is rather complex right now and the documentation is being overhauled for the 1.0.0 release of Isomer.

We've *simplified the process by supplying an install script*, but if you encounter any trouble/problems, checkout these detailed installation steps.

If you still can't get it to install, *contact us via irc or emai* and we'll happily try to help you get your installation running.

This is very important for us, since the system has not yet been deployed very often and we're not yet aware of all of the pitfalls and traps on that route.

Warning: **Isomer is not compatible with Python 2 and 3.2!**

2.1.6 Manual Installation

These instructions are for Debian or Ubuntu based systems. Installation on other distributions is possible and being worked on.

Preparation

Before doing anything with Isomer, be sure to have all the dependencies installed via your distribution's package manager.

For Debian Unstable use this:

```
sudo apt-get install nginx mongodb python3 python3-pip python3-grib \
                       python3-bson python3-pymongo python3-serial
```

If you want (and can, depending on your platform/distribution), install the mongo and bson extensions for speedups:

```
sudo apt-get install python3-pymongo-ext python3-bson-ext
```

The system will need to get a bunch of more dependencies via npm to set up the frontend, so install npm and if necessary the nodejs-legacy-symlink package. The simple (but not so good) way is to use Debian's packages:

```
sudo apt-get install npm nodejs
sudo npm install npm@4.2.0 -g
```

The better way is to install nodesource.

If you want to install the full development dependencies to write documentation as well, you need to install the enchant package:

```
sudo apt-get install enchant
```

In case you want to use raster (or in future: vector) charts in Isomer' map module, you'll need to install libgdal and its binaries:

```
sudo apt-get install gdal-bin python-gdal
```

Note, that it is necessary to install python-gdal 2.7 - not the python3 variant, as the scripts are not included in that.

Getting the source

To initially obtain the development source code if you don't have it already, use git thus:

```
mkdir ~/src
cd ~/src
git clone https://github.com/isomeric/isomer
cd isomer
git submodule update --init
git pull
cd frontend
git pull
```

Backend

Caution: This is currently outdated!

The management tool usually can install everything you need. It starts by adding a new system user for Isomer and generating a (currently only self signed) certificate.

The process also involves installing the supplied modules, getting the frontend dependencies, building and installing the documentation, etc.

It also creates a few folders in /var (lib/isomer, cache/isomer) for cache data and other stuff as well as install basic default provisions into the database.

Finally, it installs and activates a systemd and nginx service script to launch Isomer on bootup and make it available to users.

```
virtualenv -p /usr/bin/python3 --system-site-packages venv
source venv/bin/activate
pip install -Ur requirements.txt
python setup.py develop
sudo venv/bin/python ./iso system all
```

If you want to develop (documentation) as well, you'll need to use the *requirements-dev.txt* instead of the normal one.

If you want to manually start Isomer, invoke the launcher thus:

```
sudo ./venv/bin/python iso launch
```

Running the launcher as root to be able to open ports below 1024 should be safe, as it drops its root privileges, unless you specify –insecure, which is strongly discouraged and only meant for development purposes. The default is to use port 8055 and relay that with the supplied nginx site definition

Documentation

Before building any documentation, you'll need to install the `requirements-doc.txt` (located in the Isomer repository root):

```
pip install -r requirements-doc.txt
```

Manual build

To build the html documentation, change to the docs subdirectory and use make to build the files:

```
cd docs
make html
```

The built files will reside in `isomer/docs/build/html`.

You can also build the PDF file (and various other formats) by using the Makefile inside the docs directory.

```
cd docs
make latexpdf
```

The rendered pdf output will reside in ``isomer/docs/build/pdf

Just running make without arguments gives you a list of the other available documentation formats.

Automatic build & installation

Caution: This is currently outdated!

The documentation is available online on ReadTheDocs.org. If you wish to build and install the included documentation for offline use, run these commands:

```
sudo ./venv/bin/python ./iso install docs
```

This installs all necessary documentation tools and copies the files to the expected Isomer web data folder.

Installing from a Source Package

If you have downloaded a source archive, this applies to you.

```
python3 setup.py install
```

For other installation options see:

```
python3 setup.py --help install
```

Installing from the Development Repository

If you have cloned the source code repository, this applies to you.

If you have cloned the development repository, it is recommended that you use setuptools and use the following command:

```
python3 setup.py develop
```

This will allow you to regularly update your copy of the Isomer development repository by simply performing the following in the Isomer working directory:

```
git pull -u
cd frontend
git pull -u
```

Note: You do not need to reinstall if you have installed with setuptools via the Isomer repository and used setuptools to install in "develop" mode.

Windows & OS X installation notes

These instructions are WiP. The easiest way to get Isomer on Win7 or newer is to install and user Docker or a virtual machine

To install on Windows, you'll need to install these packages first:

- Python 3.5 https://www.python.org/downloads/windows/
- MongoDB https://www.mongodb.org/downloads#production
- pymongo
- numpy

Platform specific

There are some collected instructions for various hardware platforms:

Raspberry Pi

Swap

Since this machine doesn't have much RAM, don't forget to add a swap partition or file.

Linux

Debian

Newer versions of Debian are supported. In fact, riot develops on a healthy mix of Debian Testing/Unstable/Experimental.

Ubuntu

Mostly the same as with Debian, that is why this is only a link to Debian platform specifics.

2.1.7 Setup

Attention: If you're running Isomer via Docker, please note that you have to run the setup commands inside the docker container. See *the details for that*.

Attention: If you're working with a virtual environment, do not forget to activate it first!

Modules

You can install modules from local sources or github right now.

Installation

```
iso -e current instance install-module -i -s git URL
```

Frontend rebuild

After installing a module, you will have to rebuild the frontend:

```
iso -e current environment install-frontend
```

Note: We're actively trying to eliminate this step, but currently it is not avoidable.

Admin Account

You can add a new admin user via:

```
iso db user create-admin
```

There is more *documentation about creating admins and users in the manual section*.

2.2 Isomer User's Manual

Welcome to the Isomer Users Manual! This part of the documentation explains how to work with Isomer and use the core modules.

Attention: Sadly, there is not much content, yet. *Do you want to help out?*

2.3 Isomer Administrator's Manual

Welcome to the Isomer Administrator's Manual! This part of the documentation explains how to administrate Isomer instances.

2.3.1 Command Line Tool

Isomer provides a comprehensive CLI tool to manage Isomer instances:

2.3.2 Module setup

Without installing or having any pre-installed modules, Isomer will not offer much functionality.

Instance module installation

To install a module into your active default environment, use e.g.:

```
iso -e current instance install-module -i -s github https://github.
com/isomeric/isomer-enrol
```

It is also possible to install a module you already downloaded:

```
iso -e current instance install-module -i -s copy path/to/repo
```

Attention: Due to technical issues, you will need to rebuild the frontend for any environment with newly installed modules. This will be removed in future.

2.3.3 User accounts

Without any accounts, you won't be able to use Isomer's frontend unless you have the isomer-enrol module module installed and configured to accept self-registrations.

Note: The isomer-enrol module provides methods for user self registration and administration in the frontend. It also provides password change functionality and other (customizable) user account infrastructure.

Normal users

Normal users can use most of the functionality, but not change any vital system parameters. Some functionality maybe restricted by the *Role Based Access Control* system, so you may need to adjust roles, as well.

You can add a new user via:

```
iso db user create-user
```

It is also possible to provide the username on the command line:

```
iso db user --username myuser create-user
```

It will ask for a password, but you can supply this via:

```
iso db user --username myuser --password mypass create-user
```

Admin Account

You can add a new admin user via:

```
iso db user create-admin
```

The arguments for `iso db user` will be used.

2.3.4 Role Based Access Control

From Wikipedia : Role based access control is an approach to restricting system access to authorized users.

Isomer implements RBAC on two levels:

- Object access control for persistent objects
- Event access control for component or user interface fired events

Note: This documentation is work in progress.

Access control

Object RBAC

Event RBAC

Users and Roles

2.3.5 Exit error code directory

The management tool provides exit error codes when operations failed. To help you understand and fix the problem at hand, here is a directory of all known error codes:

Attention: This is work in progress. Most codes are not yet populated. See #50020 for an Example

Errorcode: 50020

Please refer to https://isomer.readthedocs.io/en/latest/manual/Administration/Errors/50020.html for latest information about this problem.

Message

50020: `No database is available`

Symptoms

- Isomer takes a few seconds to launch, then exits with this error message
- Not much other log output is visible
- Some management tool commands may fail in similar ways

Remedies

- Check if the database is actually running
- Check if you supplied the correct hostname and port via `iso instance info | grep database`
- Check if the correct hostname and port are picked up by isomer via `iso launch --no-run` - they should be listed on one of the first lines in the commands output

CHAPTER 3

Developer Documentation

3.1 Developer Documentation

Here you find documentation on core concepts, general mechanisms, design choices but also the hard facts gathered from inline documentation strings or (soon) the automated API doc collector.

3.1.1 How to help the project?

Glad to see you're interested in helping out the project!

Generally, you can ping riot if you want to help out and don't exactly know where to start.

Here, we list a few possible opportunities where you can help us and become part of the driving community:

Communication

People need to be more aware of this project as it may be of great value to them. If you're interested in spreading the word and getting people involved, you're very welcome to do so. Again, please ping riot to get crucial info on how to do so.

Testing

There are various degrees to which you can test the project:

- Check the installation processes if they actually work on your platform and everything installs smoothly
- Test-drive your installation or the demo instance (Currently offline for maintenance)
- Build and extend parts of the automatic testing infrastructure
- Optimize and extend the continuous integration infrastructure

User Experience

We'd value your input on some very important user experience questions:

- Is the current design logical and does it allow for a smooth user experience?

- Check the supplied modules and the framework itself for consistency and good UX practices
- Develop further use cases and user stories to spark new modules

Documentation

A lot of documentation is still missing. If you're interested in writing documentation, you should be familiar with the two core tools we use for generating our documentation:

- reStructured Text formatting
- Sphinx

We still need a lot of module, core framework and source code documentation, so there's ample opportunities in this field.

Translations

Most of (if not all) parts of the project can be translated and are waiting for your help. You can use Transifex to translate all the strings we have or work with your favourite PO Editor. Have a look at *Translating Isomer* for more details.

3.1.2 Developer Guidelines

This is the rather dry material for new software developers:

Development Introduction

Here's how we do things in Isomer. . .

If you're looking for instructions on how to set up a development environment, please check out *the workflow documentation*.

Communication

- #hackerfleet IRC Channel on the FreeNode IRC Network
- Issue Tracker located at *https://github.com/isomeric/isomer/issues*

Note: If you are familiar with IRC and use your own IRC Client then connect to the FreeNode Network and `/join #hackerfleet`.

Standards

We use the following coding standard:

- PEP-008

We also lint our codebase with the following tools:

- pyflakes
- pep8
- mccabe

Please ensure your Development IDE or Editor has the above linters and checkers in place and enabled.

Alternatively you can use the following command line tool:

- flake8

Tools

We use the following tools to develop Isomer and share code:

- **Code Sharing:** Git
- **Code Hosting and Bug Reporting:** GitHub
- **Issue Tracker:** Issue Tracker
- **Documentation Hosting:** Read the Docs
- **Package Hosting:** Python Package Index (PyPi)
- **Docker Hub Automated Builds:** Dockerhub
- **Continuous Integration:** Travis CI
- **Code Quality:** Landscape
- **Frontend Testing:** Browserstack
- **Translations:** Transifex

We strongly suggest familiarizing with all of them, to make sure you understand our CI.

Big thanks to all of these magnificent and free-for-opensource services!

Contributing to Isomer

Here's how you can contribute to Isomer

Submitting Bug Reports

We welcome all bug reports. We do however prefer bug reports in a clear and concise form with repeatable steps. One of the best ways you can report a bug to us is by writing a unit test (//similar to the ones in our tests//) so that we can verify the bug, fix it and commit the fix along with the test.

To submit a bug report, please Create an Issue

Writing new tests

We're not perfect, and we're still writing more tests to ensure quality code. If you'd like to help, please Fork Isomer, write more tests that cover more of our code base and submit a Pull Request. Many Thanks!

Adding New Features

If you'd like to see a new feature added to Isomer, then we'd like to hear about it~ We would like to see some discussion around any new features as well as valid use-cases. To start the discussions off, please either:

- Chat with us on #hackerfleet on the FreeNode IRC Network

  or

- Create an Issue

Writing Documentation

We'd love to get your assistance and help with writing and translating documentation. We use sphinx which integrates nicely into the Isomer project concepts, but you don't necessarily need to delve into it that deep.

Even writing descriptive/explanatory texts and maybe supplying screenshots for functionality is welcome.

Note: We plan to automate the screenshot functionality, so updating docs is less work.

Setting up a Isomer Development Environment

This is the recommended way to setup a development environment for developing the backend, frontend and modules of Isomer.

Getting Started

Here is a summary of the steps to your own development environment:

1. Fork Isomer (*if you haven't done so already*)
2. Clone your forked repository using Git
3. Install the local management tool
4. Install an Isomer development instance
5. Set up further development tools as desired

And you're done!

Setup

Attention: This part needs an overhaul, as it pretty much details the standard instance-base installation approach. This can be avoided by working with simple plain virtual environments and a few of the iso tool install commands.

The setup guide shall aid you in setting up a development environment for all purposes and facettes of Isomer development. It is split up in a few parts and a common basic installation.

Get the sourcecode

After forking the repository, clone it to your local machine:

```
git clone git@github.com:yourgithubaccount/isomer.git ~/src/isomer
```

Setting up a basic development Instance

First install the management tool:

```
cd ~/src/isomer
./iso
```

This installs basic dependencies and installs the iso tool into your path. Now, use it to set up system directories and users:

```
iso system all
```

In theory, doing all steps is not required, but for safe measure, you should probably at least run the dependency and path setup:

```
iso system dependencies
iso system paths
```

Create a new development instance (ignore the warning about a missing default instance):

```
iso -i development instance create
```

Install the development instance from your repository clone:

```
iso -i development install -s copy -u ~/src/isomer
```

Tip: You can use arguments like *–skip-frontend* to skip over various processes of the installation, if you intend to modify the installation by e.g. hand-installing a development module before these steps are applied.

Activate the newly installed environment:

```
iso -i development turnover
```

Frontend Development

Change to frontend directory:

```
cd /var/lib/development/green/repository/frontend
```

and run the development webserver:

```
npm run start
```

Now you can launch the frontend in your browser by going to http://localhost:8081 To use other ports, either edit the webpack.config.js file or launch the dev server directly:

```
./node_modules/.bin/webpack-dev-server --host localhost --port 8888
```

Danger: Do not use the development server in production!

Module Development

Activate environment:

```
source /lib/isomer/development/green/venv/bin/activate
```

Install module for development:

```
cd ~/src/isomer-module
python setup.py develop
```

Currently, you'll need to restart (and possibly rebuild your frontend) your instance to run with changes.

General Development

Stop instance if started via system service:

```
systemctl stop isomer-development
```

Tip: You can run production instances parallel to a development instance by configuring it as another instance and changing its web-port. See *Running parallel instances* for more information on that. If you only want to run it with a development webserver, this is not necessary.

Restart instance in console mode:

```
cd /var/lib/isomer/development/green

source ./venv/bin/activate

iso --instance development --environment green --clog 10 launch
```

You should now see the startup process of your development instance log its messages to your terminal.

Tip: By typing */help* + return on that console, you can read about the offered interactive command line commands.

Development Processes

We document all our internal development processes here so you know exactly how we work and what to expect. If you find any issues or problems, please let us know!

Software Development Life Cycle (SDLC)

We employ the use of the SCRUM Agile Process and use our Issue Tracker to track features, bugs, chores and releases. If you wish to contribute to Isomer, please familiarize yourself with SCRUM and GitHub's Issue Tracker.

Bug Reports

- New Bug Reports are submitted via: https://github.com/isomeric/isomer/issues
- Confirmation and Discussion of all New Bug Reports.
- Once confirmed, a new Bug is raised in our Issue Tracker
- An appropriate milestone will be set (*depending on current milestone's schedule and resources*)
- A unit test developed that demonstrates the bug's failure.
- A fix developed that passes the unit test and breaks no others.
- A New Pull Request created with the fix.

  This should contain:

  – A new or modified unit test.

  – A patch that fixes the bug ensuring all unit tests pass.

  – The Change Log updated.

  – Appropriate documentation updated.
- The Pull Request is reviewed and approved by at least two other developers.

Feature Requests

- New Feature Requests are submitted via: https://github.com/isomeric/isomer/issues
- Confirmation and Discussion of all New Feature Requests.
- Once confirmed, a new Feature is raised in our Issue Tracker
- An appropriate milestone will be set (*depending on current milestone's schedule and resources*)
- A unit test developed that demonstrates the new feature.
- The new feature developed that passes the unit test and breaks no others.
- A New Pull Request created with the fix.

  This must contain:

  - A new or modified unit test.
  - A patch that implements the new feature ensuring all unit tests pass.
  - The Change Log updated.
  - Appropriate documentation updated.
- The Pull Request is reviewed and approved by at least two other developers.

Writing new Code

- Submit a New Issue
- Write your code.
- Use flake8 to ensure code quality.
- Run the tests:

  ```
  tox
  ```
- Ensure any new or modified code does not break existing unit tests.
- Update any relevant doc strings or documentation.
- Update the Change Log appropriately.
- Submit a New Pull Request.

Running the Tests

To run the tests you will need the following installed:

- tox installed as well as
- pytest-cov
- pytest

All of these can be installed via `pip install -r requirements-dev.txt`.

Please also ensure - if you can - that you you have all supported versions of Python that Isomer supports installed in your local environment.

To run the tests:

```
tox
```

Development Standards

We aim for the following development standards:

Cyclomatic Complexity

- Code Complexity shall not exceed `10`

  See: Limiting Cyclomatic Complexity

Coding Style

Note: We do accept "black" formatting.

- Code shall confirm to the PEP8 Style Guide.
- Doc Strings shall confirm to the PEP257 Convention.

Note: Arguments, Keyword Arguments, Return and Exceptions must be documented with the appropriate Sphinx Python Domain.

Revision History

- Commits shall be small tangible pieces of work. - Each commit must be concise and manageable. - Large changes are to be done over smaller commits.
- There shall be no commit squashing.
- Rebase your changes as often as you can.

Unit Tests

- Every new feature and bug fix must be accompanied with a unit test. (*The only exception to this are minor trivial changes*).

Translating Isomer

Since 2018, we have all parts (Backend, Frontend, Modules) prepared for translations.

To translate Isomer, you can use Transifex or any PO editor of your choice.

3.1.3 System Structure

Domains

Backend Overview

Lorem Ipsum!

Modularity

Modules

Isomer modules are software packages to extend your installation's functionality. They usually (but not always!) consist of:

- components
- events
- schemata
- provisions
- tool handlers
- frontend
- documentation

Components

Components are the logic part of a module.

Events

Events are used to communicate requests between components.

Anonymous Events

These are client side events without any authorization or identification attached.

Authorized Events

After clients have logged in, they have access to a broad selection of so called "authorized events". They have permissions and roles attached. See RBAC'

Internal Events

Console Events

Schemata

Schemata are used to specify (persistent) data structures and how they get represented via forms.

Provisions

Provisions are used when a module brings in additional data in the form of persistent objects.

Tool handlers

To allow maintenance, modules can register tool commands. These are available via the module section:

```
iso module <command>
```

To get a list of all modules' commands, just do:

```
iso module
```

Web Client Mechanics

The ClientManager handles web clients in cooperation with the WebSocket. All client and user requests run through the ClientManager.

Legitimate requests are fired off to their according request managers.

It delegates authentication requests separately to the Auth Component.

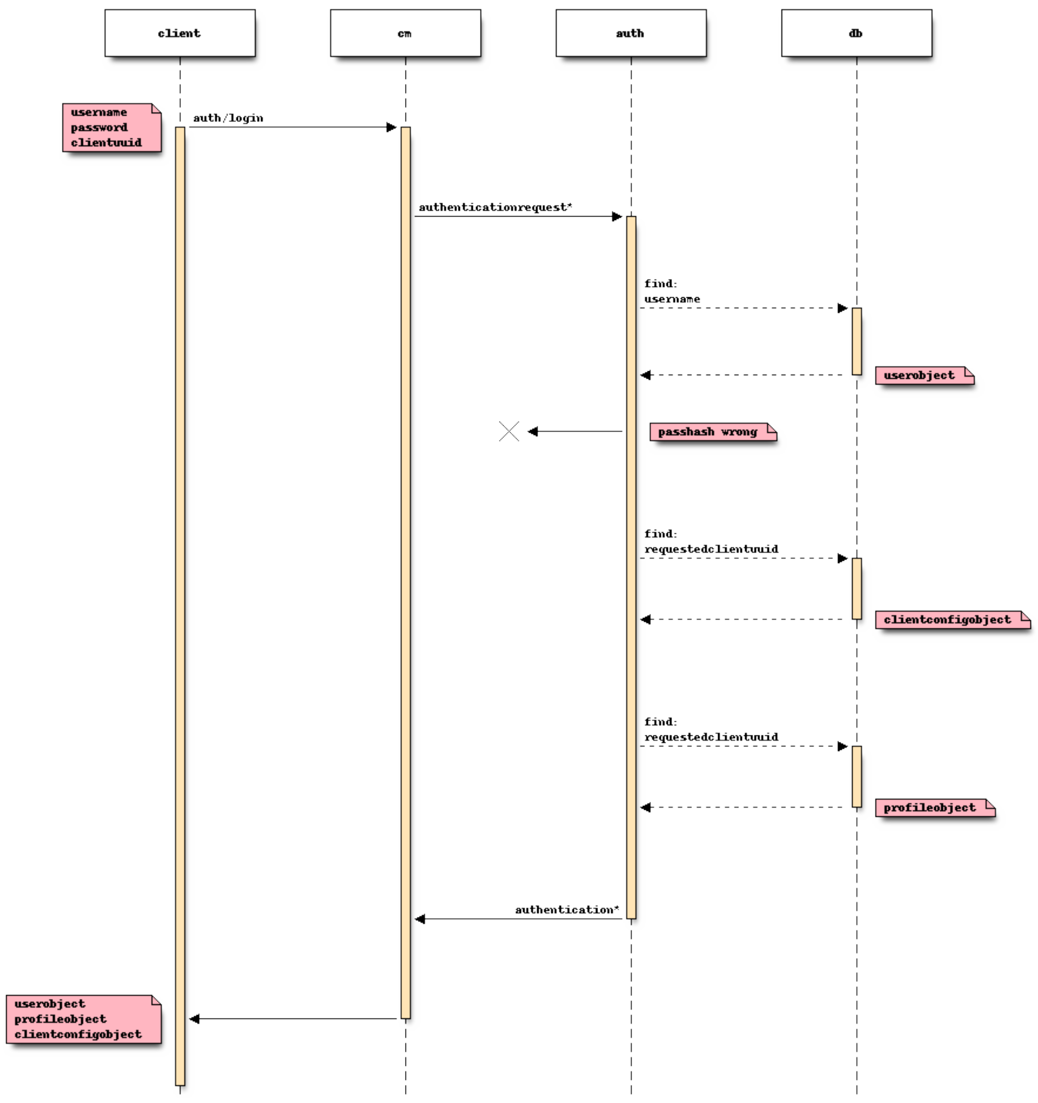
Frontend Overview

The frontend is built with Angular.js.

Concepts & Mechanisms

Authentication

Here, have a sequence diagram:



Docker

As Docker allows easy deployment and usage of Isomer on many platforms, we provide ready-to-use images on a (currently) manual basis.

Setup

The simplest way to get Isomer and a suitable database running is to run the docker compose file:

```
docker-compose -f docker/docker-compose.yml up
```

This should grab all necessary software and spin up two machines, one containing the database server and one with your Isomer instance.

Running the iso tool

To run the command via Docker compose:

```
docker-compose -f docker/docker-compose.yml run isomer iso db user
```

To run the iso tool inside your docker container without database access, just use Docker's run command, e.g:

```
docker -i -t isomeric/isomer:latest run iso system status
```

To work with the database, you need to provide it an accessible server address:

```
docker -i -t isomeric/isomer:latest run iso --dbhost mydatabasehost:27017
```

Note: Most of the command line options can also be supplied as environment variable, e.g. `export ISOMER_LAUNCH_WEBADDRESS=0.0.0.0`

Platforms

We provide amd64 and arm64 images built via buildkit and Docker's buildx command.

Publishing

Currently, we publish Docker images by hand, as building arm images on Docker- Hub is not yet easily possible without hacks. This will change, as indicated in their bugtracker.

Instances

Isomer runs so called instances to provide services to users.

Instance configuration and maintenance is handled by the *iso instance* command group.

You can also edit the instance configuration in */etc* by hand, but this is not recommended.

If you do so, please validate the configuration after editing.

What is an Instance?

An Isomer instance consists of two essential pieces:

- Metadata Configuration
- Environment system

Metadata Configuration

- Lives in /etc/isomer/instances/<instancename>.conf
- Used to define properties like * General meta data (e.g. contact, name) * Database connectivity * Web interface settings (e.g. hostname, port, certificate) * System user configuration * Installed components and sources

Environment System

An instance's actual software and aggregated things like user-uploaded data resides in so called Environments.

Usually, an instance has at least two environments, a blue and a green one. Next to these 'production' instances is the archive of older environments that gets extended every time the instance is upgraded.

One of the driving factors behind this process is the required stability when using Isomer in situations where software upgrades could potentially break an instance without any means of repair available. The implemented blue/green process allows downgrading to the earlier, uncom- promised state.

Explanation of the Blue/Green process

- Instances have two default environments: * green * blue
- Only one environment per instance can be actively used at a time
- Upgrade/downgrade or backup restore actions will always be performed on the non active environment
- Both environments will exist in parallel for diagnostics and testing but control will be handed over to the newly installed environment for testing itself
- On success (and perhaps a confirmation of an administrator), the active environment is turned over to the running one
- Furthermore, the old archive gets updated and cleared out in preparation for another
- On errors (or perhaps a cancellation request of an administrator), the newly set up environment gets shut down, cleared and the old (working) environment gets activated and started again

Parallel Instances

To allow running multiple Isomer systems on a single machine, multiple instances can be set up to run in parallel.

Provisions

These are partially external data sources like URLs.

Schemata

Schemata are used to validate and store objects across backend and frontend. They are used as document definitions for Formal which acts as a kind of ORM system.

They are also used by the frontend to generate forms and validate user input.

3.1.4 API Documentation

Attention: Sadly, the API documentation is work in progress.

3.1.5 isomer package

Package Isomer

The backend package.

This is a namespace package.

Subpackages

isomer.database package

Submodules

isomer.database.backup module

isomer.database.components module

isomer.database.profiling module

isomer.events package

Isomer Event objects

Submodules

isomer.events.client module

Isomer Client events

class authentication(*username*, *userdata*, *clientuuid*, *useruuid*, *sock*, **args*)
  Bases: `circuits.core.events.Event`

  Authentication has been granted to a client

  __init__(*username*, *userdata*, *clientuuid*, *useruuid*, *sock*, **args*)

  Parameters

  - **username** – Account username
  - **userdata** – Tuple containing both useraccount and userprofile
  - **uuid** – Unique User ID of known connection
  - **sock** – Associated Socket
  - **args** – Further Args

class authenticationrequest(*username*, *password*, *clientuuid*, *requestedclientuuid*, *sock*, *auto*, **args*)
  Bases: `circuits.core.events.Event`

  A client wants to authenticate a client connection

  __init__(*username*, *password*, *clientuuid*, *requestedclientuuid*, *sock*, *auto*, **args*)

  Parameters

  - **username** – Account username
  - **password** – Account md5 hash
  - **clientuuid** – Unique User ID of known connection
  - **sock** – Associated Socket
  - **args** – Further Args

class broadcast(*broadcasttype*, *content*, **args*)
  Bases: `circuits.core.events.Event`

  Send a packet to a known client by UUID

__init__(*broadcasttype*, *content*, **args*)

Parameters

- **uuid** – Unique User ID of known connection
- **packet** – Data packet to transmit to client
- **args** – Further Args

class clientdisconnect(*clientuuid*, *useruuid=None*, **args*)
Bases: `circuits.core.events.Event`

A client has disconnected from the system. This has to propagate to all subscription based and other user aware components.

Parameters

- **clientuuid** – UUID of disconnecting client
- **useruuid** – UUID of disconnecting user
- **args** –

__init__(*clientuuid*, *useruuid=None*, **args*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.
- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **success** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **complete** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

class send(*uuid*, *packet*, *sendtype='client'*, *raw=False*, *username=None*, *fail_quiet=False*, **args*)
Bases: `circuits.core.events.Event`

Send a packet to a known client by UUID

`__init__`(*uuid*, *packet*, *sendtype='client'*, *raw=False*, *username=None*, *fail_quiet=False*, **args*)

Parameters

- **`uuid`** – Unique User ID of known connection
- **`packet`** – Data packet to transmit to client
- **`args`** – Further Args

class `userlogin`(*clientuuid*, *useruuid*, *client*, *user*, **args*)

Bases: `circuits.core.events.Event`

A user has logged in to the system. This has to propagate to all subscription based and other user aware components.

Parameters

- **`clientuuid`** – UUID of disconnecting client
- **`useruuid`** – UUID of disconnecting user
- **`args`** –

`__init__`(*clientuuid*, *useruuid*, *client*, *user*, **args*)

An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **`channels`** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.
- **`value`** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **`success`** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **`success_channels`** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **`complete`** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **`complete_channels`** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

class `userlogout`(*useruuid*, *clientuuid*, **args*)

Bases: `circuits.core.events.Event`

A user has logged out from the system. This has to propagate to all subscription based and other user aware components.

Parameters

- **clientuuid** – UUID of disconnecting client
- **useruuid** – UUID of disconnecting user
- **args** –

`__init__`(*useruuid*, *clientuuid*, **args*)

An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.

- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **success** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **complete** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

isomer.events.objectmanager module

Module: Events

Major Isomer event declarations

class add_role(*user*, *action*, *data*, *client*, **args*)

Bases: *`isomer.events.system.authorized_event`*

class change(*user*, *action*, *data*, *client*, **args*)

Bases: *`isomer.events.system.authorized_event`*

A client requires a schema to validate data or display a form

class delete(*user*, *action*, *data*, *client*, **args*)

Bases: *`isomer.events.system.authorized_event`*

A client requires a schema to validate data or display a form

class get (*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

A client requires a schema to validate data or display a form

class getlist (*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

A client requires a schema to validate data or display a form

class objectchange (*uuid*, *schema*, *client*, **args*, ***kwargs*)
Bases: *isomer.events.objectmanager.objectevent*

A stored object has been successfully modified

class objectcreation (*uuid*, *schema*, *client*, **args*, ***kwargs*)
Bases: *isomer.events.objectmanager.objectevent*

A new object has been successfully created

class objectdeletion (*uuid*, *schema*, *client*, **args*, ***kwargs*)
Bases: *isomer.events.objectmanager.objectevent*

A stored object has been successfully deleted

class objectevent (*uuid*, *schema*, *client*, **args*, ***kwargs*)
Bases: `circuits.core.events.Event`

A unspecified objectevent

__init__ (*uuid*, *schema*, *client*, **args*, ***kwargs*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.
- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **success** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **complete** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.

- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

class put(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

A client requires a schema to validate data or display a form

class remove_role(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

class search(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

A client requires a schema to validate data or display a form

class subscribe(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

A client requires a schema to validate data or display a form

class unsubscribe(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

A client requires a schema to validate data or display a form

class updatesubscriptions(*schema*, *data*, **args*, ***kwargs*)
Bases: `circuits.core.events.Event`

A backend component needs to write changes to an object. Clients that are subscribed should be notified etc.

__init__(*schema*, *data*, **args*, ***kwargs*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.

- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.

- **success** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.

- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

- **complete** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.

- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

isomer.events.schemamanager module

Module: Events.Schemamanager

Major Isomer event declarations

class all(*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

A client requires a schema to validate data or display a form

class configuration(*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

A client requires a schema to validate data or display a form

class get(*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

A client requires a schema to validate data or display a form

isomer.events.system module

Module: Events

Major Isomer event declarations

class anonymous_event(*action*, *data*, *client*, **args*)
Bases: *isomer.events.system.isomer_ui_event*

Base class for events for logged in users.

__init__(*action*, *data*, *client*, **args*)
Initializes an Isomer anonymous user interface event.

Parameters

- **action** –
- **data** –
- **client** –
- **args** –

Returns

classmethod realname()
Return real name of an object class

class authorized_event(*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.isomer_ui_event*

Base class for events for logged in users.

__init__(*user*, *action*, *data*, *client*, **args*)
Initializes an Isomer authorized user interface event.

Parameters

- **user** – User object from :py:class:isomer.web.clientmanager.User

- **action** –
- **data** –
- **client** –
- **args** –

Returns

classmethod realname()
Return real name of an object class

roles = ['admin', 'crew']

class componentupdaterequest(*force=False*, *install=False*, **args*)
Bases: `isomer.events.system.frontendbuildrequest`

Check for updated components

class debugrequest(**args*)
Bases: `isomer.events.system.authorized_event`

Debugging event

__init__(**args*)
Initializes an Isomer authorized user interface event.

Parameters

- **user** – User object from :py:class:isomer.web.clientmanager.User
- **action** –
- **data** –
- **client** –
- **args** –

Returns

class frontendbuildrequest(*force=False*, *install=False*, **args*)
Bases: `circuits.core.events.Event`

Rebuild and/or install the frontend

__init__(*force=False*, *install=False*, **args*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.

- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.

- **success** – if this optional attribute is set to True, an associated event success (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **complete** – if this optional attribute is set to True, an associated event complete (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

get_anonymous_events()
Return all registered anonymous events

get_user_events()
Return all registered authorized events

class isomer_basic_event(**args*, ***kwargs*)
Bases: circuits.core.events.Event

Basic Isomer event class

__init__(**args*, ***kwargs*)
Initializes a basic Isomer event.

For further details, check out the circuits documentation.

class isomer_event(**args*, ***kwargs*)
Bases: *isomer.events.system.isomer_basic_event*

Isomer internal event class

class isomer_ui_event(**args*, ***kwargs*)
Bases: *isomer.events.system.isomer_basic_event*

Isomer user interface event class

class logtailrequest(*user*, *action*, *data*, *client*, **args*)
Bases: *isomer.events.system.authorized_event*

Request the logger's latest output

populate_user_events()
Generate a list of all registered authorized and anonymous events

class profilerequest(**args*)
Bases: *isomer.events.system.authorized_event*

A user has changed his profile

__init__(**args*)

Parameters

- **user** – Userobject of client
- **data** – The new profile data

class reload_configuration(*target*, **args*, ***kwargs*)
Bases: *isomer.events.system.isomer_ui_event*

Instructs a component to reload its configuration

`__init__`(*target*, **args*, ***kwargs*)
Initializes a basic Isomer event.

For further details, check out the circuits documentation.

isomer.misc package

Submodules

isomer.misc.path module

isomer.provisions package

Package: Provisions

Initial client configuration data. This contains tilelayer urls, api stuff etc.

`build_provision_store`()

Submodules

isomer.provisions.base module

Provisioning: Basic Functionality

Contains

Basic functionality around provisioning.

`log`(**args*, ***kwargs*)
Log as Emitter:MANAGE

`provision`(*list_provisions=False*, *overwrite=False*, *clear_provisions=False*, *package=None*, *installed=None*)

`provisionList`(*items*, *database_name*, *overwrite=False*, *clear=False*, *skip_user_check=False*)
Provisions a list of items according to their schema

Parameters

- **`items`** – A list of provisionable items.
- **`database_name`** –
- **`overwrite`** (`bool`) – Causes existing items to be overwritten
- **`clear`** (`bool`) – Clears the collection first (Danger!)
- **`skip_user_check`** (`bool`) – Skips checking if a system user is existing already (for user provisioning)

Returns

isomer.provisions.system module

isomer.provisions.user module

Provisioning: User

Contains

Just creates a fulltext searchable index over the username field.

provision_system_user (*items*, *database_name*, *overwrite=False*, *clear=False*, *skip_user_check=False*)
Provisions a system user

isomer.schemata package

Module Schemata

All JSONSchema compliant data schemata for Isomer

Contains

profile.py: User profile objects user.py: User account objects

Submodules

isomer.schemata.base module

isomer.schemata.client module

isomer.schemata.component module

Basic component configuration schemata template

isomer.schemata.defaultform module

area_field (*key='area'*)
Provides a select box for country selection

country_field (*key='country'*)
Provides a select box for country selection

create_object (*key*, *lookup_type*)
Returns a lookup button to inspect a selected object

emptyArray (*key*, *add_label=None*)
An array that starts empty

event_button (*key*, *title*, *target*, *action*, *data=None*)
Template for an event emitting button

fieldset (*title*, *items*, *options=None*)
A field set with a title and sub items

horizontal_divider ()
Inserts a horizontal ruler/divider

lookup_field (*key*, *lookup_type=None*, *placeholder=None*, *html_class='div'*, *select_type='strapselect'*, *mapping='uuid'*, *search_filter=None*)
Generates a lookup field for form definitions

lookup_field_multiple (*key*, *subkey=None*, *button='Add'*, *lookup_type=None*, *placeholder=None*, *html_class='div'*, *select_type='strapselect'*, *mapping='uuid'*)

lookup_object (*key*, *lookup_type=None*, *actions=None*)
: Returns a lookup button to inspect a selected object

rating_widget (*key='rating'*, *maximum=10*)
: A customizable star rating widget

section (*rows*, *columns*, *items*, *label=None*, *condition=None*)
: A section consisting of rows and columns

tabset (*titles*, *contents*)
: A tabbed container widget

test ()
: Development function to manually test all widgets

isomer.schemata.extends module

DefaultExtension (*schema_obj*, *form_obj*, *schemata=None*)
: Create a default field

isomer.schemata.geometry module

Schema: Geometry

This is non-model schema for integration into other schemata.

Contains

geometry: Any valid GeoJSON geometry (Points, Multipoints, Linestrings, Multilinestrings, Polygons and Multipolygons)

isomer.schemata.logmessage module

isomer.schemata.profile module

isomer.schemata.system module

isomer.schemata.tag module

isomer.schemata.user module

isomer.tool package

Package: Tool

Contains basic functionality for the isomer management tool.

Command groups

backup configuration create_module database defaults dev environment etc installer instance misc objects rbac remote system user

General binding glue

cli templates tool

ask (*question, default=None, data_type='str', show_hint=False*)
  Interactively ask the user for data

ask_password ()
  Securely and interactively ask for a password

check_root ()
  Check if current user has root permissions

finish (*ctx*)

format_result (*result*)
  Format child instance output

get_isomer (*source, url, destination, upgrade=False, shell=None, sudo=None*)
  Grab a copy of Isomer somehow

get_next_environment (*ctx*)
  Return the next environment

install_isomer (*platform_name=None, use_sudo=False, shell=None, cwd='.', show=False, omit_common=False, omit_platform=False*)
  Installs all dependencies

log (**args, **kwargs*)
  Log as Emitter:MANAGE

run_process (*cwd, args, shell=None, sudo=None, show=False, stdout=None, stdin=None, timeout=5*)
  Executes an external process via subprocess.check_output

Submodules

isomer.tool.backup module

isomer.tool.cli module

isomer.tool.configuration module

Module: Configuration

Instance component configuration management.

get_configuration (*col, component*)
  Get a configuration via name or uuid

isomer.tool.database module

Module: Database

Database management functionality.

delete_database (*db_host, db_name, force*)
  Actually delete a database

isomer.tool.defaults module

Module: Defaults

Isomer distribution default settings.

Contains database setup, certificate locations, platform details, service templates and a table of exit codes for the management tool.

isomer.tool.dev module

isomer.tool.environment module

isomer.tool.etc module

isomer.tool.installer module

isomer.tool.instance module

isomer.tool.misc module

isomer.tool.objects module

Module: Objects

Object management functionality and utilities.

isomer.tool.rbac module

Module: RBAC

Role based access control management functionality.

isomer.tool.remote module

isomer.tool.system module

isomer.tool.templates module

Module: Templates

Internal template handling utilities.

format_template (*template*, *content*)
Render a given pystache template with given content

format_template_file (*filename*, *content*)
Render a given pystache template file with given content

insert_nginx_service (*definition*)
Insert a new nginx service definition

write_template (*template*, *target*, *content*)
Write a new file from a given pystache template file and content

write_template_file (*source*, *target*, *content*)
Write a new file from a given pystache template file and content

isomer.tool.tool module

isomer.tool.user module

isomer.ui package

Web bits

Package Isomer.ui

The backend package dealing with the user interface.

Subpackages

isomer.ui.clientmanager package

Submodules

isomer.ui.clientmanager.authentication module

isomer.ui.clientmanager.basemanager module

isomer.ui.clientmanager.cli module

isomer.ui.clientmanager.encoder module

isomer.ui.clientmanager.floodprotection module

isomer.ui.clientmanager.languages module

isomer.ui.clientmanager.latency module

isomer.ui.objectmanager package

Submodules

isomer.ui.objectmanager.basemanager module

isomer.ui.objectmanager.cli module

isomer.ui.objectmanager.crud module

isomer.ui.objectmanager.roles module

isomer.ui.objectmanager.subscriptions module

Submodules

isomer.ui.activitymonitor module

Module: ActivityMonitor

Surveillance piece to check out what the users are doing, so the system can react accordingly (e.g. not disturb with unimportant alerts when user is actively doing something)

Possibilities: * check if users noticed an alert * notify users, about what other users are doing * offer further information * achievements ;) (stared 100 hours at the map)

Should be user configurable and toggleable, at least most parts/bits.

class ActivityMonitor(**args*)

Bases: *isomer.component.ConfigurableComponent*

ActivityMonitor manager

Handles

- incoming ActivityMonitor messages
- ActivityMonitor broadcasts

__init__(**args*)

Check for configuration issues and instantiate a component

activityrequest(*event*)

ActivityMonitor event handler for incoming events

:param event with incoming ActivityMonitor message

channel = 'isomer-web'

referenceframe(*event*)

Handles navigational reference frame updates. These are necessary to assign geo coordinates to alerts and other misc things.

:param event with incoming referenceframe message

userlogin(*event*)

Checks if an alert is ongoing and alerts the newly connected client, if so.

isomer.ui.auth module

isomer.ui.builder module

isomer.ui.clientobjects module

isomer.ui.configurator module

isomer.ui.schemamanager module

isomer.ui.syslog module

class Syslog(**args*)

Bases: *isomer.component.ConfigurableComponent*

System log access component

Handles all the frontend log history requests.

`__init__`(**args*)
Check for configuration issues and instantiate a component

`disconnect`(*event*)

`follow`()

`history`(*event*)

`subscribe`(*event*)

`class history`(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

`class subscribe`(*user*, *action*, *data*, *client*, **args*)
Bases: `isomer.events.system.authorized_event`

isomer.ui.tagmanager module

Submodules

isomer.component module

Configurable Component

Contains

Systemwide configurable component definition. Stores configuration either in database or as json files. Enables editing of configuration through frontend.

See also

Provisions

`exception ComponentDisabled`
Bases: `Exception`

`class ConfigurableComponent`(*uniquename=None*, **args*, ***kwargs*)
Bases: `isomer.component.ConfigurableMeta`, `circuits.core.components.Component`

Configurable component for default Isomer modules

`__init__`(*uniquename=None*, **args*, ***kwargs*)
Check for configuration issues and instantiate a component

`class ConfigurableController`(*uniquename=None*, **args*, ***kwargs*)
Bases: `isomer.component.ConfigurableMeta`, `circuits.web.controllers.Controller`

Configurable controller for direct web access

`__init__`(*uniquename=None*, **args*, ***kwargs*)
Check for configuration issues and instantiate a component

`class ConfigurableMeta`(**args*, *no_db=False*, ***kwargs*)
Bases: `isomer.component.LoggingMeta`

Meta class to add configuration capabilities to circuits objects

`__init__`(**args*, *no_db=False*, ***kwargs*)
Check for configuration issues and instantiate a component

configform = []

configprops = {}

register(**args*)
Register a configurable component in the configuration schema store

reload_configuration(*event*)
Event triggered configuration reload

unregister()
Removes the unique name from the systems unique name list

class ExampleComponent(**args*, ***kwargs*)
Bases: `isomer.component.ConfigurableComponent`

Exemplary component to demonstrate basic component usage

__init__(**args*, ***kwargs*)
Show how the component initialization works and test this by adding a log statement.

configprops = {'setting': {'default': 'Yay', 'description': 'Some string setting

class LoggingComponent(*uniquename=None*, **args*, ***kwargs*)
Bases: `isomer.component.LoggingMeta`, `circuits.core.components.Component`

Logging capable component for simple Isomer components

__init__(*uniquename=None*, **args*, ***kwargs*)
Check for configuration issues and instantiate a component

class LoggingMeta(*uniquename=None*, **args*, ***kwargs*)
Bases: `object`

Baseclass for all components that adds naming and logging functionality

__init__(*uniquename=None*, **args*, ***kwargs*)
Check for configuration issues and instantiate a component

log(**args*, ***kwargs*)
Log a statement from this component

names = []

handler(**names*, ***kwargs*)
Creates an Event Handler

This decorator can be applied to methods of classes derived from `circuits.core.components.BaseComponent`. It marks the method as a handler for the events passed as arguments to the `@handler` decorator. The events are specified by their name.

The decorated method's arguments must match the arguments passed to the `circuits.core.events.Event` on creation. Optionally, the method may have an additional first argument named *event*. If declared, the event object that caused the handler to be invoked is assigned to it.

By default, the handler is invoked by the component's root `Manager` for events that are propagated on the channel determined by the BaseComponent's *channel* attribute. This may be overridden by specifying a different channel as a keyword parameter of the decorator (`channel=...`).

Keyword argument `priority` influences the order in which handlers for a specific event are invoked. The higher the priority, the earlier the handler is executed.

If you want to override a handler defined in a base class of your component, you must specify `override=True`, else your method becomes an additional handler for the event.

Return value

Normally, the results returned by the handlers for an event are simply collected in the `circuits.core.events.Event`'s `value` attribute. As a special case, a handler may return a `types.GeneratorType`.

This signals to the dispatcher that the handler isn't ready to deliver a result yet. Rather, it has interrupted it's execution with a `yield None` statement, thus preserving its current execution state.

The dispatcher saves the returned generator object as a task. All tasks are reexamined (i.e. their `next()` method is invoked) when the pending events have been executed.

This feature avoids an unnecessarily complicated chaining of event handlers. Imagine a handler A that needs the results from firing an event E in order to complete. Then without this feature, the final action of A would be to fire event E, and another handler for an event `SuccessE` would be required to complete handler A's operation, now having the result from invoking E available (actually it's even a bit more complicated).

Using this "suspend" feature, the handler simply fires event E and then yields `None` until e.g. it finds a result in E's `value` attribute. For the simplest scenario, there even is a utility method `circuits.core.manager.Manager.callEvent()` that combines firing and waiting.

isomer.debugger module

Module: Debugger

Debugger overlord

class CLI(**args*)

Bases: `isomer.component.ConfigurableComponent`

Command Line Interface support

This is disabled by default. To enable the command line interface, use either the Configuration frontend or the iso tool:

```
iso config enable CLI
```

__init__(**args*)
Check for configuration issues and instantiate a component

cli_help(**args*)

configprops = {}

register_event(*event*)
Registers a new command line interface event hook as command

stdin_read(*data*)
read Event (on channel `stdin`) This is the event handler for `read` events specifically from the `stdin` channel. This is triggered each time stdin has data that it has read.

class IsomerDebugger(*root=None, *args*)

Bases: `isomer.component.ConfigurableComponent`

Handles various debug requests.

__init__(*root=None, *args*)
Check for configuration issues and instantiate a component

channel = 'isomer-web'

cli_compgraph(*event*)

cli_errors(**args*)

cli_exception_test(**args*)

cli_locations(**args*)

cli_log_level(**args*)

cli_mem_diff(*event*)

cli_mem_growth(**args*)

cli_mem_heap(**args*)

cli_mem_hogs(**args*)

cli_mem_summary(*event*)

configprops = {'notificationusers': {'default': [], 'description': 'Users that sh

debug_store_json(*event*)

logtailrequest(*event*)

exception TestException
Bases: `BaseException`

Generic exception to test exception monitoring

class cli_comp_graph(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_errors(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Display errors in the live log

class cli_help(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Display this command reference

Additional arguments:

-v	Add detailed information about hook events in list

command Show complete documentation of a hook command

class cli_locations(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Display all locations of running instance

class cli_log_level(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Adjust log level

Argument: [int] New logging level (0-100)

class cli_mem_diff(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_mem_growth(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_mem_heap(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_mem_hogs(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_mem_summary(**args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Draw current component graph

class cli_register_event(*cmd*, *thing*, **args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Event to register new command line interface event hooks

__init__(*cmd*, *thing*, **args*, ***kwargs*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

  When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.
- **value** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **success** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **success_channels** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **complete** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **complete_channels** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

class clicommand(*cmd*, *cmdargs*, **args*, ***kwargs*)
Bases: `circuits.core.events.Event`

Event to execute previously registered CLI event hooks

__init__(*cmd*, *cmdargs*, **args*, ***kwargs*)
An event is a message send to one or more channels. It is eventually dispatched to all components that have handlers for one of the channels and the event type.

All normal arguments and keyword arguments passed to the constructor of an event are passed on to the handler. When declaring a handler, its argument list must therefore match the arguments used for creating the event.

Every event has a `name` attribute that is used for matching the event with the handlers.

Variables

- **channels** – an optional attribute that may be set before firing the event. If defined (usually as a class variable), the attribute specifies the channels that the event should be delivered to as a tuple. This overrides the default behavior of sending the event to the firing component's channel.

When an event is fired, the value in this attribute is replaced for the instance with the channels that the event is actually sent to. This information may be used e.g. when the event is passed as a parameter to a handler.

- **`value`** – this is a `circuits.core.values.Value` object that holds the results returned by the handlers invoked for the event.
- **`success`** – if this optional attribute is set to `True`, an associated event `success` (original name with "_success" appended) will automatically be fired when all handlers for the event have been invoked successfully.
- **`success_channels`** – the success event is, by default, delivered to same channels as the successfully dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.
- **`complete`** – if this optional attribute is set to `True`, an associated event `complete` (original name with "_complete" appended) will automatically be fired when all handlers for the event and all events fired by these handlers (recursively) have been invoked successfully.
- **`complete_channels`** – the complete event is, by default, delivered to same channels as the initially dispatched event itself. This may be overridden by specifying an alternative list of destinations using this attribute.

isomer.error module

Module: Error

Error handling

`abort`(*error_object*)

`log`(**args*, ***kwargs*)
Log as previous emitter

isomer.iso module

Isomer Management Tool

This is the management tool to install, configure and maintain Isomer instances.

`main`()
Try to load the tool and launch it. If it can't be loaded, try to install all required things first.

`warn`(**args*, ***kwargs*)

isomer.launcher module

isomer.logger module

Module: Logger

Isomer's own logger to avoid namespace clashes etc. Comes with some fancy functions.

Log Levels

verbose = 5 debug = 10 info = 20 warn = 30 error = 40 critical = 50 off = 100

`clear`()
: Clear the live log

`get_logfile`() → str
: Return the whole filename of the logfile

`is_marked`(*what*) → bool
: Check if log line qualifies for highlighting

`is_muted`(*what*) → bool
: Checks if a logged event is to be muted for debugging purposes.

  Also goes through the solo list - only items in there will be logged!

  Return type `bool`

  Parameters **`what`** –

`isolog`(**what*, ***kwargs*)
: Logs all non keyword arguments.

  Parameters

  - **`what`** (*`tuple/str`*) – Loggable objects (i.e. they have a string representation)
  - **`lvl`** (*`int`*) – Debug message level
  - **`emitter`** (*`str`*) – Optional log source, where this can't be determined automatically
  - **`sourceloc`** (*`str`*) – Give specific source code location hints, used internally
  - **`frameref`** (*`int`*) – Specify a non default frame for tracebacks
  - **`tb`** (*`bool`*) – Include a traceback
  - **`nc`** (*`bool`*) – Do not use color
  - **`exc`** (*`bool`*) – Switch to better handle exceptions, use if logging in an except clause

`set_color`()
: Activate colorful logging

`set_logfile`(*path: str*, *instance: str*, *filename: str = None*)
: Specify logfile path

  Parameters

  - **`path`** (`str`) – Path to the logfile
  - **`instance`** (`str`) – Name of the instance
  - **`filename`** (`str`) – Exact name of logfile

`set_verbosity`(*new_lvl: int*)
: Adjust logging verbosity

`setup_root`(*newroot: isomer.components.Component*)
: Sets up the root component, so the logger knows where to send logging signals.

  Parameters **`newroot`** (*`isomer.components.Component`*) –

isomer.migration module

Module: Migration

make_migrations(*schema=None*)
Create migration data for a specified schema

isomer.schemastore module

isomer.scm_version module

isomer.version module

Version Module

Unified Isomer wide version number.

3.2 Recent Changes

- **Frontend ref updated** by *ri0t* at *2019-12-19 17:11:09*
- **Frontend ref updated** by *ri0t* at *2019-12-19 16:41:36*
- **Reordered dependencies and switched to copy-mode** by *ri0t* at *2019-12-19 16:41:26*
- **Changed wording** by *ri0t* at *2019-12-19 15:39:34*
- **Frontend ref updated** by *ri0t* at *2019-12-19 01:26:25*
- **Dependencies optimized for docker image generation** by *ri0t* at *2019-12-19 01:26:03*
- **Safety is sadly a bit too confused to use it, right now** by *ri0t* at *2019-12-18 17:59:46*
- **Frontend ref updated** by *ri0t* at *2019-12-18 17:45:17*
- **Dockerfile updated** by *ri0t* at *2019-12-18 17:45:01*
- **Fixed readme headers** by *ri0t* at *2019-12-18 17:44:35*

3.3 Road Map

We manage our roadmap via milestones on our github issuetracker.

3.4 Contributors

The following users and developers have contributed to Isomer:

- Heiko 'riot' Weinen riot@c-base.org
- Johannes 'ijon' Rundfeldt ijon@c-base.org
- Martin Ling
- River 'anm' MacLeod
- Sascha 'c_ascha' Behrendt c_ascha@c-base.org
- You?

Isomer proudly uses the circuits framework. circuits was originally designed, written and primarily maintained by James Mills (http://prologic.shortcircuit.net.au/).

Anyone not listed here, ping us. We appreciate any and all contributions to Isomer and other Hackerfleet components.

3.5 Supporters

- **Initial project conception phase funding:** Kenny Bentley
- **Hosting and nix expertise:** Lassulus
- **Free OSS license of IntelliJ IDEA Ultimate:** Jetbrains
- **Repository and issue tracker hosting:** Github
- **Repository and issue tracker hosting:** GitLab
- **Free OSS cross platform/browser user interface testing:** Browserstack

3.6 Frequently Asked Questions

3.6.1 General

. . . What is Isomer? Isomer is an opensource collaborative application platform.

. . . What platforms does Isomer support? We currently test on Debian, various flavours of Python (3.3, 3.4, 3.5, 3.6, pypy) It'll probably run on various other platforms as well. E.g. we've made good experiences with Arch Linux.

Got more questions?

- Meet us and chat with us online on the #hackerfleet IRC Channel

Note: Please be patient when using IRC, responses might take a few hours!

3.7 Glossary

ESRI Esri (a.k.a. Environmental Systems Research Institute) is an international supplier of geographic information system (GIS) software, web GIS and geodatabase management applications. The company is headquartered in Redlands, California.

GDAL GDAL is a translator library for raster geospatial data formats. As a library, it presents a single abstract data model to the calling application for all supported formats. The related OGR library (which lives within the GDAL source tree) provides a similar capability for simple features vector data.

GDAL supports many popular data formats, including commonly used ones (*GeoTIFF*, *JPEG*, *PNG* and more) as well as the ones used in GIS and remote sensing software packages (ERDAS Imagine, *ESRI* Arc/Info, ENVI, PCI Geomatics). Also supported many remote sensing and scientific data distribution formats such as HDF, EOS FAST, *NOAA* L1B, NetCDF, FITS.

OGR library supports popular vector formats like ESRI Shapefile, TIGER data, *S-57*, MapInfo File, DGN, GML and more.

GeoTIFF GeoTIFF is a public domain metadata standard which allows georeferencing information to be embedded within a TIFF file.

The potential additional information includes map projection, coordinate systems, ellipsoids, datums, and everything else necessary to establish the exact spatial reference for the file.

GPS a navigational system involving satellites and computers that can determine the latitude and longitude of a receiver on Earth by computing the time difference for signals from different satellites to reach the receiver [syn: {Global Positioning System}, {GPS}]

JPEG JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality. JPEG is the most widely used image compression standard on the internet.

JSON In computing, JavaScript Object Notation or JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format used for asynchronous browser–server communication.

LDAP Lightweight Directory Access Protocol (RFC 1777, X.500, DS, AD, CORBA)

MQTT Message Queuing Telemetry Transport (ISO/IEC PRF 20922) is a publish- subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker.

NMEA National Marine Electronics Association [protocol] (org., USA, GPS), http://www.nmea.org

NOAA The National Oceanic and Atmospheric Administration is an American scientific agency within the United States Department of Commerce that focuses on the conditions of the oceans, major waterways, and the atmosphere.

PNG Portable Network Graphics is a raster-graphics file-format that supports lossless data compression. PNG was developed as an improved, non-patented replacement for Graphics Interchange Format (GIF).

Radar measuring instrument in which the echo of a pulse of microwave radiation is used to detect and locate distant objects [syn: {radar}, {microwave radar}, {radio detection and ranging}, {radiolocation}]

S-57 IHO S-57 / Electronic Nautical Charts (ENCs) - proprietary and often encrypted (see S-63) to prevent unauthorized distribution.

S-63 Encrypted versions of S-57 nautical charts

VCS Version Control System, what you use for versioning your source code

XMPP Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML (Extensible Markup Language). It enables the near-real-time exchange of structured yet extensible data between any two or more network entities.

Originally named Jabber, the protocol was developed by the Jabber open-source community in 1999 for near real-time instant messaging (IM), presence information, and contact list maintenance.

Designed to be extensible, the protocol has been used also for publish- subscribe systems, signalling for VoIP, video, file transfer, gaming, the Internet of Things (IoT) applications such as the smart grid, and social networking services.

CHAPTER 4

Indices and tables

- Index
- modindex
- search
- *Glossary*
- *Recent Changes*

Python Module Index

Index