paws Documentation

Release 0.4.0

Lenson A. Pellouchoud

Feb 21, 2017

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

The paws package aims to provide a fast and lean platform for building and executing workflows for data processing. It was originally developed to perform analysis of diffraction images for research purposes at SLAC/SSRL. At the core of paws is a workflow engine that uses a library of operations to crunch through data and expose select results while attempting to minimize resource consumption.

paws is currently written in Python, based on Qt via the PySide bindings. Internally, paws keeps track of data in Qt-based tree models, which can be controlled either directly (through the paws api) or through a gui (employing the Qt model-view framework).

paws also provides an interface to xi-cam, a synchrotron x-ray diffraction data analysis package written by the CAMERA Institute and Pandolfi, et al at the Lawrence Berkeley National Lab.

Some the core goals of paws:

- Eliminate redundant development efforts
- Streamline and standardize routine data analysis
- Simplify data storage and provide large-scale analysis
- Perform data analysis in real time for results-driven feedback

The paws developers would love to hear from you if you have wisdom, thoughts, haikus, bugs, artwork, or suggestions. Limericks are also welcome. Get in touch with us at paws-developers@slac.stanford.edu.

CHAPTER

TWO

QUICK START

Minimal and usually-effective installation instructions.

Here is a reference to the *brief introduction*.

This chapter is for setting up `paws` quickly in an environment that is prepared to install Python packages with `pip`.

CHAPTER

THREE

INSTALLATION

Here are instructions for installing `paws` from PyPI, or downloading and testing the `paws` source code.

Installing with pip

Instructions will go here for installing `paws` using the Python package installer `pip` (currently not implemented).

Downloading Source

The source code for `paws` is hosted on github. Clone the repository from `https://github.com/slaclab/paws.git`. You should then be able to run `paws` by invoking `python main.py` from the root directory.

Testing

`paws` comes with a tests that can be used to ensure the platform runs as expected. After *downloading the source*, invoke `python -m unittest discover` from the root directory.

CHAPTER

FOUR

API DOCUMENTATION

This is the complete auto-generated documentation of the `paws` package, made with sphinx-apidoc.

paws package

Subpackages

paws.api package

Module contents

Module defining the API for paws

`paws.api.`**`core_app`**(*app_args=[]*)
Return a reference to a new QCoreApplication or a currently running QApplication.

Input arguments are passed to the QApplication constructor. If a RuntimeError is thrown, it is assumed that a QApplication is already running, and an attempt is made to return a reference to that QApplication. If that fails, this returns None.

Parameters **`app_args`** – arguments to pass to the QApplication constructor

Returns reference to a new or existing QCoreApplication

Return type PySide.QtCore.QCoreApplication or None

`paws.api.`**`start`**(*app_args=[]*)
Instantiate an Operation Manager, a Workflow Manager, and a Plugin Manager. Return references to them.

paws.api.start() calls paws.api.core_app(), then sets up and returns references to a paws Workflow Manager (paws.api.workflow_manager), Operation Manager (paws.api.op_manager), and Plugin Manager (paws.api.plugin_manager).

Parameters **`app_args`** – arguments to pass to the QApplication constructor

Returns references to paws operation manager, workflow manager, and plugin manager

Return type tuple of paws.core.operations.op_manager.OpManager, paws.core.workflow.wf_manager.WfManager, paws.core.plugins.plugin_manager.PluginManager

Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p