Station Executive Documentation

Release 0.7.4

Oculus Purple Automation

May 31, 2019

CONTENTS

1 Station Executive **1**
- 1.1 Quick Start . . . 1
- 1.2 Installation Options . . . 1
- 1.3 Platform Support . . . 2

2 Developer's Guide **3**
- 2.1 Welcome . . . 3
- 2.2 CLI Utilities . . . 9
- 2.3 Station Development . . . 10
- 2.4 Tool Use and Development . . . 13
- 2.5 API Endpoints . . . 18
- 2.6 Flexible Data Layer . . . 22
- 2.7 Deployment . . . 22
- 2.8 Guidelines . . . 22

3 Station Executive Modules **23**
- 3.1 stationexec.executive module . . . 23
- 3.2 stationexec.main module . . . 24
- 3.3 stationexec.version module . . . 24

4 Sequencer **25**
- 4.1 stationexec.sequencer.sequencer module . . . 26

5 Operation **29**
- 5.1 stationexec.sequencer.operation module . . . 29
- 5.2 stationexec.sequencer.handlers module . . . 33
- 5.3 stationexec.sequencer.loop_condition module . . . 33
- 5.4 stationexec.sequencer.operationloop module . . . 35
- 5.5 stationexec.sequencer.operationstates module . . . 36

6 Station **37**
- 6.1 stationexec.station.data_storage module . . . 37
- 6.2 stationexec.station.events module . . . 40
- 6.3 stationexec.station.handlers module . . . 42

7 Tools **43**
- 7.1 stationexec.toolbox.tool module . . . 43
- 7.2 stationexec.toolbox.toolbox module . . . 44
- 7.3 stationexec.toolbox.asynctoolbase module . . . 46
- 7.4 stationexec.toolbox.camerabase module . . . 50
- 7.5 stationexec.toolbox.handlers module . . . 51

7.6 stationexec.toolbox.tool_launch module . . . 52
7.7 stationexec.toolbox.tool_utilities module . . . 54

8 Authentication **55**
8.1 stationexec.authentication.auth module . . . 55
8.2 stationexec.authentication.authlevel module . . . 55
8.3 stationexec.authentication.login module . . . 55
8.4 stationexec.authentication.logout module . . . 56
8.5 stationexec.authentication.tokenmanager module . . . 56
8.6 stationexec.authentication.userauthinfo module . . . 57

9 CLI Utilities **59**
9.1 stationexec.cli.cli_tools module . . . 59
9.2 stationexec.cli.common module . . . 59
9.3 stationexec.cli.for_stations module . . . 60
9.4 stationexec.cli.for_tools module . . . 60

10 Built In Tools **61**
10.1 stationexec.built_in.exampletool module . . . 61
10.2 stationexec.built_in.exampletool2 module . . . 61
10.3 stationexec.built_in.station_storage module . . . 61

11 Logger **63**
11.1 stationexec.logger.log module . . . 63
11.2 stationexec.logger.logger module . . . 64

12 Utilities **65**
12.1 stationexec.utilities.byte_conversion module . . . 65
12.2 stationexec.utilities.classproperty module . . . 65
12.3 stationexec.utilities.colors module . . . 65
12.4 stationexec.utilities.config module . . . 66
12.5 stationexec.utilities.exceptions module . . . 67
12.6 stationexec.utilities.gettool module . . . 68
12.7 stationexec.utilities.result_references module . . . 68
12.8 stationexec.utilities.shutdown module . . . 69
12.9 stationexec.utilities.singleton module . . . 69
12.10 stationexec.utilities.time module . . . 69
12.11 stationexec.utilities.uuidstr module . . . 70

13 Web **71**
13.1 stationexec.web.handlers module . . . 71
13.2 stationexec.web.web_helpers module . . . 72
13.3 stationexec.web.websocket module . . . 72

14 Build and Release **75**

15 Documentation **77**

16 Generating the Documentation **79**
16.1 Install sphinx . . . 79
16.2 Generate Docs . . . 79
16.3 Install LaTeX for PDF . . . 79

17 Release notes **81**
17.1 What's new in Station Exec 0.7.4 . . . 81
17.2 What's new in Station Exec 0.7.3 . . . 81

17.3 Station Exec 0.7.0, 0.7.1, 0.7.2 . . . 82
17.4 What's new in Station Exec 0.6.1 . . . 82
17.5 What's new in Station Exec 0.6.0 . . . 82
17.6 What's new in Station Exec 0.5.6 . . . 82
17.7 What's new in Station Exec 0.5.5 . . . 82
17.8 What's new in Station Exec 0.5.4 . . . 82
17.9 What's new in Station Exec 0.5.3 . . . 83
17.10 What's new in Station Exec 0.5.2 . . . 83
17.11 What's new in Station Exec 0.5.1 . . . 83
17.12 What's new in Station Exec 0.5.0 . . . 83
17.13 What's new in Station Exec 0.4.4 . . . 84
17.14 What's new in Station Exec 0.4.3 . . . 84
17.15 What's new in Station Exec 0.4.2 . . . 85
17.16 What's new in Station Exec 0.4.1 . . . 86
17.17 What's new in Station Exec 0.4.0 . . . 86
17.18 What's new in Station Exec 0.3.1 . . . 87
17.19 What's new in Station Exec 0.3.0 . . . 87
17.20 What's new in Station Exec 0.2.0 . . . 87
17.21 What's new in Station Exec 0.1.1 . . . 87
17.22 What's new in Station Exec 0.1.0 . . . 88

HTTP Routing Table **89**

Python Module Index **91**

Index **93**

CHAPTER

ONE

STATION EXECUTIVE

Station Executive is a light-weight, flexible software framework for sequencing tasks and interacting with external data sources - including physical hardware - in a networked context.

Each `Station` is comprised of a series of `operations` and a set of `Tools`. The `operations` are intelligently executed based upon data-flow dependencies and in parallel (as OS and hardware support allows). A `tool` is any source or sink of data that can be used to accomplish the tasks (anything from a robot to a database).

1.1 Quick Start

Run in command line (after installation)::

```
se-hello 8888
```

Open your browser and navigate to 'http://localhost:8888' to view the main UI.

1.2 Installation Options

From distributed package:

```
# Windows
pip install stationexec-<version>-py2.py3-none-any.whl

# Linux as User
pip install --user stationexec-<version>-py2.py3-none-any.whl

# Linux as Root
sudo pip install stationexec-<version>-py2.py3-none-any.whl
```

Running From Repository Clone:

```
# Editable installation
pip install -e .
se-hello 8888
```

Python Virtualenv:

```
# Fedora (change 2 to 3 for Python 3) - Run in project folder
sudo yum install python2-virtualenv
virtualenv-2 env2
```

(continues on next page)

(continued from previous page)

```
source env2/bin/activate
pip install stationexec-<version>-py2.py3-none-any.whl
```

1.3 Platform Support

OS	32bit	64bit	Python Version	Tested
Windows	Untested	Yes	2.7.9+, 3.6	Windows 10
Linux	Untested	Yes	2.7.9+, 3.6	Fedora 28/29, Centos 7, Ubuntu 16+
ARM	Yes	Yes	2.7.9+, 3.6	Debian
Mac	Untested	Yes	2.7.9+	OSX 10.10+

CHAPTER

TWO

DEVELOPER'S GUIDE

2.1 Welcome

Welcome to Station Executive! Station Executive (stationexec) is a light-weight, flexible, automation software framework to facilitate sequencing and interacting with external data sources in a manufacturing environment. It is especially well suited for automating task execution, assembly, and testing. Stationexec is inherently network ready, allowing derived stations to easily create or join a complex networked environment.

The 'Station' in Station Executive is meant to invoke the metaphor of a train station. Each 'Station' is a stop along the way to a final destination in a journey. The framework is to be used to create these 'stations' that will automate, assemble, test, and more in any manufacturing-related context. The network-readiness of stationexec allows each individual station to be orchestrated by an external system to create a fully-automated system.

This developer guide walks through how to install, setup, develop, test, and deploy a customized instance of stationexec replete with tools, a UI, a web API, databased storage, and more.

2.1.1 Anatomy

Stationexec is comprised of three primary components: the stationexec core library, stations, and tools.

A deployment of stationexec on a system will look like a 'stations' and 'tools' folder inside of some master folder in user space (called 'stationexec' for demonstration purposes). Each of the 'stations' and 'tools' folders will contain all of the necessary station and tool definitions for an instance of the deployed station type (explored below).

A full tree for example:

```
stationexec/
├── stations
│   ├── __init__.py
│   ├── example
│   │   ├── __init__.py
│   │   ├── operations.json
│   │   ├── operations.py
│   │   ├── station.json
│   │   ├── station.py
│   │   ├── tool_manifest.json
│   │   └── ui
│   │       └── example_ui.html
│   └── ...
├── tools
│   ├── __init__.py
│   ├── example_tool
│   │   ├── example_tool.py
```

(continues on next page)

(continued from previous page)

```
│   │   ├── __init__.py
│   │   └── ui
│   │       └── example_tool_control.html
│   ├── robot
│   │   ├── __init__.py
│   │   ├── robot.py
│   │   └── ui
│   │       └── robot_control.html
│   └── ...
├── example.json
└── <identifier>.json
```

To run the example station as shown in this example, execute from command line from this folder (–localonly flag is used to disable remote database access, as it is generally not configured at the early stages of development):

```
stationexec -f example.json --localonly -p 8888
```

The UI for this station could then be viewed at http://localhost:8888

stationexec core library

The stationexec core library is provided as a Python .whl library that can be installed using pip. This library is a versioned, self-contained package with all core functionality and features. This includes:

- The Sequencer - handles multi-threaded execution of prioritized operations with dependencies. Operations written by the station creator will be loaded by the sequencer, ordered according to critical path, priority and dependency, then executed.
- The Toolbox - manages tool connectivity and access control. It loads and initializes all defined tools and maintains connectivity, resources, and health for each tool. Access to each resource is through the toolbox to maintain order in a multi-threaded environment.
- The UI - many components have customizable user interfaces to allow interaction with the station, both for control and visualization. The base user interface is built-in to the library, allowing station authors to create simple HTML modules for each component that will automatically be incorporated into the system at large.
- The Server - primary functionality of stationexec is exposed via REST interfaces to allow for natural interaction in a networked context. Stationexec can provide data and be controlled over these interfaces. An author can also easily create new endpoints to extend functionality to the station and tools.

The core library itself will not require editing by the author when designing new stations and tools.

Stations

A Station is a deployed instance of stationexec that manages a system used for automation, assembly, test, or processing of any kind. A station can be as simple or complex as required - there are no minimum or maximum specifications for what a system is able to accomplish.

In practice, a station definition is simply a folder that contains all of the necessary files that give a station its identity. Each station folder is self-contained and can co-exist with any number of other station definitions in the 'stations' folder.

A new station definition template can be created by using the `se-gen` CLI tool. This will create a station folder filled with template files for a station that the author can then edit to meet their needs (see the `CLI Utilities` section of this guide for more details, usage instructions, and other helpful utilities).:

```
# Create a 'special_delivery' station definition in the 'stations' folder
se-gen --type station --name special_delivery
```

An individual station is made of three primary parts:

1. Station Definition
2. Operation Definition
3. Tool Manifest

Station Definition

The station definition is given by an event handling file, a configuration file, and the user interface.

The **station.py** is a place to handle station events. Use this space to determine if conditions are met for the sequence to be able to run, to handle setup/teardown tasks at the beginning and end of a sequence run, handle emergency stop events, and start/stop the sequence based on hardware input.

Configuration

The station configurations for startup are contained within two different files. The first is **station.json**, which resides inside the named station folder ('example' as seen in the file tree above) and contains the default settings for all station instances created from the station definition in the folder.

The second is **<identifier>.json**, which is a file that should be named descriptively to associate it with the station it invokes. This file describes the station instance that will be launched and is passed in as an argument to the command-line launcher (after the -f as seen earlier). This file can live anywhere - it is recommended to store these in the top-level folder for ease of access. The additional configurations can be used to override the configurations from station.json per each instance.

This file identifies which station it can be used to launch, the id of the station instance, a display name for the station, and any additional configurations that are unique for a particular instance of the station.

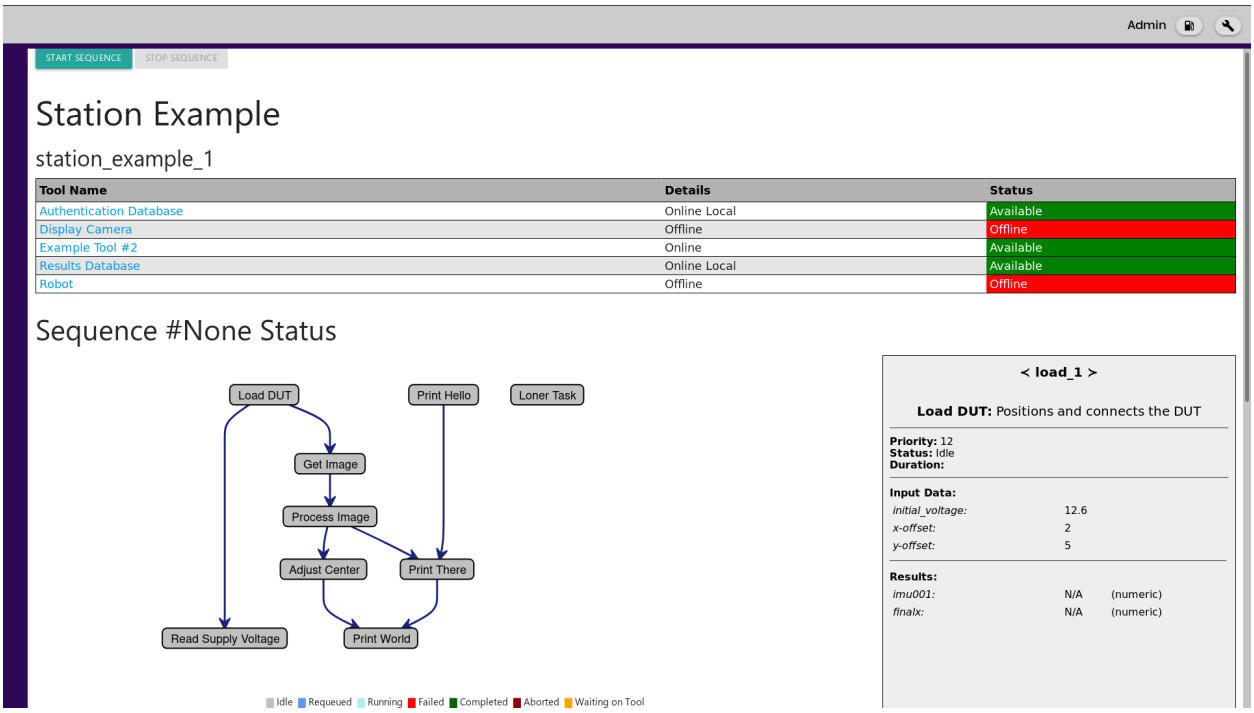
Example

A bare-bones top-level config file for an existing station definition of "subway" (the station folder is named "subway") with an instance identifier of "workcell_1" and a display name of "Workcell Demonstration" would look like this:

```
{
  "station_type": "subway",
  "station_instance": "workcell_1",
  "display_name": "Workcell Demonstration"
}
```

User Interface

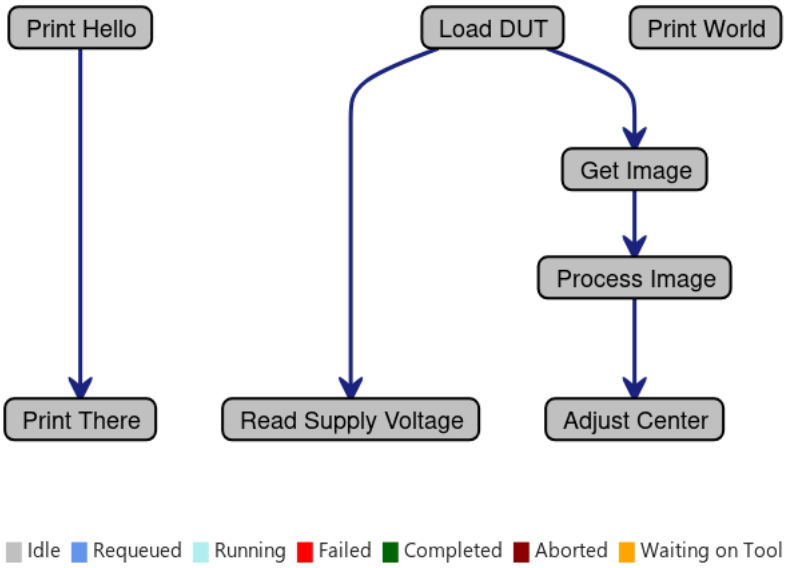
Each station has a UI folder that should contain an .html file called **<station_id>_ui.html**. This UI will be the default page displayed when stationexec runs for this station. Populate the UI as required for display and operation.



Operation Definition

Operations are the individual tasks to be executed in the station. The author defines both the content of the operations (**operations.py**) and the relationships of the operations in the context of the sequence (**operations.json**) The **operations.py** file contains the content of the tasks, what work they accomplish, and how the tools are used. The **operations.json** file describes how to reference the operation, what data the operation saves to database (including limits), constants the operation may need, and dependency relationships between operations.

Stationexec takes the operations and their defined characteristics and generates a directed acyclic graph (DAG) to represent the execution flow. More information on how this happens can be found in the `Sequencer` documentation in the software documentation section. Parallel paths through the sequence will be automatically executed in parallel (depending upon setup configuration) for most efficient sequence execution, in an order determined by their dependencies, number of dependents, historical runtime, critical path location, and other factors.



Example DAG view

View the Sequence information in the full `Station` section of this guide for more details on how to write an operation, how to setup a sequence, and what utilities are available to make your station creation successful.

Tool Manifest

The tool manifest (**tool_manifest.json**) defines what tools will be made available to the station during its operation. The file contains information on which tools to use, how to refer to them, and what the connection/initialization parameters are for each tool. The toolbox will take this data and attempt to connect to the tools according to the specified parameters and to keep the tools connected. The tools referenced in the manifest must already be implemented and present in the 'tools' folder.

In an operation, the author will be able to checkout a tool (as long as it is connected and not in use by another process) using the `checkout_tool` method.

Example

An example entry in the tool manifest file may look like this:

```
{
    "tool_type": "exampletool",
    "name": "Example",
    "tool_id": "example_tool_1",
    "configurations": {
        "host": "127.0.0.1",
        "port": 8000
    }
},
```

The toolbox will search for a defined tool called 'exampletool' inside the 'tools' folder, and the toolbox will create a tool with id 'example_tool' and display name of 'Example'. The 'tool_type' can be thought of as the name of the class of the tool, and the 'tool_id' is like the name of a class instance. Multiple tools can use the same 'tool_type' to create different instances of a similar tool on one station. Each 'tool_id' must be unique. The toolbox will connect to

this tool according to how the tool is written using the provided configurations at stationexec startup. To use this tool in an operation, the author would use `checkout_tool` to reference the tool by 'tool_id' as follows:

```
try:
    tool = self.checkout_tool('example_tool_1')
except Exception:
    # Unable to checkout tool - was not found, was offline, or was in use
    # for too long - abort sequence
    return OperationState.ERROR

tool.do_action_1()

# Optional return of tool
self.return_tool(tool)
```

The Tool return at the end is optional because the sequencer will automatically return all Tools checked out during an operation when the operation completes. Since a Tool may be used by multiple operations, it is more efficient to return it as soon as possible so the other Operations may use it.

Tools

Tools refer to any data source or sink that a station can interact with. This will often be hardware items like cameras, robots, TCP/Serial data acquisition or control devices, and more. The Tool API consists of six methods that a new tool must implement in order for the Toolbox to load and manage it correctly. Beyond these methods, the author is allowed to create any methods required to fulfill their needs, such as methods to format or return data.

The Toolbox provides continuous monitoring, health checking, reporting, and reconnecting for every tool, using the API methods the author must complete. Beyond this, the toolbox manages access to the tool object to prevent collisions. A tool object can be "checked out" during an operation which allows the author to call any of the tool's constituent methods. No tool may be accessed without first checking it out.

Create A New Tool

A new tool can be started by using the `se-gen` CLI tool. This will create a tool folder filled with template files for a tool that the author can then edit to meet their needs (see the `CLI Utilities` section of this guide for more details, usage instructions, and other helpful utilities).:

```
# Create a tool of tool_type 'dancing_robot' tool in the 'tools' folder
se-gen --type tool --name dancing_robot

# Create a TCP or Serial controlled tool of tool_type 'custom_controller' in the
↪'tools'
# folder that follows the async tool template
se-gen --type tool --name custom_controller --tooloptions async

# Create a camera tool of tool_type 'special_camera' in the 'tools' folder that␣
↪follows
# the camera tool template
se-gen --type tool --name special_camera --tooloptions camera
```

User Interface

Each tool has a ui folder that should contain an .html UI titled **<tool_id>_control.html**. This UI is intended to expose granular control of the tool for calibration, testing, maintenance, or debug purposes. Buttons on the UI will correspond

to method calls in the `on_ui_command` method of the tool.

2.2 CLI Utilities

Stationexec provides several command line utilities to assist the author, developer, and user in all of their tasks. These entry-points are installed when the stationexec library is installed on the machine and can be executed directly by name in a command line on any operating system.

2.2.1 se-hello

The example 'hello-world' style program for stationexec. Run this to experience an example run of stationexec and see a station and tools in action.

```
se-hello 8888
```

Navigate to http://localhost:8888 in a browser to interact with the example

2.2.2 se-setup

Setup folder structure of stationexec

2.2.3 se-start

The stationexec launcher creates a running instance of a station. The run command pairs with the system configuration file that can add to or override station-level configuration options for this run.

e.g. Launch the station 'robot_manager' viewable at port 8448 at debug level 2.

```
se-start robot_manager --port 8448 --debug 2
```

Options

Command	Short	Help	Type
–debug	-d	verbosity level for logging (default 0)	numeric
–instance	-l	unique name for running station instance	string
–name	-n	proper name for running station instance	string
–port	-p	server port number (default 8888)	numeric
–threads	-t	max parallel operation threads (default 10)	numeric
–dev		development mode flag	
–file	-f	file that contains any configurations	string path

2.2.4 se-tool

Options: gen - generate a new tool launch - launch a named tool for testing and debugging list - show all tools found in the system

```
se-tool launch exampletool
```

2.2.5 se-which

Show the install location of stationexec

2.3 Station Development

A new station can be started by using the `se-gen` CLI tool. This will create a station folder filled with template files for a station that the author can then edit to meet their needs (see the `CLI Utilities` section of this guide for more details, usage instructions, and other helpful utilities).:

```
# Create a 'special_delivery' station in the 'stations' folder of top-level
↪'stationexec' folder
se-gen -t station -l stationexec/stations -n special_delivery
```

An individual station is made of three primary parts:

1. Station Definition
2. Sequence Operations
3. Tool Manifest

2.3.1 Sequence Definition

The sequence definition is made up of a JSON file (operations.json) and a Python file (operations.py). These two files describe the connection and function of all operations and are used to define and execute a sequence.

operations.json

Fields

id - Unique string ID that maps to the operation class name in the Python file file - (optional)

operation_name - String display name of operation

description - String description of operation purpose

operation_results - List of objects defining the results that will be returned from the operation. List can be empty, but the list must exist.

result object fields: - name - description - type (see valid list below) - lower - upper

Current valid types:

- numeric
- integer
- float
- text/plain
- text/html
- binary
- image/jpeg
- image/png
- image/gif
- image/bmp
- audio/mpeg

- audio/ogg
- audio/*
- audio/midi
- audio/wav
- video/mp4
- video/ogg

dependency_list - List of "id" fields of other operations that must complete before this operation is valid to run.

operation_data - Object containing parameters that the operation will need for its functionality. These values may be constant, but may also refer to the results from prior operations by prefixing the result name with 'result:', e.g. if one operation produces a result names "index", a later operation may use that result as data by specifying the value "result:index".

Example operation definition:

```
[
  {
    "id": "ExampleOperation01",
    "operation_name": "Example Operation",
    "description": "Creates a place for everything and puts everything in its place",
    "operation_results": [
      {
        "name": "thing01",
        "description": "first thing to be performed; should be greater than 12",
        "lower": 12
      },
      {
        "name": "thing02",
        "description": "second thing; should be in range 4459.2-4522.9",
        "lower": 4459.2,
        "upper": 4522.9
      }
    ],
    "dependency_list": [
    ],
    "operation_data": {
      "initial_value": 12.6,
      "offset_1": 2,
      "y-offset": 5
    }
  },
]
```

operations.py

prepare

operation_action

tool check-out/check-in

algorithm server

requeue

return states

- OperationState.COMPLETED, OperationState.ERROR, or OperationState.REQUEUE

overflow file

```
from stationexec.logger import log
from stationexec.sequencer.operation import Operation
from stationexec.sequencer.operationstates import OperationState

class ExampleOperation01(Operation):
    def prepare(self):
        # Optional preparation step; can use to determine if conditions are right
        if not <conditions required to run>:
            # Requeue this task if cannot run at this time
            log.info("Re-queuing")
            return OperationState.REQUEUE
        return OperationState.COMPLETED

    def operation_action(self, incoming_data, expected_results):
        log.info("I am class {0}".format(str(self.__class__)))

        # Check-out tool defined in manifest for operation
        self.checkout_tool('example_tool')

        # Get operational data from "operation_data" in JSON file
        start_value = incoming_data["initial_value"]

        # Store expected results (defined in "operation_results")
        self.save_result("thing01", 12.07)
        self.save_result("thing02", 4461.62)

        if <something is broken>:
            log.error("Cannot run - something is broken")
            return OperationState.ERROR
        time.sleep(1)

        return OperationState.COMPLETED
```

2.3.2 Station Definition

2.3.3 Station Tool Manifest

```
[
    {
        "tool_type": "example",
        "name": "Example",
        "tool_id": "example_tool",
        "configurations": {
            "port": 8001,
            "exposure": 3,
            "example_config": "value"
        }
    },
    {
        "tool_type": "example2",
        "name": "Example 2",
        "tool_id": "example2_tool",
        "configurations": {
            "amazing_thing1": 1,
```

(continues on next page)

(continued from previous page)

```
            "amazing_thing2": 2,
            "amazing_thing3": "skymall"
        }
    },
]
```

2.3.4 Create a Station

Start by making a copy of the 'station_template' folder inside of 'stationexec/templates' into the workspace of your choice inside a parent level station folder. The folder structure should resemble the one below:

Station Folder Structure

- **Parent Tool Folder**
  - **station_template - Folder**
    - __init__.py - Required; empty
    - operations.json - Required
    - operations.py - Required
    - station.json - Required
    - station.py - Required
    - station_handler.py - Optional
    - tool_manifest.json - Required
    - **ui**
      - station_template_ui.html - Required
- __init__.py - Required; empty

This layout represents a tool of type 'station_template'. Change the names of things to represent the tool that you are developing. For an example tool called Robot Station, the names would be changed as follows:

- 'station_template' folder becomes 'robot_station'
- 'ui/station_template_ui.html' becomes 'ui/robot_station_ui.html'

'station_handler.py' is optional and does not have a controlled name so it can be changed or removed as desired. If making edits to the file name, be sure to adjust the import path at the top of the main station .py file to reflect the change and new location. Also change the name of the html file rendered in the handler to the new name of the file in the ui folder.

2.4 Tool Use and Development

A tool can be any sort of data source or sink. This could be anything from a serial device to a custom software application to a database to a web application. Each folder inside 'tool' defines a distinct tool and how to interact with the tool.

A tool object inherits from *stationexec.toolbox.tool* and implements the basic interaction methods. Beyond these few methods, the tool object provides the methodology to interact with the tool of choice. View the example tool (stationexec/built_in/example) to see how this is done. The 'station' definition contains a tool manifest (tool_manifest.json) that defines which tools will be loaded and which parameters will be provided to configure

the tool. The `toolbox <stationexec.toolbox.toolbox>` loads all tool objects, maintains connections, provides status monitoring, and provides access to the tools upon request.

2.4.1 Create a Tool

Start by making a copy of the 'tool_template' folder inside of 'stationexec/templates' into the workspace of your choice inside a parent level tool folder. The folder structure should resemble the one below:

Tool Folder Structure

- **Parent Tool Folder**
  - **tool_template - Folder**
    - __init__.py - Required; empty
    - tool_template.py - Required
    - tool_handler_template.py - Optional
    - **ui**
      - custom_tool_ui.html - Optional
      - tool_template_control.html - Required
- __init__.py - Required; empty

This layout represents a tool of type 'tool_template'. Change the names of things to represent the tool that you are developing. For an example tool called Special Robot, the names would be changed as follows:

- 'tool_template' folder becomes 'special_robot'
- **'tool_template.py' becomes 'special_robot.py'**
  - 'class ToolTemplate' inside of the file becomes 'class SpecialRobot'
- 'ui/tool_template_control.html' becomes 'ui/special_robot_control.html'

'tool_handler_template.py', 'class ToolHandlerTemplate' inside the file, and 'ui/custom_tool_ui.html' are both optional and do not have controlled names so they can be changed or removed as desired. If making edits to the handlers, be sure to adjust the import path at the top of the main tool .py file to account for the name changes. This should also be adjusted from the default to account for the new folder as it is no longer inside the stationexec package.

Programming

All tools must ultimately inherit from the *`Tool`* base class. The *`Tool`* class provides underlying functionality to plug your tool into the StationExec framework and also to aid in common tasks. There are a few required methods to implement when developing a tool. They may not all be required for the particular tool, and so may be empty (`pass`), but they must exist. These are the bare minimum of methods required for a tool. Any other methods for operation can be added alongside these existing and will all be available for use in normal operation (see `Setup and Use Tool` below).

__init__

Setup and store class member variables. The available arguments come from the 'configurations' list in the tool_manifest file for the station. Must call the `super` method with **kwargs to setup the base tool.

initialize

Perform all actions required to set device up completely for use. After this method is called, the tool is considered ready to use. Recommended to call 'self.set_online' or 'self.set_offline' with the outcome of the initialization. If the

initialization fails the driver should be written to allow execution to continue. `verify_status` below should be able to detect and recover from the failure, assuming the failure is due to external issues like a faulty cable or network connection and can be recovered from.

verify_status

Periodically called (default 5 seconds) to determine if tool is online and healthy. Use as a space to fix the tool if things have gone awry.

shutdown

Close all open connections and cleanup all resources. Called when program is shutting down.

reset

Reboot a malfunctioning tool.

on_ui_command

Called when a POST is received at '/tool/command' (see `API Endpoints` for more details) targeted at this tool. This method should map the incoming 'command' argument and **kwargs to a tool method. See `Control UI` below for more details.

Tool Status

If the connection or health status of the tool ever changes, call 'self.set_online' or 'self.set_offline' to let the system know:

```
# Set the tool status to online
self.set_online()

# Set the tool status to offline
self.set_offline
```

Tool Utility

The ToolUtil class is intended to make development and debugging of tools in a context similar to deployment much simpler. The example below explains how to use the functionality to work with your tool. Add this code to the bottom of your main tool file and run the file to use.

Example:

```
if __name__ == "__main__":
    from stationexec.toolbox.tool_util import ToolUtil

    # Instantiate the ToolUtil object
    tu = ToolUtil()

    # Build the dictionary of configurations for the tool
    #  All 'key': 'value' pairs will be passed into the __init__
    #  of the tool to be used to setup the tool as needed.
    configurations = {
        # 'key': 'value',
    }

    # Build the correctly formatted dictionary to configure the tool␣
↪completely
    #  with this call. Arguments are: tool_type, name, tool_id,␣
↪configurations
```

(continues on next page)

(continued from previous page)

```
    #  tool_type: name of the tool folder and file
    #  name: display name for the tool - the friendly name
    #  tool_id: the unique identifier that is used to reference the tool
    #  configurations: dict from above
    tool_config = tu.build_tool_dict('example', 'Example', 'example_tool',␣
↪configurations)

    # Prepare the tool object - this call takes the tool object reference␣
↪itself
    #  (in this case, the 'example' tool, so pass the class 'Example' in)␣
↪and the
    #  tool_config prepared in the previous step.
    tu.tool_setup(Example, tool_config)

    # Commands are optional helpers that can call any tool function at some␣
↪later point
    #  in the future. This is helpful if you want to debug a function␣
↪without dealing with
    #  the web server or ui. The first argument is how many seconds to wait␣
↪before invoking
    #  the call. The next is the function itself. Then any args, then␣
↪keyword args. This list
    #  can be passed in to the tool_run function and all functions will be␣
↪called on schedule.
    commands = [
        # tu.tool_command_build(5, tu.obj.function_to_call, arg1, arg2,␣
↪kwarg1='abc'),
    ]

    # Start the tool running. Schedules all (if any) commands defined above␣
↪to run at the
    #  appropriate times. Starts a server at the port requested (default␣
↪port is 8888),
    #  and invokes the tool in a similar context to what it runs in during␣
↪deployment. With
    #  the server, you can access the tool UI at localhost:<port> and use␣
↪that to send commands
    #  to the tool. Duration is how many seconds the tool context runs for.␣
↪This is useful to
    #  force a clean shutdown after a test and nice for non-UI debugging.␣
↪Set to 0 for indefinite run.
    #  Navigate to localhost:<port>/shutdown to shutdown.
    tu.tool_run(commands, port=8888, duration=0)
```

2.4.2 Setup and Use a Tool

Tool Manifest

Tool Usage

Control UI

Checkout

2.4.3 Tool User Interfaces

Control UI

The HTML file in the UI folder of a tool that shares the name of the tool + _ui will be automatically served when the URL '/tool/<tool_id>/ui' is visited

e.g. for a tool of type example_tool:

- **example_tool - Folder**
  - __init__.py
  - example_tool.py - Required
  - **ui**
    - example_tool_ui.html - Required

The tool is instantiated with the following arguments in the tool_manifest file:

```
{
    "tool_type": "example_tool",
    "name": "Example Tool",
    "tool_id": "example_tool_1",
    "configurations": {}
}
```

The web page described in 'example_tool_ui.html' will be served from '/tool/example_tool_1/ui' The file must have the same name as the tool type + _ui.html or it will not be found.

Typically this page will be served in the main context of the StationExec user interface, so the header and body tags are not required.

Every one of these default tool pages has access to the pretty name of the tool and the id of the tool. In the HTML, insert **{{tool_name}}** where you would like the pretty name of the tool to appear and **{{tool_id}}** where you would like the id to appear when the page is rendered. To place in javascript, make sure to put quotes around it if it should be treated as a string.

Tool commands are sent by buttons. All buttons on the page that will send commands to the tool that spawned the page will have class="{{tool_id}}" so that the button directs to the proper tool The 'id' of the button will be passed to the tool object as one of the arguments for the 'on_ui_command' function. The tool will decide which method to execute based on this argument.

To pass arguments with a command, create form input objects that have a class that is the same name as the id of the button. So if a button performs 'action1' it will have id="action1"; if there is a numeric input that should be sent with it, the input will have class="action1". It is recommended to wrap the button and any related argument inputs inside of a for clarity.

The scripts that load from "/static/js/" are mapped to the 'stationexec/ui/js' folder in the stationexec package.

Custom UI and Handler

A custom HTML file can be added to the UI folder to serve any purpose that the developer wishes for it. It is entirely optional and can be used to extend the capability of a tool beyond the built-in UI. There are no requirements for the naming of the file - it will be served from the tool handler (if one exists). The file must exist in the ui folder of the tool it belongs to.

For example, in the case of the tool template folder, the ToolHandlerTemplate inside tool_handler_template.py will have the following inside its 'get' method:

```
self.render('custom_tool_ui.html')
```

This will show the web page described in this document on whichever URL endpoint was defined in the tool 'get_endpoints' method.

As the primary default UI had tool pretty name and id available to it as **{{tool_name}}** and **{{tool_id}}**, the handler can make any data available to this page as desired.

To pass in custom data to the rendering template engine, pass in named arguments to the self.render as follows:

```
self.render('custom_tool_ui.html', socket=8888, url='/amazing/thing', place='SkyMall)
```

Now in the HTML and javascript, the developer would be able to access these arguments by name - **{{socket}}** **{{url}}** and **{{place}}** anywhere they are useful.

2.4.4 Provided Tool Types

Async Tool

Camera

2.5 API Endpoints

Primary functionality of stationexec is exposed via REST interfaces to allow for natural interaction in a networked context. Stationexec can provide data and be controlled over these interfaces. An author can also easily create new endpoints to extend functionality.

2.5.1 To Be Defined

- /login
- /logout
- /restart
- /config
- /settings

2.5.2 Endpoints

`POST /shutdown`
: Shuts down stationexec

`POST /sequence/start`
: Starts the sequence

`POST /sequence/stop`
: Requests a stop to the running sequence

`GET /sequence/status`
: Returns JSON describing the status of each operation in the sequence

  Example Request (jquery):

```
$.ajax({
    url: "/sequence/status",
    dataType: 'json',
    context: this,
    complete: function (resp) {
        var data = resp.responseJSON;
}});
```

Example Response:

```
[
  {
    "active": true,
    "sequence_id": 4,
    "stop": false,
    "shutdown": false
  },
  {
    "status": 0,
    "description": "Submit image for processing",
    "order": 2,
    "priority": 5,
    "operation_name": "Process Image",
    "id": "ProcessImage"
  },
  {
    "status": 1,
    "description": "Return the string",
    "order": 1,
    "priority": 5,
    "operation_name": "Print There",
    "id": "There"
  }
]
```

Query Parameters

- **None** – None

Request Headers

- Authorization – Authentication Required

Response Headers

- Content-Type – content/json

Status Codes

- 200 OK – no error

GET /tool

Returns JSON describing available endpoints for tool user interfaces

Example Request (jquery):

```
$.ajax({
    url: "/tool",
    dataType: 'json',
```

(continues on next page)

(continued from previous page)

```
    context: this,
    complete: function (resp) {
        var data = resp.responseJSON;
}});
```

Example Response:

```
{
  "example_tool": {
    "ui": "/tool/ui/example_tool",
    "name": "Example",
    "cal": "/tool/cal/example_tool"
  },
  "example2_tool": {
    "ui": "/tool/ui/example2_tool",
    "name": "Example 2",
    "cal": "/tool/cal/example2_tool"
  }
}
```

Query Parameters

- **None** – None

Response Headers

- Content-Type – content/json

Status Codes

- 200 OK – no error

GET /tool/status

Returns JSON describing online/offline and status message for each connected tool

Example Request (jquery):

```
$.ajax({
    url: "/tool/status",
    dataType: 'json',
    context: this,
    complete: function (resp) {
        var data = resp.responseJSON;
}});
```

Example Response:

```
[
  {
    "status": "Offline",
    "tool_id": "example_tool",
    "tool_type": "example",
    "name": "Example",
    "inuse": false,
    "status_bool": false,
    "details": "Tool is offline"
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "status": "Online",
    "tool_id": "example2_tool",
    "tool_type": "example2",
    "name": "Example 2",
    "inuse": false,
    "status_bool": true,
    "details": "Tool is online"
  }
]
```

Query Parameters

- **None** – None

Response Headers

- Content-Type – content/json

Status Codes

- 200 OK – no error

POST /tool/command

Post JSON to this endpoint to perform an action with a tool

Example Request (jquery):

```
$.ajax({
    url: "/tool/command",
    headers: {'Content-Type':'application/json'},
    method: 'POST',
    dataType: 'json',
    data: JSON.stringify(command),
});
```

Query Parameters

- **command** – JSON argument below

Status Codes

- 200 OK – no error

JSON 'command' argument:

```
{
  target: <tool_id>,
  arguments: {
    command: <command>,
    arg1: 0
  }
}
```

GET /tool/ui/<tool_id>

Serves the control or maintenance/cal UI for the tool_id specified

Example Request (jquery):

```
$('#' + target_id).load("/tool/ui/<tool_id>", function(response, status, xhr) {
    if(status == "error"){$('#' + target_id).html("Unable to load");}});
```

Query Parameters

- **None** – None

Response Headers

- Content-Type – content/json

Status Codes

- 200 OK – no error
- 404 Not Found – page not found

2.6 Flexible Data Layer

TBD

2.7 Deployment

On target machine:

- Install stationexec library
- Create project directory in user space (e.g. 'stationexec')
- Create 'stations' and 'tools' folders in this directory
- Populate the 'stations' and 'tools' folders with the stations and tools required for the deployment
- Create the top level configuration .json file inside the top level folder (e.g. station_launch.json)
- Launch the station:

```
stationexec -f station_launch.json
```

2.8 Guidelines

The code inside and associated with this project should follow Python PEP8 as closely as possible. Supplementally, consult Google Python Style Guide for related best practices.

CHAPTER

THREE

STATION EXECUTIVE MODULES

3.1 stationexec.executive module

Executive

class stationexec.executive.**Executive**(*instance_config*)

Bases: object

The Executive oversees all operation of the station and holds all objects used by the station.

Initialize an Executive object and prepare to run *Operation*s for this Station instance.

Parameters **instance_config** (*dict*) – the configuration dictionary for this station id

config = None

station_info = None

initialize()

Create and initialize all sub-objects and prepare for actual processing.

get_cfg(*key*, *default=None*)

Get config value from station configuration or default (None, unless specified)

Parameters

- **key** (*str*) – the key to find
- **default** (*str*) – if specified, value to return if key not found; defaults to None

Returns value of key

get_endpoints()

Get the endpoints for each sub-object used by Station

shutdown(*user_token=None*, ***kwargs*)

stop all threads in this and lower objects, and prepare to exit

get_web_info()

Returns a tuple with the configured web port and either a dict with the SSL config info for this station instance or else None if not configured for https.

ui_command(*source*, ***kwargs*)

Put ui command in the processing queue

Launches _ui_command thread loop if it is not running

Parameters

- **source** (*str*) – source of the event

- **kwargs** (*dict*) – Event data dictionary data: target: cmd: JSON string arguments from UI

Returns None

launch_sequence(*user*, *runtimedata*, ***kwargs*)

terminate_sequence(*user*, ***kwargs*)

sequence_setup()

sequence_cleanup()

sequence_status()

estop(***kwargs*)

estop_clear(***kwargs*)

station_uuid

3.2 stationexec.main module

class stationexec.main.**Main**

Bases: object

Set up tornado configuration, set station and instance, and initialize other objects.

start()

shutdown(***kwargs*)

3.3 stationexec.version module

CHAPTER

FOUR

SEQUENCER

The *Sequencer* takes a list of operations to be performed on a station and, using their dependencies, calculates a priority for each *Operation* which it then uses to schedule the sequence of operations based on the configured threads available on the station.

Each operation is a uniquely named class derived from the *Operation* class, and the class name is used as the id of the *Operation*. The sequence specification file for a station, operations.json, specifies the operation id(s) that each Operation depends on, i.e. requires to be completed successfully before it can run. This allows us to calculate a simple relationship between the operations and assign them a priority of execution based on how many later jobs depend on them, which allows us to calculate the critical path and attempt to execute those operations first at each stop.

The *Sequencer* works using 4 simple lists to hold the state of each Operation:

- the waiting list, where operations wait until their dependencies have completed
- the ready list, which holds operations that are ready to run because their dependencies have been completed
- the running list, which hold operations currently executing
- the completed list, which holds operations that have finished (successfully or not)

All *Operation* in the ready list have had all their dependencies completed, but may be forced to stay in the ready list due to threading limits of the station or because they are waiting on some tool, algorithm result, or similar data. For example, one step may submit data to an algorithm server and return a request id as data, and the following step may use that request id to query the algormithm server whether the result is ready yet, returning REQUEUE until the algorithm result is ready for further processing.

Each *Operation* has a prepare method which is called just before it is run, and this method can return either *OperationState.COMPLETED* or *OperationState.REQUEUE*; the latter asks that the operation be returned to the ready list. This prepare method can be used to ensure tools, resources, or algorithm results are available.

A running *Operation* can exit with *OperationState.COMPLETED*, *OperationState.REQUEUE*, or *OperationState.ERROR*. If exiting with *REQUEUE*, the *Operation* will be moved from the running list back to the ready list. If exiting *COMPLETED* or *ERROR*, the *Operation* will be moved to the completed list, except that an *ERROR* will trigger the sequence of operations to abort, the assumption being that any error exit state indicates a fatal result was obtained and the sequence of operations cannot continue.

When the *Sequencer* begins, it starts by putting all the operation ids into the waiting list, then it repeatedly iterates over the lists, doing:

1. For each operation in the waiting list, if all the dependent jobs are in the completed list, then move the operation id to the ready list.
2. While the current count of operations running is less than the station maximum, try to run a operation from the ready list, moving that operation id to the running list. Select operations with the highest priority first.
    a. Run the operation's prepare method, and use the exit code to decide whether to move it to the running list or return it to the ready list.

3. For each operation id in the running list, monitor it to see when it completes. Upon completion, gather operation results and move operation from the running to the completed list.
4. Stop when there are no operation ids in the waiting, ready, or running lists, or when an operation returns ERROR.

4.1 stationexec.sequencer.sequencer module

This class contains the *Sequencer* implementation used to calculate and run a sequence of *Operation* on a Station.

class stationexec.sequencer.sequencer.**Sequencer**(*checkout_tool*, *return_tool*, *station_info*, *get_config*)

Bases: object

The Operation sequencer

Initialize the Sequencer.

Parameters

- **checkout_tool** –
- **return_tool** –
- **station_info** (*Addict*) –
- **get_config** (*function*) – system config data accessor

initialize()

Initialize the Sequencer by loading the station operations list and initializing all the operation objects.

Raises ***InvalidAttribute*** – if operations.json file has an error

run(*user_token*, *runtime_data*)

Start a Thread to run the loaded sequence from the beginning. Call *stop()* to stop the sequence early.

Parameters **user_token** (*str*) – token of the user starting this run [required]

reload(*user_token=None*)

Reload the operations.json and associated .py files from disk.

Parameters **user_token** (*str*) – token of currently logged in user

stop(*user_token=None*)

Stop any running sequence of operations.

Parameters **user_token** (*str*) – [optional] token of user who asked for shutdown, or None for system

shutdown(*user_token=None*)

Stop all threads in this and lower objects

Parameters **user_token** (*str*) – [optional] token of user who asked for shutdown, or None for system

is_available()

Return True if we are not running nor preparing to run a sequence.

is_running()

Return True if a sequence is currently running. This does *NOT* include if we were asked to run a sequence and are preparing to run it at the current moment but have not yet started it.

current_sequence_id()
Return the UUID of the current (if running) or last (if not running) sequence. Return None if no sequences have been run since boot up.

get_all_operation_status()
Returns a dictionary with the current status of all operations in this sequence under the 'operations' key, the sequence state in the 'sequence' key, and the loop information in the 'loops' key.

Returns list of dictionaries of operation information; 1st record is sequence information

Return type dict(str,object)

sequence_success_determinator()
Return an `OperationStatus` representing the current or prior sequences status.

get_operation_result(*value*)
Get the generated result for the Operation result specified in the string.

BUT, keep in mind that what looks like a reference to another Operation's result might just be raw data, such as a hex value "AC:F1"!!

If the value is a valid reference to an existing Operation's declared result, we will either return the result or else raise ValueError if the Operation has not yet generated that result.

If the value does not match any Operation's declared result, then we also raise a ValueError.

Returns generated value or None

Raises ValueError if specified result could not be found

shutdown_requested
True if a call to *shutdown()* (shutting down station) has occurred.

stop_requested
True if a call to *stop()* (stop a sequence) has occurred.

CHAPTER

FIVE

OPERATION

An *Operation* is a step within the larger sequence of operations that a particular station executes. The base class implements support routins, and the actual station implementation should override the *operation_action()* method with the operations to perform.

An Operation may also override the *initialize()* method, and may override the *prepare()* method to perform any necessary checks before the operation is run, such checking as whether an algorithm result is available or a tool is available.

When completed, the *Sequencer* will call the *cleanup()* method for the Operation.

When it begins executing, the Operation will be passed the data specified in the station's `operations.json` file. This data can include references to results from earlier operations by specifying the earlier result name as `result:<name of result>`. (Note: This requires that multiple operations will not generate the same result name.) The incoming data dict is provided as the first argument to the *operation_action()* method. The second argument to that method is the dict of expected results to be generated by this Operation, for reference. The generated results *will* be checked against this list by the Sequencer once the Operation completes.

While executing, the Operation should call the *is_shutdown()* method occasionally to query if the Sequencer is asking the Operation to shutdown i.e. abort. If True, the Operation should exit as quickly as is practical, ignoring results; in this state, the sequence is being stopped and aborted, so results are not important.

The Operation should save results using the *save_result()* method, giving the name of the result and its value. This name must match that specified in the station's operations.json file, and the Sequencer will verify that all expected results have been generated by an Operation or else mark that Operation as failed.

An Operation should use the *checkout_tool()* method to get direct access to a station tool, and should call *return_tool()* when the tool is no longer needed, so that other operations can use the tool. If the developer forgets to return a tool, the Operation base class will return it at the end of the operation's run, but developers should remember to return the tool as soon as possible to make it available quickly to other Operations.

Before exiting, the Operation should call the *set_status()* method with *OperationState.COMPLETED* or *OperationState.ERROR*. If necessary, the Operation may call it with `OperationState.REQUEUE` before completing its execution, and the Operation will be requeued by the Sequencer to run again at a later time.

5.1 stationexec.sequencer.operation module

This class defines routines common to all sequence operations. An Operation is a part of a Sequence of operations needed to perform a step in the building of a device. The operations are all performed on one station.

class stationexec.sequencer.operation.**Operation**(*operation_id*, *name*, *tool_checkout*, *tool_return*)

Bases: `object`

An Operation is the base class of all operations in a station's sequence of steps needed to perform a function, and it defines all the common, default routines.

Create object.

Parameters

- **operation_id** (*str*) – the name of the class, a unique internal id for referring to this operation
- **name** (*str*) – the short human readable text describing this operation
- **tool_checkout** (*function*) –
- **tool_return** (*function*) –

get_id()
Get the unique id/class of this operation

Returns the id

Return type str

get_name()
Get the human readable name of this operation

Returns the name

Return type str

initialize()
Override this method to initialize object.

prepare(*incoming_data*, *expected_results*)
Override this method to perform any operations necessary before running this operation.

This method should NOT attempt to acquire tools.

Parameters

- **incoming_data** (*dict(str,str)*) – incoming data used for running this operation
- **expected_results** (*list(dict(str,str))*) – data on EXPECTED results from this operation

Returns *ERROR*, *COMPLETED*, or *REQUEUE*.

Return type OperationState

operation_action(*incoming_data*, *expected_results*)
Override this method with the actions to perform for this operation.

If self.is_shutdown() becomes true, this method should cleanly exit as soon as possible.

This method should save results by calling *self.save_result()*. If the method does not create a result for every item in expected_results, then the operation will automatically fail.

Parameters

- **incoming_data** (*dict(str,str)*) – incoming data used for running this operation
- **expected_results** (*list(dict(str,str))*) – EXPECTED results from this operation

Returns *COMPLETED*, *ERROR*, or *REQUEUE*

Return type *OperationState*

run(*incoming_data*, *expected_results*, *actual_results*, *loop_iteration*)
DO NOT OVERRIDE THIS METHOD!

Called by the Sequencer to invoke this operation. Checks that all results were supplied. Frees any tool requested but not released.

initialize() must be called before calling this method.

Parameters

- **incoming_data** (*dict(str,str)*) – incoming data used for running this operation
- **expected_results** (*list(dict(str,str))*) – data on EXPECTED results from this operation
- **actual_results** (*dict(str,str)*) – dictionary for ACTUAL output results
- **loop_iteration** (*int*) – iteration of loop containing node, or None if not in a loop

Returns *COMPLETED*, *ERROR*, *REQUEUE*, *RESULTFAILURE*, or *ABORTED*

Return type *OperationState*

static isnumeric(*value*)
Return true if the value is an integer or float type

shutdown()
Ask the Operation to shut down and stop running. How quickly it does so depends on the implementation.

is_shutdown()
Returns True if this operation was asked to shutdown().

cleanup(*incoming_data*, *actual_results*)
Override this method to perform any cleanup actions once the sequence has stopped running. Returns nothing. Check *self.is_shutdown()* to determine if *shutdown()* was called. Check *self.get_status()* for the main return status.

This method should NOT attempt to acquire tools.

Parameters

- **incoming_data** (*dict(str,str)*) – incoming data used for running this operation
- **actual_results** (*dict(str,str)*) – dictionary for ACTUAL output results

set_status(*status*)
Set the current status of this operation.

Parameters **status** (OperationState) – the current status

get_status()
Return the current operation status.

Returns operation's status

Return type *OperationState*

save_result(*name*, *value*, ***kwargs*)
Save a result from this operation.

Optional named argument 'type' can specify a type for the value, which can override the expected type.

Parameters

- **name** (*str*) – the name of the value to store
- **value** (*object*) – the value data to store

get_results()
Return a dictionary of the results from this operation, indexed by the result name. These results are only considered valid if the operation was run and completed without errors. This dictionary only contains names and a dictionary containing the raw data in the 'value' key, and an optional type in the 'type' key, but not the extended type information or value information from the Operations.json file.

Returns operation results

Return type dict(str, dict)

checkout_tool(*tool_id*, *blocking=True*)
Request a tool by name from the tool box. Name is the 'tool_id' field from the station's tool manifest, `tool_manifest.json`

Parameters

- **tool_id** (*str*) – unique tool_id from the manifest file
- **blocking** (*bool*) – True to wait on a busy tool until it becomes available, else return None if in use or tool is offline

Returns 'object', a *Tool* instance, or None if blocking=False and named tool is already in use or is offline

Return type Tool|None

Raises

- ***ToolNotExistsException*** – if no tool known by that name
- ***ToolLockException*** – if internal error locking tool
- ***AbortException*** – if shutdown request detected while blocked

return_tool(*tool_obj*)
Return a tool object to the tool box.

Parameters **tool_obj** (Tool) – *Tool* object returned from *checkout_tool()*

Returns None

Raises

- ***ToolNotExistsException*** – if named tool could not be found
- **KeyError** – if internal object error
- ***ToolLockException*** – if tool found, but failed to unlock

ui_log(*message*)
Write a message on the front page log

Parameters **message** –

set_runtime_data(*runtime_data*)
Store the sequencer runtime data in operation.

Parameters **runtime_data** (*dict*) – the runtime description data

get_runtime_data()
Return the sequencer runtime data

get_loop_iteration()
If this is a node in a loop, return the current loop iteration count, else None

5.2 stationexec.sequencer.handlers module

class stationexec.sequencer.handlers.**SequenceStartHandler**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

This endpoint handler is used to request a sequence start.

launch_sequence = None

initialize(***kwargs*)
Prepare to handle endpoint operation

post()
Request execution of a sequence

class stationexec.sequencer.handlers.**SequenceStopHandler**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

This endpoint handler is used to stop a running sequence of operations.

terminate_sequence = None

initialize(***kwargs*)
Prepare to handle endpoint operation

post()
Request termination of sequence

class stationexec.sequencer.handlers.**SequenceStatusHandler**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

This endpoint handler is used to request information on the currently running sequence of operations.

sequence_status = None

initialize(***kwargs*)
Prepare to handle endpoint operation

get()
Write JSON encoded string of the sequence executed status

5.3 stationexec.sequencer.loop_condition module

stationexec.sequencer.loop_condition.**compare**(*a*, *b*)
Compare operand a and b, which are assumed to be of the same type. If either is None, return None, indicating comparison could not be performed.

Parameters

- **str, float] a** (*Union[int,*) – 1st operand to compare
- **str, float] b** (*Union[int,*) – 2nd operand to compare

Returns -1,0,1

Return type int

Raises **ValueError** – if types are not compatible

stationexec.sequencer.loop_condition.**is_float_string**(*op*)
Return True if op is a string and can be converted to a float value.

stationexec.sequencer.loop_condition.**is_int_string**(*op*)
Return True if op is a string and can be converted to an int value.

stationexec.sequencer.loop_condition.**is_hex_string**(*op*)
Return True if op is a string which can be converted to an int value using hex().

stationexec.sequencer.loop_condition.**convert_value_to_type**(*value*, *convert_to_type*)
Convert value to be compatible with the specified type, if possible. If not possible, then returns None. For example, trying to convert a string like "hello" to an int will return None.

Supported types are int, str, float, (hex, as sub-case of int, with leading 0x)

Unsupported are complex, boolean

Parameters

- **value** (*Union[int,str,float]*) – the value to convert
- **convert_to_type** (*type*) – the python type to convert value to

Returns the converted value, or None on conversion error

Return type Union[int,str,float,None]

class stationexec.sequencer.loop_condition.**LoopConditionOperand**(***kwargs*)
Bases: object

is_range()

get_range()

get_operand()

class stationexec.sequencer.loop_condition.**LoopRepeatOperand**(***kwargs*)
Bases: object

This class hold a loop repeat count object.

increment()

reset()

static is_range()

static get_range()

get_operand()

stationexec.sequencer.loop_condition.**eval_log**(*func*)
A decorator which logs the result of every call to LoopCondition.eval().

Parameters **func** (*function*) – the wrapped function

Returns the wrapping function

Return type function

class stationexec.sequencer.loop_condition.**LoopCondition**(*loop_type*, ***kwargs*)
Bases: object

This class holds a loop condition.

eval(*f_get_op_result*, *operations*)

value_is_result(*operations*)
Does the condition value look like a result from another Operation? If so, does the specified Operation generate the specified result?

Parameters **operations** (*dict*) – Operations in this station

Returns True if a known Operation result

operand_is_result(*operations*)
Does the condition operand look like a result from another Operation? If so, does the specified Operation generate the specified result?

Parameters **operations** (*dict*) – Operations in this station

Returns True if a known Operation result

value

comparison

operand

reset()

5.4 stationexec.sequencer.operationloop module

class stationexec.sequencer.operationloop.**OperationLoop**(***kwargs*)
Bases: object

Store information about a loop declared in the Operation sequence.

is_repeat()

is_precheck()

is_postcheck()

loop_id

loop_type

condition

member_index_list

member_id_of_idx(*idx*)
Return the Operation id of the specified index operation in this loop, else None

member_idx_of_id(*op_id*)
Return the index order of the specified Operation id, or None if not in this loop

first_idx_in_loop

last_idx_in_loop

first_id_in_loop()
Return the Operation id of the first Operation in the loop.

last_id_in_loop()
Return the Operation id of the last Operation in the loop.

5.5 stationexec.sequencer.operationstates module

These are the valid non-percentile states that an *Operation* can be in.

class stationexec.sequencer.operationstates.**OperationState**

Bases: enum.IntEnum

IDLE has not been run yet, i.e. 0% completed

RUNNING is now running, i.e. 1% done

COMPLETED completed, i.e. 100%

WAITING_ON_TOOL waiting to check out a busy tool

REQUEUE data or tools are missing, requeue and re-run later

ERROR exception/error, do not go on with dependent operations

RESULTFAILURE result boundary failure

ABORTED asked to shut down

IDLE = 0

RUNNING = 1

COMPLETED = 100

WAITING_ON_TOOL = 105

REQUEUE = 120

ERROR = 130

RESULTFAILURE = 140

ABORTED = 200

CHAPTER

SIX

STATION

6.1 stationexec.station.data_storage module

The DataStorage object serves as an abstraction layer for the storage and retrieval of data.

6.1.1 Sources

Each event registration requires a source - an identifier that lets DataStorage know what is handling the events. Sources for event handling (storage and retrieval) can be a) tools b) anything else.

To specify a tool, register the handler when the tool object is created and use the tool_id as the source. When an event triggers, DataStorage can check out this tool to handle the event to not interfere with any other processes that may be needing that same tool

For all other sources (anything that is not a tool), set the source to None. This lets DataStorage know that the handler method may be called at any time.

6.1.2 Storage

The storage set of events refer to key times in stationexec operation where something occurs that should be saved.

Events

To register a handler for a storage event, call `register_for_storage_event` with a source (defined above), the specific event (see the StorageEvents enum), and a method that will be performing the action. Storage events can have an unlimited number of handlers. Registration is only allowed before the storage worker is launched in `initialize`.

A storage event is triggered by a call to `trigger_storage_event` with the event enum and a dict containing all data that could be material to the event in question. Storage event triggers will come from stationexec when the appropriate event happens. The sections below on the Work Queue and the Storage Worker describe in detail how these events are processed.

Members of enum `StorageEvents`:

- ON_SEQUENCE_START
- ON_SEQUENCE_END
- ON_OPERATION_START
- ON_OPERATION_END
- ON_RESULT_STORE

- ON_DATA_STORE
- ON_REGISTER_STATION
- ON_UPDATE_STATION
- ON_CREATE_USER
- ON_UPDATE_USER
- ON_LOG_DATA
- ON_MAINTENANCE_EVENT

Work Queue

Storage events are not processed immediately when they are triggered. Since there are no needed results from a store operation and the operation could be slow, the events are processed asynchronously. When a storage event is triggered, each registered handler for the event is placed into a queue to be processed later.

The queue is organized by data source so that, if applicable, multiple storage events from one source can be processed all at once (especially useful when the source is a tool - only have to checkout the tool once to process all of the data). The information stored in the queue is as follows:

- Source - the source that will handle the event
- Event enumeration - which storage event caused this handler to be placed into the queue
- Handling method - the method that will process the event
- JSON formatted string of data - the data that will be processed for storage
- Timestamp of when data was added to queue - this represents the time of the storage event,

in case actual storage doesn't take place until some time later

Storage Worker

The Storage Worker is a thread that will periodically process events in the work queue. It is launched in the `initialize` method of DataStorage. Every 'write_period' seconds, the worker will attempt to execute storage events for each data source. The configurable period of the worker loop is set with the 'write_period' value when creating the DataStorage object.

If the source of a handler is a tool, the worker will check the tool out (otherwise it can process the event directly). The worker will attempt loop through all outstanding data storage events for that source, calling the handler method with the data. If a storage method raises an exception, that object is placed back into the queue to be processed later (in the case that the tool is offline or in use).

6.1.3 Retrieval

The retrieve set of events refer to key times in stationexec operation where previously saved data must be retrieved for analysis, viewing, or processing.

Events

To register a handler for a retrieve event, call `register_for_retrieval_event` with a source (defined above), the specific event (see the RetrievalEvents enum), and a method that will be performing the action. Retrieve events can

have only one handler. If multiple handlers register for a storage event, they overwrite the previous event - only the last handler registered for an event is used. Registration is only allowed before DataStorage `initialize` is called.

A retrieval event is triggered by a call to `trigger_retrieval_event` with the event enum and a dict containing all data that could be material to the event in question. Retrieve event triggers will come from stationexec when the data is needed.

Members of enum `RetrievalEvents`:

- GET_OPERATION_AVERAGE_DURATION
- GET_STATION_DATA
- GET_USER_DATA
- GET_USER_VALID_LOGIN
- GET_ALL_USERS
- GET_LOG_DATA
- GET_MAINTENANCE_DATA

6.1.4 How This Works in StationExec

StationExec provides a default tool/set of tools that register for all storage and retrieval events. Without configuration, this stores to and retrieves data from local SQLite database(s). With configuration, this data can be stored to and retrieved from a local/remote MySQL database.

To extend the behavior by storing to secondary data stores, simply register additional storage handlers for tools that handle your particular storage case (different schema, different database, not a database, HTTP endpoint, text file, etc.)

The default solution can be removed with configurations to make way for a full custom solution. View the 'Event Requirements' section below to ensure that you provide at least all of the necessary handlers.

6.1.5 Event Requirements

There are two requirements for events on developers who are using DataStorage.

First - the following retrieve events must always have a handler:

- GET_STATION_DATA
- GET_USER_DATA
- GET_ALL_USERS

It does not matter if this is the default or a developer-provided handler.

Second - if a retrieve event is implemented, there are corresponding storage events that become required in service of the retrieves. These must be handled by the same source that handles the retrieves.

Retrieve	Store
GET_STATION_DATA	ON_REGISTER_STATION ON_UPDATE_STATION
GET_USER_DATA	ON_CREATE_USER ON_UPDATE_USER
GET_ALL_USERS	ON_CREATE_USER ON_UPDATE_USER
GET_OPERATION_AVERAGE_DURATION	ON_OPERATION_START ON_OPERATION_END
GET_LOG_DATA	ON_LOG_DATA
GET_MAINTENANCE_DATA	ON_MAINTENANCE_EVENT

These requirements are enforced in the `_handler_audit` method, which will raise an exception with error messages describing which events are in violation of the rules.

exception stationexec.station.data_storage.**StorageShutdown**
Bases: exceptions.Exception

class stationexec.station.data_storage.**DataStorage**(*write_period=10*)
Bases: object

initialize()
Make sure required events are covered, load old data, and start worker thread

set_tool_management(*checkout_tool*, *return_tool*)
Set callbacks for managing tools - called after toolbox is enabled

shutdown()
Cleanup

register_event(*source*, *event*, *method*)

trigger_event(*event*, *data*)

6.2 stationexec.station.events module

stationexec.station.events.**register_for_event_group**(*source*, *event_group*, *callback*)

stationexec.station.events.**register_for_events**(*source*, *event_enums*, *callback*)

stationexec.station.events.**register_for_event**(*source*, *event_enum*, *callback*)

stationexec.station.events.**emit_event**(*event_enum*, *data_dict*)

stationexec.station.events.**emit_event_non_blocking**(*event_enum*, *data_dict*)

class stationexec.station.events.**StorageEvents**
Bases: enum.Enum

ON_CUSTOM_STORAGE = 0

ON_SEQUENCE_START = 1

ON_SEQUENCE_END = 2

ON_OPERATION_START = 3

ON_OPERATION_END = 4

ON_RESULT_STORE = 5

ON_DATA_STORE = 6

ON_REGISTER_STATION = 7

ON_UPDATE_STATION = 8

ON_CREATE_USER = 9

ON_UPDATE_USER = 10

ON_LOG_DATA = 11

ON_MAINTENANCE_EVENT = 12

class stationexec.station.events.**RetrievalEvents**
Bases: enum.Enum

GET_CUSTOM_DATA = 0

GET_OPERATION_AVERAGE_DURATION = 1

GET_STATION_DATA = 2

GET_USER_DATA = 3

GET_USER_VALID_LOGIN = 4

GET_ALL_USERS = 5

GET_LOG_DATA = 6

GET_MAINTENANCE_DATA = 7

class stationexec.station.events.**ActionEvents**

Bases: enum.Enum

SHUTDOWN = 0

START_SEQUENCE = 1

STOP_SEQUENCE = 2

EMERGENCY_STOP = 3

EMERGENCY_STOP_CLEAR = 4

RELOAD_TOOL_MANIFEST = 5

RELOAD_SEQUENCE_OPERATIONS = 6

RELOAD_STATION = 7

class stationexec.station.events.**InfoEvents**

Bases: enum.Enum

SHUTTING_DOWN = 0

SEQUENCE_STARTED = 1

SEQUENCE_FINISHED = 2

SEQUENCE_FAILED = 3

SEQUENCE_ABORTED = 4

EMERGENCY_STOP = 5

EMERGENCY_STOP_CLEARED = 6

USER_LOGGED_IN = 7

USER_LOGGED_OUT = 8

UNAUTHORIZED_ACCESS = 9

WEBSOCKET_INCOMING = 10

SEQUENCE_UPDATE = 11

TOOL_UPDATE = 12

MESSAGE_UPDATE = 13

OBJECT_UPDATE = 14

ALERT_UPDATE = 15

POPUP_UPDATE = 16

LOG = 17

SERVER_STARTED = 18

TOOL_COMMAND = 19

STATION_COMMAND = 20

TOOLS_LOADED = 21

SEQUENCE_LOADED = 22

STATION_LOADED = 23

6.3 stationexec.station.handlers module

class stationexec.station.handlers.**StationHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)

get()

class stationexec.station.handlers.**StationStatusHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)

get()

class stationexec.station.handlers.**StationCommandHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)

post()
Send a command to the station.

class stationexec.station.handlers.**StationHandlerSeqSummary**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)

get()

class stationexec.station.handlers.**StationHandlerSeqRecent**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)

get()

CHAPTER

SEVEN

TOOLS

7.1 stationexec.toolbox.tool module

Base class for all tools. Items inheriting will be loaded and interacted with via tool/toolbox.py.

The **kwargs that will be passed in when the class is created will contain all identifying and configuration data from tool_manifest.json. Name the arguments to __init__ accordingly.

The following items must be implemented in a derived class (otherwise the base class will throw a NotImplemented-Error exception):

- initialize
- verify_status
- shutdown
- on_ui_command

These methods will all be called by tool/toolbox.py to setup and interact with the tool.

class stationexec.toolbox.tool.**Tool**(*tool_type*, *name*, *tool_id*, ***kwargs*)

Bases: object

Tool base class that all tools will inherit from

Initialize a Tool object.

Parameters

- **tool_type** (*str*) – name of tool driver
- **name** (*str*) – display name of tool
- **tool_id** (*str*) – unique identifier for tool
- **kwargs** (*dict*) – dictionary containing all values required for initializing tool

initialize()

Perform all actions required to set tool up completely for use.

Returns None

verify_status()

Periodically called (default 5 seconds) to determine if tool is online and healthy. Use as a space to fix the tool if things have gone awry.

Returns None

shutdown()

Perform all actions required to clean up and close tool.

Returns None

on_ui_command(*command*, ***kwargs*)
Handle a request from the UI to perform an action.

Parameters **command** (*str*) – tool command

Returns None

set_status(*status*)

set_online()

set_offline()

listen_for_event(*event*, *callback*)

emit_event(*event*, *data*)

ui_log(*message*)

value_to_ui(*target*, *value*)

get_endpoints()

get_status()

7.2 stationexec.toolbox.toolbox module

stationexec.toolbox.toolbox.**get_tools**(**tools*)
Decorator to be used to checkout tools into a function. Intended for use in station.py primarily. Add the decorator on top of a function with all of the required tool ids as the arguments and the tool references will be passed in an object to the first argument in your function. If the tool is offline or in use, an error will be logged stating so and the function will not be called. When the

e.g.

```
@get_tools("camera", "robot")
def function_needing_tools(tools, other_arg):
    tools.camera.perform_action()
    tools.robot.perform_other_action()
```

class stationexec.toolbox.toolbox.**ToolBox**(*manifest_data*, *debug*, *dev*, *db_data*)
Bases: object

Init the Toolbox object.

Parameters

- **manifest_data** (*list*) – data loaded from tool manifest file
- **debug** (*int*) –
- **dev** (*bool*) –
- **db_data** (*dict*) –

load_tool_modules(*tool_info*)
Load modules for tools into memory

Returns

initialize()
Initialize object

Returns None

initialize_tools(*tools*)

create_tool(*tool_cfg*)
Create an instance of a named tool. The tool_type name must exist in the tools/ directory. The name is displayed to users. The tool_id must be a unique name. Configurations is a dict passed to the tool to configure it, and varies by tool.

Parameters **tool_cfg** (*dict*) – dictionary containing tool_type, name, tool_id, configurations keys

passed to tool driver

Raises ***ToolExistsException*** – if a tool of the same tool_id already exists

shutdown()
Cleanup this and lower objects and prepare to exit.

Returns None

tool_exists(*tool_id*)
Check if the named tool exists.

Parameters **tool_id** (*str*) – unique tool identifier

Returns True if a tool with the specified name exists, otherwise False

Return type bool

tool_status_listener()
Log and process incoming status updates from the tools

Returns

get_status()
Return a dictionary with the status of every tool in the toolbox. No particular order is returned.

Returns dictionary with status and info on all known tools

Return type dict(str,str)

get_tool_status(*tool_id*)
Get the online and in_use status of a particular tool

Parameters **tool_id** (*str*) – ID of tool

Returns (bool) online, (bool) in_use

get_tool_list()

get_tools_for_ui()

Returns

get_endpoints()

checkout_tool(*process*, *tool_id*)
If tool is available, return instance of tool for use and lock tool until return_tool() is called.

Parameters

- **process** (*str*) – optional info on who/what is using the tool
- **tool_id** (*str*) – unique tool identifier

Returns tool instance, if tool is available and not locked.

Raises

- ***ToolNotExistsException*** – if tool_id does not exist
- ***ToolUnavailableException*** – if tool_id is offline
- ***ToolInUseException*** – tool in use by another process

return_tool(*process*, *tool_obj*)
Release the lock on the tool checked out using checkout_tool() after done using.

Parameters

- **process** (*str*) – process that is returning the tool
- **tool_obj** – tool object from checkout_tool()

Returns None

verify_status_loop(***kwargs*)

7.3 stationexec.toolbox.asynctoolbase module

This is a general Asynchronous communication handling class. Classes that connect to tools via Serial or TCP can inherit from this class.

class stationexec.toolbox.asynctoolbase.**AsyncToolBase**(*message_processor=None*, ***kwargs*)

Bases: *stationexec.toolbox.tool.Tool*

A communication class to base comm tools on.

initialize()

verify_status()

set_status(*status_bool*)

shutdown()

send(*data*)
Send data to active connection

Parameters **data** –

Returns

receive(*wait=False*, *after=None*)
Receive next piece of data available from connection

If wait is false, oldest available data is returned immediately, if no data, return None If wait is true, if no data is available, wait until there is data to return If after is defined, the data must be received after the specified time

Parameters

- **wait** (*bool*) –
- **after** (*datetime*) –

Returns

send_receive(*data*, *clear_buffer=False*)
Send a packet and wait for a response that comes after the sending

Parameters

- **data** (*str*) –
- **clear_buffer** (*bool*) –

Returns

clear_rx_buffer()

Returns

run_on_connect(*method*, *delay=0*)

on_ui_command(*command*, ***kwargs*)

class stationexec.toolbox.asynctoolbase.**AsyncMessage**(*msg_id*, *data*)
Bases: object

class stationexec.toolbox.asynctoolbase.**AsyncTool**(*delimiter=None*, *no_delimiter=False*, *timeout=1*, ***kwargs*)
Bases: object

An communication class to base comm tools on.

Initialize object.

One of the following keyword arguments should be specified as the connection method for the device:

- com - com port of device
- serial - serial id of device
- vidpid - VID/PID of USB device
- host,port - the hostname name or ip and the port on the host

Optional arguments for both types

- timeout - default second timeout for operations and retries
- delimiter - end of data line marker - if None, will look for

, , and

Optional additional keyword arguments for serial devices:

- baud - option baud rate for serial devices, default 115200
- parity - optional parity for serial devices, default N(one)
- stopbits - optional stop bits for serial devices, default 1
- poll_seconds - how often to poll serial device, default 0.1 sec

param dict kwargs keyword arguments, see above

initialize()
Begins the process of trying to connect to the serial/ip tool. Connection will run in a loop for the life of the tool

Returns None

shutdown()
Safely close all open serial/ip connection resources

Returns None

set_status_callback(*callback*)
Set the callback that allows this object to affect the status of a tool

Parameters **callback** –

Returns

set_processor_callback(*callback*)
Set the callback that this object can use to pass parsed messages for processing

Parameters **callback** –

Returns

close_on_error()
Close all serial/ip connection resources in event of a connection or communication error to prepare for a new attempt at connection

Returns None

on_close()
Invoked when the TCP stream is closed, whether deliberate or unexpected.

Returns None

run_on_connect(*method*, *delay=0*)
Register callback functions to be invoked after all successful connect/reconnect attempts

Parameters

- **method** (*function*) – Handle to the method
- **delay** (*int*) – How long to wait after connection to call the method

Returns None

connection(***kwargs*)
Calls the methods that check for connectivity every 5 seconds.

Coroutine that runs indefinitely - launched by *initialize*

Returns None

connection_serial(***kwargs*)
Establish connection to serial port

Called by *connection* - if the current connection is not valid, do connect; otherwise, return.

Returns None

connection_ip(***kwargs*)
Establish TCP connection

Called by *connection* - if the current connection is not valid, do connect; otherwise, return.

Returns None

on_connect()

Called after every successful connect/reconnect. If connection is asynchronous, spawn the stream reading methods. Then, call all of the 'on_connect' methods that were set in *run_on_connect*

Returns None

write_async(***kwargs*)

Callback for asynchronous sending. Launched in *write*. Will send all available data and exit

In the case of an invalid connection, the method returns and will be spawned again in the event of a successful reconnect.

Returns None

read_serial(***kwargs*)

Indefinite callback for asynchronous serial port reading. Launched in *on_connect* on all successful connect/reconnect attempts. After 'poll_seconds' time elapses, attempt to read from the serial port.

In the case of an invalid serial connection, the method returns and will be spawned again in the event of a successful reconnect.

Returns None

read_ip(*raw_data*)

Called when new data appears on the TCP stream. Set as the stream read method in *on_connect*.

In the case of an invalid serial connection, the method returns and will be spawned again in the event of a successful reconnect.

Parameters **raw_data** (*bytes*) – Raw data read from the TCP stream for processing

Returns None

write(*data*)

Add a message to the outgoing messages queue

Parameters **data** (*str*) – message to be written

Returns

read(*wait=True*, *after=None*)

Cleanup rx queue and return oldest message; return None if no data available

If wait, wait until timeout expires for available data If after, message must be received after specified time

Parameters

- **wait** (*bool*) –
- **after** (*datetime*) –

Returns data string or None

parse_rx_data(*raw_data*)

Parse incoming data into individual data packets

Parameters **raw_data** –

Returns

clear_rx_queue()

Empty the RX queue

Returns None

7.4 stationexec.toolbox.camerabase module

This is a camera class to extend the base Tool class. This restricts the Tool interface further with camera specific functionality. This class should be used as the base for creating any camera driver.

Adds the required methods:

- capture_image
- set_exposure

And the optional methods:

- begin_video_stream
- end_video_stream

class stationexec.toolbox.camerabase.**CameraBase**(*frames_per_second=15*, *stream_image_width=460*, *stream_image_height=329*, ***kwargs*)

Bases: *stationexec.toolbox.tool.Tool*

get_endpoints()

capture_image()
Capture and return camera image

Returns None

get_latest()

get_latest_raw()

set_exposure(*exposure*)
Set the camera exposure value

Returns None

start_capture(*message='Capturing'*)
Start video/constant capture on the camera

Parameters **message** (*str*) – Optional status message to indicate camera in capture mode

Returns None

stop_capture()
Stop the video/constant capture

Returns None

begin_video_stream()

Start the camera capturing and launch the 'run' loop that grabs, processes, and caches the latest image from the camera for streaming to the UI or for ease of grabbing the latest image

Returns None

end_video_stream()
Terminate video streaming

Returns None

run(***kwargs*)
Routine spawned by begin_video_stream that runs continuously to grab latest image from camera and cache it. Exits on error or on call to end_video_stream. Can check status of loop with boolean self.is_streaming

Returns None

capture_stream_image()
Capture and return camera image for video stream

Returns None

set_stream_color_mode(*color_mode*)

save_image(*image_data*)
Save copy of the raw image with a timestamp for later use

Parameters **image_data** –

Returns

process_image()
Process saved raw image for streaming

Returns

initialize()

verify_status()

shutdown()

on_ui_command(*command*, ***kwargs*)

class stationexec.toolbox.camerabase.**CameraStreamHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

get_latest = None

no_image_yet = None

fps = None

initialize(***kwargs*)
Prepare to handle endpoint operation

get(**args*, ***kwargs*)

7.5 stationexec.toolbox.handlers module

class stationexec.toolbox.handlers.**ToolCommand**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

post()

Send a command to a tool. Command to tool is placed in the JSON body. Command contains the target (tool_id) and command with arguments.

class stationexec.toolbox.handlers.**ToolboxStatus**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize(***kwargs*)
Prepare to handle endpoint operation

get (*parameter=None*)
Write JSON encoded string containing status of all tools

class stationexec.toolbox.handlers.**ToolboxUIFrame** (*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize (***kwargs*)
Prepare to handle endpoint operation

get (*parameter=None*)
Provide the tool control UI main HTML

class stationexec.toolbox.handlers.**ToolboxUI** (*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

initialize (***kwargs*)
Prepare to handle endpoint operation

get (*tool_id=None*)
Render and display requested UI of an available tool

7.6 stationexec.toolbox.tool_launch module

The ToolLaunch class is purely a developer-focused module. The methods are intended to make development and debugging of tools in a context similar to deployment much simpler.

This is instantiated and launched from the cli tool launcher utility

class stationexec.toolbox.tool_launch.**ToolLaunch**
Bases: object

Helper class to ease tool development

build_tool_dict (*tool_type*, *name*, *tool_id*, *configurations*)
Build the correctly formatted dictionary to configure the tool completely with this call.

Parameters

- **tool_type** – name of the tool folder and file
- **name** – display name for the tool - the friendly name
- **tool_id** – the unique identifier that is used to reference the tool
- **configurations** (*dict*) – all required configuration values

Returns complete tool configuration data

Return type dict

tool_setup (*tool_object*, *tool_config_dictionary*)
Prepare the tool object by instantiating with the proper config data

Parameters

- **tool_object** (*class*) – the un-instantiated tool class
- **tool_config_dictionary** (*dict*) – output from *build_tool_dict*

Returns None

static tool_command_build (*delay*, *command*, **args*, ***kwargs*)
Helper function to build a command tuple. Commands are optional helpers that can call any tool function

at some later point in the future. This is helpful if you want to debug a function without dealing with the web server or ui.

Parameters

- **delay** (*float*) – how many seconds to wait before invoking
- **command** (*method*) – the method to invoke
- **args** (*list*) – un-named arguments
- **kwargs** (*dict*) – keyword arguments

Returns formatted command request

Return type tuple

setup_tool_server(*port=8888*)

Create a web server at specified port to serve the tool web page and allow tool control messages to be received.

Parameters **port** (*int*) – web port to serve on

Returns None

tool_command(*tool_id*, *cmd*)

Called from the *ToolCommand* handler. Parses arguments from the web request and calls the `on_ui_command` method of the tool with the request

Parameters

- **tool_id** (*str*) – ID of the tool
- **cmd** (*str*) – JSON formatted string from UI

Returns None

tool_status_listener()

Log and process incoming status updates from the tools

tool_run(*port=8888*, *polling_period=5*)

Start the tool running. Schedules all (if any) commands defined above to run at the appropriate times. Starts a server at the port requested (default port is 8888), and invokes the tool in a similar context to what it runs in during deployment. With the server, you can access the tool UI at localhost:<port> and use that to send commands to the tool.

Navigate to localhost:<port>/shutdown to shutdown.

Parameters

- **port** (*int*) – Port to start server on
- **polling_period** (*int*) – Period in seconds between checking for connectivity

Returns None

shutdown()

class stationexec.toolbox.tool_launch.**MainHandler**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

Serve tool UI

util = None

initialize(***kwargs*)

Prepare to handle endpoint operation

get()

class stationexec.toolbox.tool_launch.**ShutdownHandler**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

Exit tool context

util = None

initialize(***kwargs*)

Prepare to handle endpoint operation

post()

class stationexec.toolbox.tool_launch.**ToolCommand**(*application*, *request*, ***kwargs*)

Bases: *stationexec.web.handlers.ExecutiveHandler*

Handles incoming requests to send command to a tool

util = None

initialize(***kwargs*)

Prepare to handle endpoint operation

post()

Send a command to a tool. Command to tool is placed in the JSON body. Command contains the target (tool_id) and command with arguments.

7.7 stationexec.toolbox.tool_utilities module

exception stationexec.toolbox.tool_utilities.**ToolNotFound**(*message*)

Bases: exceptions.Exception

stationexec.toolbox.tool_utilities.**find_tool_by_name**(*tool_name*)

Parameters **tool_name** –

Returns

stationexec.toolbox.tool_utilities.**load_tool_object**(*tool_name*)

Parameters **tool_name** –

Returns

stationexec.toolbox.tool_utilities.**list_all_available_tools**()

Returns

CHAPTER

EIGHT

AUTHENTICATION

8.1 stationexec.authentication.auth module

This file implements the access level decorator that limits access to endpoints.

stationexec.authentication.auth.**restrict_access**(*minimum_level_to_access*)
Decorator to ensure user is logged in with acceptable access level.

Valid levels are those in the *AuthLevel* class.

8.2 stationexec.authentication.authlevel module

These are the valid list of access levels for users in this application.

For Python 3, see read_only_properties module to simplify this implementation.

class stationexec.authentication.authlevel.**AuthLevel**
Bases: enum.IntEnum

This class defines all valid access levels for users, as well as NONE which means no access.

NONE = 0

OPERATOR = 1

TECHNICIAN = 2

ADMINISTRATOR = 3

ROOT = 4

8.3 stationexec.authentication.login module

class stationexec.authentication.login.**LoginHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

station_info = None

initialize(***kwargs*)

get()
Write simple HTML login page

post()
Create a new unique token and set cookie 'user' if the login is valid, otherwise send back to /login to try again

8.4 stationexec.authentication.logout module

class stationexec.authentication.logout.**LogoutHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

executive = None

initialize(***kwargs*)
Prepare to handle endpoint operation

get()
Clear the login cookie 'user', and delete token if known.

8.5 stationexec.authentication.tokenmanager module

This file implements browser tokens and stores logged in user information.

stationexec.authentication.tokenmanager.**create_token**(*auth_level*, *userid*, *username*, *fullname*)
Helper function to create a random 32 character token encoded as Base64, and add it to the valid token internal list.

Returns the new token id and the token in a tuple. You can use get_token() with token id to fetch the token.

Parameters

- **auth_level** (*int*) – access level of user for this token
- **userid** (*int*) – user UUID for this user
- **username** (*str*) – username for this user
- **fullname** (*str*) – full name for this user

stationexec.authentication.tokenmanager.**destroy_token**(*token*)
Remove a token from the valid tokens internal list.

Parameters **token** (*int*) – token to destroy

stationexec.authentication.tokenmanager.**get_token**(*token_index*)
Return the token from the internal list, at the specified index.

Parameters **token_index** (*int*) – token index

stationexec.authentication.tokenmanager.**valid_token**(*token*)
Returns true if the specified token is in the internal valid token list.

Parameters **token** (*str*) – the current_user token

stationexec.authentication.tokenmanager.**get_access_level_for_token**(*token*)
Return the access level of the user stored in the token, if valid, else return 0.

Parameters **token** (*str*) – the current_user token

stationexec.authentication.tokenmanager.**get_userid_for_token**(*token*)
Return the userid of the user stored in the token, if valid, else return 0.

Parameters **token** (*UUID*) – the current_user token

stationexec.authentication.tokenmanager.**get_username_for_token**(*token*)
Return the username of the user stored in the token, if valid, else return None.

Parameters **token** (*str*) – the current_user token

stationexec.authentication.tokenmanager.**get_userfullname_for_token**(*token*)
Return the full name of the user stored in the token, if valid, else return None.

Parameters **token** (*str*) – the current_user token

8.6 stationexec.authentication.userauthinfo module

class stationexec.authentication.userauthinfo.**UserAuthInfo**
Bases: object

static check_password(*password*, *how*, *old_password*)
Check if user password matches existing password.

Parameters

- **password** (*str*) – text of password to check
- **how** (*str*) – which method to use to encrypt
- **old_password** (*str*) – encrypted password (with embedded salt) to compare to

Returns True if password matches

Return type bool

Raises **RuntimeError** – if unknown encryption method in 'how'

static encrypt_password(*password*, *how*)
Encrypt a user password with a new salt.

Parameters

- **password** (*str*) – text of password to encrypt
- **how** (*str*) – which method to use to encrypt

Returns encrypted password string

Return type str

Raises **RuntimeError** – if unknown encryption method in 'how'

static encryption_type()
Return an identifier for the current type of password encryption being used in this object.

CHAPTER

NINE

CLI UTILITIES

9.1 stationexec.cli.cli_tools module

9.1.1 CLI Utilities

cli_hello

cli_setup

cli_which

stationexec.cli.cli_tools.**cli_hello**()
CLI utility to run the 'hello_prpl' demo station

Returns

stationexec.cli.cli_tools.**cli_setup**()

Returns

stationexec.cli.cli_tools.**cli_start**()
CLI utility to start stationexec

Returns

stationexec.cli.cli_tools.**cli_station**()

stationexec.cli.cli_tools.**cli_tool**()

stationexec.cli.cli_tools.**cli_which**()

Returns

9.2 stationexec.cli.common module

stationexec.cli.common.**prompt_for_text**(*message*, *default=None*, *filter=None*)

Parameters

- **message** –
- **default** –
- **filter** –

Returns

stationexec.cli.common.**prompt_for_list**(*message*, *choices*, *filter=<function <lambda>>*)

Parameters

- **message** –
- **choices** –
- **filter** –

Returns

stationexec.cli.common.**prompt_for_bool**(*message*, *default=False*)

Parameters

- **message** –
- **default** –

Returns

stationexec.cli.common.**create_init_files**(*location*)

Parameters **location** –

Returns

stationexec.cli.common.**create_folder**(*location*)

Parameters **location** –

Returns True if folder already exists, False if new folder created

stationexec.cli.common.**create_from_template**(*file_name*, *repl*, *new_name*)

9.3 stationexec.cli.for_stations module

stationexec.cli.for_stations.**for_stations**(*template_path*, *destination_path*)

Parameters

- **template_path** –
- **destination_path** –

Returns

9.4 stationexec.cli.for_tools module

stationexec.cli.for_tools.**for_tools**(*template_path*, *destination_path*)

Parameters

- **template_path** –
- **destination_path** –

Returns

stationexec.cli.for_tools.**interactive_tool**()

stationexec.cli.for_tools.**interactive_tool_ui**()

CHAPTER

TEN

BUILT IN TOOLS

A set of tools built-in to Station Executive that will be useful to most users.

10.1 stationexec.built_in.exampletool module

Example tool to demonstrate the construction of a tool

10.2 stationexec.built_in.exampletool2 module

Second Example tool to demonstrate the construction of a tool

10.3 stationexec.built_in.station_storage module

Tool that implements the Flexible Data Layer for all data storage in the system run; writes all data to local SQLite file database, configurable to use MySQL

CHAPTER

ELEVEN

LOGGER

Assisting methods to capture log information and store as required.

11.1 stationexec.logger.log module

This file implements a method to publish log messages of varying severities.

Other files should import this file, then make calls like:

```
from stationexec.logging import log
log.info("This is a message")
```

class stationexec.logger.log.**LogKind**

Bases: enum.Enum

The supported kinds of log messages.

DEBUG = 'debug'

INFO = 'info'

WARNING = 'warning'

ERROR = 'error'

EXCEPTION = 'exception'

USAGE = 'usage'

DISPLAY = 'display'

stationexec.logger.log.**debug**(*level*, *message*)

Publish a debug message.

Parameters

- **level** (*int*) – debug verbosity level of message, beginning at 1; 99 can be used during development to always display the debug message, but should not be left in the code
- **message** (*str*) – text of log message

stationexec.logger.log.**info**(*message*)

Publish an informational level message.

Parameters **message** (*str*) – text of log message

stationexec.logger.log.**warning**(*message*)

Publish a warning message.

Parameters **message** (*str*) – text of log message

stationexec.logger.log.**error**(*message*)
Publish an error message.

Parameters **message** (*str*) – text of log message

stationexec.logger.log.**exception**(*message*, *e*)
Publish an exception error event.

Parameters

- **message** (*str*) – text of log message
- **e** (*Exception*) – exception that occurred

Returns the exception passed in

11.2 stationexec.logger.logger module

class stationexec.logger.logger.**Logger**
Bases: *stationexec.utilities.singleton.Singleton*

A class which processes log messages.

station_name = None

init(*station_name*, *debug*)
'initialize' instead of '__init__' to fit the singleton pattern

__init__ is called every time the class is referenced, whereas init is only called explicitly.

log_message(*message*, *stream*, *debug_level*, *data*, *stack_trace*, *pid*, *caller*, ***kwargs*)

CHAPTER

TWELVE

UTILITIES

12.1 stationexec.utilities.byte_conversion module

stationexec.utilities.byte_conversion.**to_bytes**(*seq*)

Convert a sequence to a bytes type

Adapted from pyserial serialutil.py/to_bytes

Parameters **seq** –

Returns

12.2 stationexec.utilities.classproperty module

class stationexec.utilities.classproperty.**ClassProperty**

Bases: property

Support definition of class-level properties in static classes.

see https://stackoverflow.com/a/1383402/1663987

12.3 stationexec.utilities.colors module

This class defines ANSI color codes to change the cursor color

End with ENDC, which turns off all colors and modes.

Example:

```
import stationexec.colors
print(Colors.WARNING + "Warning: something went wrong" + Colors.ENDC)
```

class stationexec.utilities.colors.**Colors**

Bases: object

HEADER = '\x1b[95m'

OKBLUE = '\x1b[96m'

OKGREEN = '\x1b[92m'

WARNING = '\x1b[93m'

ERROR = '\x1b[91m'

```
INFO = '\x1b[45m'
ENDC = '\x1b[0m'
BLINK = '\x1b[5m'
NOBLINK = '\x1b[25m'
BOLD = '\x1b[1m'
NOBOLD = '\x1b[22m'
UNDERLINE = '\x1b[4m'
NOUNDERLINE = '\x1b[24m'
```

12.4 stationexec.utilities.config module

Config.py contains helpers for loading configuration files and getting the correct paths and parameters for navigating the program.

exception stationexec.utilities.config.**RootDoesNotExist**
Bases: exceptions.Exception

stationexec.utilities.config.**set_station_identity**(*identity*)

stationexec.utilities.config.**get_system_config**()

stationexec.utilities.config.**get_all_paths**()
Find and cache all important paths for stationexec

Changes the working directory to the discovered app root path

Returns

stationexec.utilities.config.**verify_paths_exist**()

stationexec.utilities.config.**get_default_paths**()

stationexec.utilities.config.**format_name**(*name_in*)
Reformat name_in to adhere to StationExec convention - lowercase, separated by underscore - and generate a class name and display name version

e.g. name_in = Strong-bad name_out = strong_bad class_name = StrongBad display_name = Strong Bad

Parameters **name_in** (*str*) – string name to be reformatted

Returns name_out, class_name, display_name

Return type tuple

stationexec.utilities.config.**merge_config_data**(*input_arguments*)
Merge data from command line arguments, input configuration file, and default system configuration file into one config dictionary

Parameters **input_arguments** –

Returns

stationexec.utilities.config.**load_config**(*config_file_path*, *config_file_name=None*)
Load the specified JSON config_file_name from the specified config_file_path

Note: carriage return characters are removed from the input

Parameters

- **config_file_path** (*str*) – file path
- **config_file_name** (*str*) – [optional] name of file located at config_file_path, to append to end of path

Returns JSON read from files

Return type dict

Raise simplejson.JSONDecodeError on JSON syntax errors

stationexec.utilities.config.**remote_path_import**(*module_name*, *module_path*)

Programmatically import module at a particular path

Parameters

- **module_name** (*str*) – name of module file to import (no file extension)
- **module_path** (*str*) – file path

Returns Imported module object

Return type module

12.5 stationexec.utilities.exceptions module

exception stationexec.utilities.exceptions.**StationError**(*module=None*, *code=None*, *message=None*)

Bases: exceptions.Exception

Custom exception to be thrown by Station

make additional arguments optional so that class is pickleable

exception stationexec.utilities.exceptions.**InvalidAttribute**(*message*, *field=None*, *filename=None*)

Bases: exceptions.Exception

Custom exception used when json files are missing required fields or have invalid values

make additional arguments optional so that class is pickleable

exception stationexec.utilities.exceptions.**MissingResult**(*message*, *field=None*)

Bases: exceptions.Exception

Custom exception used when Operation is missing a result

make additional arguments optional so that class is pickleable

exception stationexec.utilities.exceptions.**ToolExistsException**(*message*)

Bases: exceptions.Exception

Exception indicating an attempt to create a tool name that already exists.

exception stationexec.utilities.exceptions.**ToolNotExistsException**(*message*)

Bases: exceptions.Exception

Exception indicating an attempt to check out a tool name that does not exist.

exception stationexec.utilities.exceptions.**ToolLockException**(*message*)
Bases: exceptions.Exception

Exception indicating an attempt to lock or unlock a tool failed fatally.

exception stationexec.utilities.exceptions.**ToolUnavailableException**(*message*)
Bases: exceptions.Exception

Exception indicating an tool offline and not available for use.

exception stationexec.utilities.exceptions.**ToolInUseException**(*message*)
Bases: exceptions.Exception

Exception indicating a tool is in use by another process and not available.

exception stationexec.utilities.exceptions.**AbortException**
Bases: exceptions.Exception

Exception indicating that a Operation has detected a shutdown request. Has no message.

12.6 stationexec.utilities.gettool module

class stationexec.utilities.gettool.**GetTool**(*toolbox*, *tool_id*, *process*)
Bases: object

Context manager for temporarily checking out a tool.

Checkout out the named tool in a context.

Returns *Tool* object

Return type *Tool*

class stationexec.utilities.gettool.**GetToolRetry**(*toolbox*, *tool_id*, *process*, *retries=3*, *wait=3*)
Bases: object

Context manager for temporarily checking out a tool.

Checkout out the named tool in a context.

Returns *Tool* object

Return type *Tool*

Raises **RuntimeError** – if times out waiting for tool to be available

12.7 stationexec.utilities.result_references module

stationexec.utilities.result_references.**looks_like_result_reference**(*value*)
Does the result string have the form Operation:ResultName?

Parameters **value** (*str*) – string to check

Returns True if it could be a result reference

Return type bool

stationexec.utilities.result_references.**reference_parse_parts**(*reference*)
Split a result reference to another Operation's result into the component parts: i.e. the Operation name and the result name.

Parameters **reference** (*str*) – the result reference

Returns the operation name, the result name

Return type Union[List[str],None]

Raises **ValueError** – if reference is not a string or is not formatted as a reference

stationexec.utilities.result_references.**operation_has_result**(*operation_list*, *reference*)

If the given reference appears to be of the form Operation:ResultName, then determine if the named Operation exists and generates the specified ResultName.

Parameters

- **operation_list** (*dict*) – operations dict
- **reference** (*str*) – result reference string

Returns True if Operation generates Result

Return type bool

12.8 stationexec.utilities.shutdown module

stationexec.utilities.shutdown.**install_signal_handlers**(*self*)

stationexec.utilities.shutdown.**signal_list**()

Return a list of signals to be caught on this OS

stationexec.utilities.shutdown.**process_name**(*pid=None*)

Return the process name (i.e. cmdline) for the specified PID, or the current process if pid=None

stationexec.utilities.shutdown.**sig_handler**(*self*, *signum*, *frame*)

catch a signal and shut down the station exec and all threads

stationexec.utilities.shutdown.**stop_loop**()

12.9 stationexec.utilities.singleton module

class stationexec.utilities.singleton.**Singleton**

Bases: object

init(**args*, ***kwargs*)

Override this as needed.

12.10 stationexec.utilities.time module

stationexec.utilities.time.**local_to_utc**(*dt*)

stationexec.utilities.time.**get_local_time**()

stationexec.utilities.time.**get_utc_now**()

stationexec.utilities.time.**to_timestamp**(*dt*, *is_local_time=False*)

stationexec.utilities.time.**to_datetime**(*ts*, *local=False*)

12.11 stationexec.utilities.uuidstr module

stationexec.utilities.uuidstr.**get_uuid**()

stationexec.utilities.uuidstr.**uuid2str**(*auuid*)
Return a representation of a UUID for storage into a database

Parameters **auuid** (*UUID*) – a UUID object

Returns a string with the dashes removed, or None if the argument was None

stationexec.utilities.uuidstr.**str2uuid**(*astr*)
Convert a string representation of a UUID from a database into a Python UUID object.

Parameters **astr** (*str*) – a UUID string from a database

Returns a UUID object, or None if the argument was None

CHAPTER

THIRTEEN

WEB

13.1 stationexec.web.handlers module

class stationexec.web.handlers.**ExecutiveHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

Adds additional functionality to RequestHandler.

json_args = None

prepare()
Automatically decode any JSON arguments send to the endpoint, into a variable named json_args.

on_finish()

get_current_user()
Override which returns the cookie used for authentication.

data_received(*chunk*)

class stationexec.web.handlers.**ShutdownHandler**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

post()
Request the Station Executive to shut down

class stationexec.web.handlers.**ExecutiveUI**(*application*, *request*, ***kwargs*)
Bases: *stationexec.web.handlers.ExecutiveHandler*

web_info = None

initialize(***kwargs*)
Prepare to handle endpoint operation

get()
Serve main Station Executive web user interface

class stationexec.web.handlers.**ExecutiveStaticFileHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.StaticFileHandler

Subclass to serve static files from the application. See Tornado source code for subclassing notes: https://fburl.com/pxn4y5pc

URLs of the form /static/filename map to the base/ui folder

URLs of the form /static/station/filename map to the station/<current station type>/ui folder

URLs of the form /static/tool/<toolname>/filename map tp the tool/<toolname>/ui folder

module_path = None

internal_tool_path = None

tool_path = None

station_path = None

debug = None

initialize(*debug*, ***kwargs*)
Prepare to handle endpoint operation

data_received(*chunk*)

parse_url_path(*url_path*)

set_extra_headers(*path*)

classmethod get_absolute_path(*_root*, *path*)
Override base absolute path behavior

validate_absolute_path(*_root*, *absolute_path*)
Custom path resolution to fit project folder structure

class stationexec.web.handlers.**ExecutiveTemplateLoader**(***kwargs*)
Bases: tornado.template.BaseLoader

Subclass to serve templates. See Tornado source code for subclassing notes: https://fburl.com/irmhelhj

Store custom arguments and call base class

resolve_path(*name*, *parent_path=None*)
Custom path resolution to fit project folder structure

13.2 stationexec.web.web_helpers module

stationexec.web.web_helpers.**check_if_client_available**(*host*, *port*)

stationexec.web.web_helpers.**http_get**(*url*, *body*, *decode_json=False*, *configurations=None*)

stationexec.web.web_helpers.**http_post**(*url*, *body*, *decode_json=False*, *configurations=None*)

stationexec.web.web_helpers.**http10_get**(*url*, *body=''*, *decode_json=False*)

stationexec.web.web_helpers.**http10_post**(*url*, *body=''*, *decode_json=False*)

13.3 stationexec.web.websocket module

class stationexec.web.websocket.**StationSocket**(*application*, *request*, ***kwargs*)
Bases: tornado.websocket.WebSocketHandler

Tornado endpoint handler for requesting a new web socket

manager = None

initialize(***kwargs*)
Get station socket ready for operation.

open()

on_message(*message*)

on_close()

class stationexec.web.websocket.**SocketManager**

Bases: object

An object that stores all the sockets that have been opened by clients.

initialize()
Get socket manager ready for operation.

save(*socket*)
Store a new socket.

delete(*socket*)
Remove a socket from the active list.

send_all(***event_data*)
Send a message to all open sockets.

CHAPTER

FOURTEEN

BUILD AND RELEASE

Package version is located in stationexec/version.py Change this value to set the version of the package.

To build the project from source, enter the top level project directory and execute:

```
python setup.py bdist_wheel
```

CHAPTER

FIFTEEN

DOCUMENTATION

CHAPTER

SIXTEEN

GENERATING THE DOCUMENTATION

16.1 Install sphinx

```
sudo apt/yum install python-sphinx
pip install Sphinx
pip install sphinx-rtd-theme
pip install sphinxcontrib-httpdomain
```

16.2 Generate Docs

Navigate to 'docs' folder in source and run

```
make html
```

to generate a new version of the HTML docs. Launch 'index.html' from docs/build/html

For PDF, make sure LaTeX is installed (potentially painful and error prone - consider yourself warned) and generate html first.

```
make latexpdf
```

Find 'StationExecutive.pdf' in docs/build/latex

16.3 Install LaTeX for PDF

Linux Only (Tested on Ubuntu and CENTOS 7) Instructions derived from those at the TeX Users Group Download the ~3GB 'texlive.iso' from the CTAN Mirror

Mount the iso, then run 'sudo ./install-tl' from inside the mounted drive, enter option 'i' when prompted, then wait for completion. Add LaTeX to path - update texlive path, date, and install for your machine as necessary:

16.3.1 Ubuntu

Append the following to ~/.bashrc

```
PATH=/usr/local/texlive/2018/bin/x86_64-linux:$PATH; export PATH
```

16.3.2 Fedora

Create 'texlive.sh' in /etc/profile.d and add the following:

```
PATH=/usr/local/texlive/2018/bin/x86_64-linux:$PATH
export PATH
```

16.3.3 CENTOS - execute as root

```
echo 'pathmunge /usr/local/texlive/2018/bin/x86_64-linux/' /usr/profile.d/texlive.sh
chmod +x /usr/profile.d/texlive/sh
. /etc/profile
```

CHAPTER

SEVENTEEN

RELEASE NOTES

17.1 What's new in Station Exec 0.7.4

17.1.1 May 31, 2019

```
* Release Cleanup
* UI
    * Tool wrapper UI to reduce boilerplate
    * Station UI wrapper to reduce boilerplate
* New queue/thread for handling button presses from UI (lower thread overhead)
* UI commands now come through the websocket for faster processing
* Added a few new events for later use
```

17.2 What's new in Station Exec 0.7.3

17.2.1 May 22, 2019

```
* New Flexible Data Storage methodology
    * Event Based Storage/Retrieval
    * Built-In SQLite/MySQL Tool
* System Updates
    * Permanent Installation Folder
    * Event Infrastructure
    * New CLI Tools
    * Code Cleanup
    * Modularization for Testing
    * Critical Module Access Restriction
    * Removal of Outmoded Ideas
* Tool Updates
    * Overhaul of Check-In/Check-Out Procedure
    * Simplified Tool Requirements
    * Refactor of Tool Creation in Toolbox for Flexibility
* Merging Station into Executive
```

17.3 Station Exec 0.7.0, 0.7.1, 0.7.2

```
* Internal development releases
```

17.4 What's new in Station Exec 0.6.1

17.4.1 February 15, 2019

```
* Minor tune-ups for internal demo
* Added optional case in async tool for parsing data without a delimiter
```

17.5 What's new in Station Exec 0.6.0

17.5.1 January 30, 2019

```
* Minor tune-ups for internal demo
* Added path reference to stations
```

17.6 What's new in Station Exec 0.5.6

17.6.1 December 4, 2018

```
* New ability to send data from tools to UI elements
* Fixed issue with blocking TCP during standalone tool operation
* Ability to manually affect the station status
```

17.7 What's new in Station Exec 0.5.5

17.7.1 November 26, 2018

```
* New Serial Modbus built-in tool
* Stock now returns 'id' in addition to quantity
* Ability to clear all stock and start over
```

17.8 What's new in Station Exec 0.5.4

17.8.1 November 26, 2018

```
* Fixing issues with UI logging
* Fixing issues with stock tracking in sequence
```

17.9 What's new in Station Exec 0.5.3

17.9.1 November 20, 2018

```
* Sequence Looping
* Stock tracking
```

17.10 What's new in Station Exec 0.5.2

17.10.1 October 30, 2018

```
* Fix to the station and tool templates to have more accurate information and to␣
↪generate items that
   are actually executable
* se-gen cli utility creates the necessary python files so that all modules are␣
↪discoverable
* se-gen station creates a bare-bones launcher json file and places it in the working␣
↪directory
```

17.11 What's new in Station Exec 0.5.1

17.11.1 October 17, 2018

```
* Bug fix - station did not implement correct version of a method, which caused a␣
↪crash
* Removed opencv-python from base requirements
```

17.12 What's new in Station Exec 0.5.0

17.12.1 October 15, 2018

```
* Command Line Interface Utilities
    * Huge update to how stationexec works from the user's perspective (much easier!)
    * Launch stationexec, generate tools and stations, debugging tools, and more
* Move to internal repository
    * Reworking of project structure to split stationexec, stations, and tools into␣
↪separate development efforts
    * Numerous style changes - internal repo has aggressive lint and style opinions
* UI Re-architecture
    * Broke up monolithic javascript file that was managing UI view - decentralized␣
↪the UI
    * Now gives slightly better performance and much better extensibility for WAY␣
↪less effort and complexity
    * Much easier to add new pages and navigate more easily
    * By default, UI navigates between Station and Tool control pages
    * Printing of messages from sequence and tools to the UI
```

(continues on next page)

(continued from previous page)

```
* Station Updates
    * Massive improvements based on experience with Cleaning and Subway stations
* Sequence Updates
    * Now can reuse operations multiple times just from operations.json
    * Station calls at beginning and end of sequence more stable
    * Early support for submission of jobs to Algorithm Server
* Tool Updates
    * Running tools moved to a new CLI utility that takes care of path issues
    * Support for synchronous command/response in asynctool
    * Improved Meca500 driver - now supports hardware R2 and R3
    * Improved operation of magnemotion driver
    * Camera base improved drastically and now makes deriving cameras much easier
    * Basler/Cleaning Camera updated - solid streaming, image processing
    * Brought IDS up to date - now also a solid streaming and acquisition camera
    * Added LED Control
* Updated Developer Guide to reflect current status of project - especially to␣
↪incorporate CLI Utilities
* Usability and stability fixes - here are a few:
    * Stationexec no longer checks for dependencies on load - now a manual CLI␣
↪process (much more efficient)
    * Cleaned up log file for standard run so that critical information is more␣
↪easily readable
* Style improvements
```

17.13 What's new in Station Exec 0.4.4

17.13.1 August 14, 2018

```
* Bug fix - two required packages missing from requirements.txt in 0.4.3
* Magnemotion tool in an early, usable state
* Usability and stability fixes - here are a few:
    * Tool UI improvements
        * Consistency improvements in naming of tool UI files
        * Hiding unsupported tabs
        * Faster and more consistent tool status updates
* Style improvements
```

17.14 What's new in Station Exec 0.4.3

17.14.1 August 1, 2018

```
* New database capabilities
    * Tool and table to log exceptions during execution - helpful for tracking␣
↪software issues over time
    * Tool and table to log tool usage - can be used to track maintenance on tools
* Improved DAG visualization
    * New drawing package used that is more flexible
    * More predictable and controllable layout
* Ease of use changes for Developers:
    * Ability to display a nicely formatted table showing individual or group tool␣
↪status
```

(continues on next page)

(continued from previous page)

```
        * Added to all tool UIs
        * Makes adding a table showing the status of a subset of tools to a station␣
↪page easy
    * localonly mode enforced for 'hello_prpl' example project
    * Check for installed packages - a quick check to ensure all required python␣
↪packages are present to aid
        the developer in getting started quickly.
    * Added extra checks in operations for when system is shutting down
    * Added experimental force shutdown mode that terminates operations when stopping␣
↪sequences
* Copious usability and stability fixes - here are a few:
    * More fixes related to the process -> thread change
    * Improvements to database syncing for performance and stability
    * Gave GetTool a non-blocking mode in case we don't want to halt execution on an␣
↪unavailable tool
    * Prevent cameras from launching multiple acquisition loops
    * Allow asynctool to directly set tool status to aid responsiveness in status␣
↪changes
    * Fix sqlite transaction issues plaguing Python 3 runs
    * Only running database sync method when the external database is available
    * Updating built-in jquery to most recent stable version
    * Changing how UUIDs are stored to fix incompatibility between Python 2 and 3␣
↪representation of bytes and strings
* Widespread documentation and style improvements
```

17.15 What's new in Station Exec 0.4.2

17.15.1 July 12, 2018

```
* Changing architecture of stationexec from multi-process to multi-thread
    * BIG change with primarily beneficial results
    * Support for stationexec on Windows is MUCH more stable
        * Multi-processing is poorly supported on Windows - threading works great
        * Many items like threads and sockets would not work (which makes tool␣
↪connections hard) - now they do
        * Python 3 on Windows is back on the table! It behaved differently than 2 in␣
↪multiprocessing - no longer.
    * More complex tools can be supported
        * Tools like cameras that interact with an installed driver on the PC did not␣
↪appreciate access from
            multiple memory spaces caused huge issues
        * Now we are no longer limited as all threads share the same space
    * Communication between the sequencer and operations is much more stable and less␣
↪limiting
    * stationexec no longer runs on multiple processors, but is a fairly low-overhead␣
↪application anyway
* localonly mode
    * Allows running stations in mode that never connects to external database
    * Stores results locally for ease of initial setup
    * No MySQL database required to develop, test, and run!
* Lots of work to get cameras working
    * Basler and IDS camera drivers included
    * Support for still photos and streaming MJPEG video
```

(continues on next page)

(continued from previous page)

```
    * Easy support for adding video stream from camera to web UI
* Copious usability and stability fixes - here are a few:
    * Lots of cleanup related to tool checking/checkout due to process -> thread␣
↪change (hooray for thread locks!)
    * Fixes to improve multiplatform support for Python 2 and 3 (developing on␣
↪Windows 10, Centos 7, and Ubuntu)
    * Changing tool status checks from threads to coroutines to gain better control␣
↪over the status check execution
        and ward off thread safety issues
    * Disallow operations checking out results and authorization databases so users␣
↪cannot tamper with results
* Widespread documentation and style improvements
```

17.16 What's new in Station Exec 0.4.1

17.16.1 June 27, 2018

```
* Local/Remote database sync
    * Store results, users, and stations in a local file database
    * If connectivity is available for remote MySQL database, sync local results
    * Prevents loss of data due to loss of connectivity
    * Handles collisions and duplicate data
    * Runs periodically to allow for protection even with an intermittent connection
* New Developer Tools
    * tool_util class that eases testing of tools
    * Allows developer to launch tools in a context similar to stationexec without␣
↪the extra bulk
    * Makes debugging and testing tools much easier
* Copious usability and stability fixes - here are a few:
    * Made tool online/offline status easier to manipulate and all related systems␣
↪more robust
    * Using new sphinx module to make HTTP API docs look much better and make them␣
↪easier to read
    * Exception logging made more verbose to give better insight
    * Context manager to make tool checkout/return more intuitive
    * Making sure missing SSL certificates doesn't break software (in case they aren
↪'t wanted)
* Widespread documentation and style improvements
```

17.17 What's new in Station Exec 0.4.0

17.17.1 June 19, 2018

```
New features:
* Improvements to hello-world run
* Sequence DAG implementation
* Sequence DAG visualization
* Main UI updated
```

17.18 What's new in Station Exec 0.3.1

17.18.1 June 5, 2018

```
Fixing bugs associated with doc building (circular imports)
Fixing issues with SQLite on Python 3
Updating user guide documentation
```

17.19 What's new in Station Exec 0.3.0

17.19.1 June 2, 2018

```
New features:
* Built-in hello-world example
* Results storage to database
* Local database storage
* Offline mode
* Working user login
* New Station registration
* Local status logging to file
* Updated asynchronous tool base
* Formatting cleanup
* Usability tweaks
* Early UI Updates
```

17.20 What's new in Station Exec 0.2.0

17.20.1 May 7, 2018

```
New features:
* Config file editing support
* Tool usage logging
* Authenticated action logging
* Camera tool
* Results storage database
```

17.21 What's new in Station Exec 0.1.1

17.21.1 April 26, 2018

```
Fixing bugs associated with print function in python 2 vs 3 and a misnamed sequence␣
↪step status.
```

17.22 What's new in Station Exec 0.1.0

17.22.1 April 20, 2018

```
New features:
* Generated documentation
* Initial database connectivity
* User login capability
* Initial examples
* Sequence status display on UI
```

genindex modindex search

HTTP ROUTING TABLE

/sequence

GET /sequence/status, 18
POST /sequence/start, 18
POST /sequence/stop, 18

/shutdown

POST /shutdown, 18

/tool

GET /tool, 19
GET /tool/status, 20
GET /tool/ui/<tool_id>, 21
POST /tool/command, 21

PYTHON MODULE INDEX

S

stationexec.authentication.auth, 55
stationexec.authentication.authlevel, 55
stationexec.authentication.login, 55
stationexec.authentication.logout, 56
stationexec.authentication.tokenmanager, 56
stationexec.authentication.userauthinfo, 57
stationexec.built_in.exampletool, 61
stationexec.built_in.exampletool2, 61
stationexec.built_in.station_storage, 61
stationexec.cli.cli_tools, 59
stationexec.cli.common, 59
stationexec.cli.for_stations, 60
stationexec.cli.for_tools, 60
stationexec.executive, 23
stationexec.logger.log, 63
stationexec.logger.logger, 64
stationexec.main, 24
stationexec.sequencer.handlers, 33
stationexec.sequencer.loop_condition, 33
stationexec.sequencer.operation, 29
stationexec.sequencer.operationloop, 35
stationexec.sequencer.operationstates, 36
stationexec.sequencer.sequencer, 26
stationexec.station.data_storage, 37
stationexec.station.events, 40
stationexec.station.handlers, 42
stationexec.toolbox.asynctoolbase, 46
stationexec.toolbox.camerabase, 50
stationexec.toolbox.handlers, 51
stationexec.toolbox.tool, 43
stationexec.toolbox.tool_launch, 52
stationexec.toolbox.tool_utilities, 54
stationexec.toolbox.toolbox, 44
stationexec.utilities.byte_conversion, 65
stationexec.utilities.classproperty, 65
stationexec.utilities.colors, 65
stationexec.utilities.config, 66
stationexec.utilities.exceptions, 67
stationexec.utilities.gettool, 68
stationexec.utilities.result_references, 68
stationexec.utilities.shutdown, 69
stationexec.utilities.singleton, 69
stationexec.utilities.time, 69
stationexec.utilities.uuidstr, 70
stationexec.version, 24
stationexec.web.handlers, 71
stationexec.web.web_helpers, 72
stationexec.web.websocket, 72

INDEX

A

ABORTED (*stationexec.sequencer.operationstates.OperationState attribute*), 36
AbortException, 68
ActionEvents (*class in stationexec.station.events*), 41
ADMINISTRATOR (*stationexec.authentication.authlevel.AuthLevel attribute*), 55
ALERT_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
AsyncMessage (*class in stationexec.toolbox.asynctoolbase*), 47
AsyncTool (*class in stationexec.toolbox.asynctoolbase*), 47
AsyncToolBase (*class in stationexec.toolbox.asynctoolbase*), 46
AuthLevel (*class in stationexec.authentication.authlevel*), 55

B

begin_video_stream() (*stationexec.toolbox.camerabase.CameraBase method*), 50
BLINK (*stationexec.utilities.colors.Colors attribute*), 66
BOLD (*stationexec.utilities.colors.Colors attribute*), 66
build_tool_dict() (*stationexec.toolbox.tool_launch.ToolLaunch method*), 52

C

CameraBase (*class in stationexec.toolbox.camerabase*), 50
CameraStreamHandler (*class in stationexec.toolbox.camerabase*), 51
capture_image() (*stationexec.toolbox.camerabase.CameraBase method*), 50
capture_stream_image() (*stationexec.toolbox.camerabase.CameraBase method*), 51
check_if_client_available() (*in module stationexec.web.web_helpers*), 72
check_password() (*stationexec.authentication.userauthinfo.UserAuthInfo static method*), 57
checkout_tool() (*stationexec.sequencer.operation.Operation method*), 32
checkout_tool() (*stationexec.toolbox.toolbox.ToolBox method*), 45
ClassProperty (*class in stationexec.utilities.classproperty*), 65
cleanup() (*stationexec.sequencer.operation.Operation method*), 31
clear_rx_buffer() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 47
clear_rx_queue() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
cli_hello() (*in module stationexec.cli.cli_tools*), 59
cli_setup() (*in module stationexec.cli.cli_tools*), 59
cli_start() (*in module stationexec.cli.cli_tools*), 59
cli_station() (*in module stationexec.cli.cli_tools*), 59
cli_tool() (*in module stationexec.cli.cli_tools*), 59
cli_which() (*in module stationexec.cli.cli_tools*), 59
close_on_error() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
Colors (*class in stationexec.utilities.colors*), 65
compare() (*in module stationexec.sequencer.loop_condition*), 33
comparison (*stationexec.sequencer.loop_condition.LoopCondition attribute*), 35
COMPLETED (*stationexec.sequencer.operationstates.OperationState attribute*), 36
condition (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35
config (*stationexec.executive.Executive attribute*), 23
connection() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
connection_ip() (*sta-*

tionexec.toolbox.asynctoolbase.AsyncTool method), 48
connection_serial() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
convert_value_to_type() (*in module stationexec.sequencer.loop_condition*), 34
create_folder() (*in module stationexec.cli.common*), 60
create_from_template() (*in module stationexec.cli.common*), 60
create_init_files() (*in module stationexec.cli.common*), 60
create_token() (*in module stationexec.authentication.tokenmanager*), 56
create_tool() (*stationexec.toolbox.toolbox.ToolBox method*), 45
current_sequence_id() (*stationexec.sequencer.sequencer.Sequencer method*), 26

D

data_received() (*stationexec.web.handlers.ExecutiveHandler method*), 71
data_received() (*stationexec.web.handlers.ExecutiveStaticFileHandler method*), 72
DataStorage (*class in stationexec.station.data_storage*), 40
DEBUG (*stationexec.logger.log.LogKind attribute*), 63
debug (*stationexec.web.handlers.ExecutiveStaticFileHandler attribute*), 72
debug() (*in module stationexec.logger.log*), 63
delete() (*stationexec.web.websocket.SocketManager method*), 73
destroy_token() (*in module stationexec.authentication.tokenmanager*), 56
DISPLAY (*stationexec.logger.log.LogKind attribute*), 63

E

EMERGENCY_STOP (*stationexec.station.events.ActionEvents attribute*), 41
EMERGENCY_STOP (*stationexec.station.events.InfoEvents attribute*), 41
EMERGENCY_STOP_CLEAR (*stationexec.station.events.ActionEvents attribute*), 41
EMERGENCY_STOP_CLEARED (*stationexec.station.events.InfoEvents attribute*), 41
emit_event() (*in module stationexec.station.events*), 40
emit_event() (*stationexec.toolbox.tool.Tool method*), 44
emit_event_non_blocking() (*in module stationexec.station.events*), 40
encrypt_password() (*stationexec.authentication.userauthinfo.UserAuthInfo static method*), 57
encryption_type() (*stationexec.authentication.userauthinfo.UserAuthInfo static method*), 57
end_video_stream() (*stationexec.toolbox.camerabase.CameraBase method*), 50
ENDC (*stationexec.utilities.colors.Colors attribute*), 66
ERROR (*stationexec.logger.log.LogKind attribute*), 63
ERROR (*stationexec.sequencer.operationstates.OperationState attribute*), 36
ERROR (*stationexec.utilities.colors.Colors attribute*), 65
error() (*in module stationexec.logger.log*), 64
estop() (*stationexec.executive.Executive method*), 24
estop_clear() (*stationexec.executive.Executive method*), 24
eval() (*stationexec.sequencer.loop_condition.LoopCondition method*), 34
eval_log() (*in module stationexec.sequencer.loop_condition*), 34
EXCEPTION (*stationexec.logger.log.LogKind attribute*), 63
exception() (*in module stationexec.logger.log*), 64
Executive (*class in stationexec.executive*), 23
executive (*stationexec.authentication.logout.LogoutHandler attribute*), 56
ExecutiveHandler (*class in stationexec.web.handlers*), 71
ExecutiveStaticFileHandler (*class in stationexec.web.handlers*), 71
ExecutiveTemplateLoader (*class in stationexec.web.handlers*), 72
ExecutiveUI (*class in stationexec.web.handlers*), 71

F

find_tool_by_name() (*in module stationexec.toolbox.tool_utilities*), 54
first_id_in_loop() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
first_idx_in_loop (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35
for_stations() (*in module stationexec.cli.for_stations*), 60
for_tools() (*in module stationexec.cli.for_tools*), 60

format_name() (*in module stationexec.utilities.config*), 66
fps (*stationexec.toolbox.camerabase.CameraStreamHandler attribute*), 51

G

get() (*stationexec.authentication.login.LoginHandler method*), 55
get() (*stationexec.authentication.logout.LogoutHandler method*), 56
get() (*stationexec.sequencer.handlers.SequenceStatusHandler method*), 33
get() (*stationexec.station.handlers.StationHandler method*), 42
get() (*stationexec.station.handlers.StationHandlerSeqRecent method*), 42
get() (*stationexec.station.handlers.StationHandlerSeqSummary method*), 42
get() (*stationexec.station.handlers.StationStatusHandler method*), 42
get() (*stationexec.toolbox.camerabase.CameraStreamHandler method*), 51
get() (*stationexec.toolbox.handlers.ToolboxStatus method*), 51
get() (*stationexec.toolbox.handlers.ToolboxUI method*), 52
get() (*stationexec.toolbox.handlers.ToolboxUIFrame method*), 52
get() (*stationexec.toolbox.tool_launch.MainHandler method*), 53
get() (*stationexec.web.handlers.ExecutiveUI method*), 71
get_absolute_path() (*stationexec.web.handlers.ExecutiveStaticFileHandler class method*), 72
get_access_level_for_token() (*in module stationexec.authentication.tokenmanager*), 56
get_all_operation_status() (*stationexec.sequencer.sequencer.Sequencer method*), 27
get_all_paths() (*in module stationexec.utilities.config*), 66
GET_ALL_USERS (*stationexec.station.events.RetrievalEvents attribute*), 41
get_cfg() (*stationexec.executive.Executive method*), 23
get_current_user() (*stationexec.web.handlers.ExecutiveHandler method*), 71
GET_CUSTOM_DATA (*stationexec.station.events.RetrievalEvents attribute*), 40
get_default_paths() (*in module stationexec.utilities.config*), 66
get_endpoints() (*stationexec.executive.Executive method*), 23
get_endpoints() (*stationexec.toolbox.camerabase.CameraBase method*), 50
get_endpoints() (*stationexec.toolbox.tool.Tool method*), 44
get_endpoints() (*stationexec.toolbox.toolbox.ToolBox method*), 45
get_id() (*stationexec.sequencer.operation.Operation method*), 30
get_latest (*stationexec.toolbox.camerabase.CameraStreamHandler attribute*), 51
get_latest() (*stationexec.toolbox.camerabase.CameraBase method*), 50
get_latest_raw() (*stationexec.toolbox.camerabase.CameraBase method*), 50
get_local_time() (*in module stationexec.utilities.time*), 69
GET_LOG_DATA (*stationexec.station.events.RetrievalEvents attribute*), 41
get_loop_iteration() (*stationexec.sequencer.operation.Operation method*), 32
GET_MAINTENANCE_DATA (*stationexec.station.events.RetrievalEvents attribute*), 41
get_name() (*stationexec.sequencer.operation.Operation method*), 30
get_operand() (*stationexec.sequencer.loop_condition.LoopConditionOperand method*), 34
get_operand() (*stationexec.sequencer.loop_condition.LoopRepeatOperand method*), 34
GET_OPERATION_AVERAGE_DURATION (*stationexec.station.events.RetrievalEvents attribute*), 41
get_operation_result() (*stationexec.sequencer.sequencer.Sequencer method*), 27
get_range() (*stationexec.sequencer.loop_condition.LoopConditionOper method*), 34
get_range() (*stationexec.sequencer.loop_condition.LoopRepeatOperan static method*), 34
get_results() (*stationexec.sequencer.operation.Operation method*), 31
get_runtime_data() (*stationexec.sequencer.operation.Operation*

method), 32
GET_STATION_DATA (*stationexec.station.events.RetrievalEvents attribute*), 41
get_status() (*stationexec.sequencer.operation.Operation method*), 31
get_status() (*stationexec.toolbox.tool.Tool method*), 44
get_status() (*stationexec.toolbox.toolbox.ToolBox method*), 45
get_system_config() (*in module stationexec.utilities.config*), 66
get_token() (*in module stationexec.authentication.tokenmanager*), 56
get_tool_list() (*stationexec.toolbox.toolbox.ToolBox method*), 45
get_tool_status() (*stationexec.toolbox.toolbox.ToolBox method*), 45
get_tools() (*in module stationexec.toolbox.toolbox*), 44
get_tools_for_ui() (*stationexec.toolbox.toolbox.ToolBox method*), 45
GET_USER_DATA (*stationexec.station.events.RetrievalEvents attribute*), 41
GET_USER_VALID_LOGIN (*stationexec.station.events.RetrievalEvents attribute*), 41
get_userfullname_for_token() (*in module stationexec.authentication.tokenmanager*), 57
get_userid_for_token() (*in module stationexec.authentication.tokenmanager*), 56
get_username_for_token() (*in module stationexec.authentication.tokenmanager*), 57
get_utc_now() (*in module stationexec.utilities.time*), 69
get_uuid() (*in module stationexec.utilities.uuidstr*), 70
get_web_info() (*stationexec.executive.Executive method*), 23
GetTool (*class in stationexec.utilities.gettool*), 68
GetToolRetry (*class in stationexec.utilities.gettool*), 68

H

HEADER (*stationexec.utilities.colors.Colors attribute*), 65
http10_get() (*in module stationexec.web.web_helpers*), 72
http10_post() (*in module stationexec.web.web_helpers*), 72
http_get() (*in module stationexec.web.web_helpers*), 72
http_post() (*in module stationexec.web.web_helpers*), 72

I

IDLE (*stationexec.sequencer.operationstates.OperationState attribute*), 36
increment() (*stationexec.sequencer.loop_condition.LoopRepeatOperan method*), 34
INFO (*stationexec.logger.log.LogKind attribute*), 63
INFO (*stationexec.utilities.colors.Colors attribute*), 65
info() (*in module stationexec.logger.log*), 63
InfoEvents (*class in stationexec.station.events*), 41
init() (*stationexec.logger.logger.Logger method*), 64
init() (*stationexec.utilities.singleton.Singleton method*), 69
initialize() (*stationexec.authentication.login.LoginHandler method*), 55
initialize() (*stationexec.authentication.logout.LogoutHandler method*), 56
initialize() (*stationexec.executive.Executive method*), 23
initialize() (*stationexec.sequencer.handlers.SequenceStartHandler method*), 33
initialize() (*stationexec.sequencer.handlers.SequenceStatusHandler method*), 33
initialize() (*stationexec.sequencer.handlers.SequenceStopHandler method*), 33
initialize() (*stationexec.sequencer.operation.Operation method*), 30
initialize() (*stationexec.sequencer.sequencer.Sequencer method*), 26
initialize() (*stationexec.station.data_storage.DataStorage method*), 40
initialize() (*stationexec.station.handlers.StationCommandHandler method*), 42
initialize() (*stationexec.station.handlers.StationHandler method*), 42
initialize() (*stationexec.station.handlers.StationHandlerSeqRecent method*), 42
initialize() (*stationexec.station.handlers.StationHandlerSeqSummar method*), 42
initialize() (*stationexec.station.handlers.StationStatusHandler method*), 42
initialize() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 47
initialize() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
initialize() (*stationexec.toolbox.camerabase.CameraBase method*), 51
initialize() (*stationexec.toolbox.camerabase.CameraStreamHandler method*), 51

initialize() (*stationexec.toolbox.handlers.ToolboxStatus method*), 51
initialize() (*stationexec.toolbox.handlers.ToolboxUI method*), 52
initialize() (*stationexec.toolbox.handlers.ToolboxUIFrame method*), 52
initialize() (*stationexec.toolbox.tool.Tool method*), 43
initialize() (*stationexec.toolbox.tool_launch.MainHandler method*), 53
initialize() (*stationexec.toolbox.tool_launch.ShutdownHandler method*), 54
initialize() (*stationexec.toolbox.tool_launch.ToolCommand method*), 54
initialize() (*stationexec.toolbox.toolbox.ToolBox method*), 44
initialize() (*stationexec.web.handlers.ExecutiveStaticFileHandler method*), 72
initialize() (*stationexec.web.handlers.ExecutiveUI method*), 71
initialize() (*stationexec.web.websocket.SocketManager method*), 73
initialize() (*stationexec.web.websocket.StationSocket method*), 72
initialize_tools() (*stationexec.toolbox.toolbox.ToolBox method*), 45
install_signal_handlers() (*in module stationexec.utilities.shutdown*), 69
interactive_tool() (*in module stationexec.cli.for_tools*), 60
interactive_tool_ui() (*in module stationexec.cli.for_tools*), 60
internal_tool_path (*stationexec.web.handlers.ExecutiveStaticFileHandler attribute*), 72
InvalidAttribute, 67
is_available() (*stationexec.sequencer.sequencer.Sequencer method*), 26
is_float_string() (*in module stationexec.sequencer.loop_condition*), 33
is_hex_string() (*in module stationexec.sequencer.loop_condition*), 34
is_int_string() (*in module stationexec.sequencer.loop_condition*), 34
is_postcheck() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
is_precheck() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
is_range() (*stationexec.sequencer.loop_condition.LoopConditionOperand method*), 34
is_range() (*stationexec.sequencer.loop_condition.LoopRepeatOperand static method*), 34
is_repeat() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
is_running() (*stationexec.sequencer.sequencer.Sequencer method*), 26
is_shutdown() (*stationexec.sequencer.operation.Operation method*), 31
isnumeric() (*stationexec.sequencer.operation.Operation static method*), 31

J

json_args (*stationexec.web.handlers.ExecutiveHandler attribute*), 71

L

last_id_in_loop() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
last_idx_in_loop (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35
launch_sequence (*stationexec.sequencer.handlers.SequenceStartHandler attribute*), 33
launch_sequence() (*stationexec.executive.Executive method*), 24
list_all_available_tools() (*in module stationexec.toolbox.tool_utilities*), 54
listen_for_event() (*stationexec.toolbox.tool.Tool method*), 44
load_config() (*in module stationexec.utilities.config*), 66
load_tool_modules() (*stationexec.toolbox.toolbox.ToolBox method*), 44
load_tool_object() (*in module stationexec.toolbox.tool_utilities*), 54
local_to_utc() (*in module stationexec.utilities.time*), 69
LOG (*stationexec.station.events.InfoEvents attribute*), 41
log_message() (*stationexec.logger.logger.Logger method*), 64
Logger (*class in stationexec.logger.logger*), 64
LoginHandler (*class in stationexec.authentication.login*), 55
LogKind (*class in stationexec.logger.log*), 63
LogoutHandler (*class in stationexec.authentication.logout*), 56
looks_like_result_reference() (*in module stationexec.utilities.result_references*), 68
loop_id (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35

loop_type (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35
LoopCondition (*class in stationexec.sequencer.loop_condition*), 34
LoopConditionOperand (*class in stationexec.sequencer.loop_condition*), 34
LoopRepeatOperand (*class in stationexec.sequencer.loop_condition*), 34

M

Main (*class in stationexec.main*), 24
MainHandler (*class in stationexec.toolbox.tool_launch*), 53
manager (*stationexec.web.websocket.StationSocket attribute*), 72
member_id_of_idx() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
member_idx_of_id() (*stationexec.sequencer.operationloop.OperationLoop method*), 35
member_index_list (*stationexec.sequencer.operationloop.OperationLoop attribute*), 35
merge_config_data() (*in module stationexec.utilities.config*), 66
MESSAGE_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
MissingResult, 67
module_path (*stationexec.web.handlers.ExecutiveStaticFileHandler attribute*), 71

N

no_image_yet (*stationexec.toolbox.camerabase.CameraStreamHandler attribute*), 51
NOBLINK (*stationexec.utilities.colors.Colors attribute*), 66
NOBOLD (*stationexec.utilities.colors.Colors attribute*), 66
NONE (*stationexec.authentication.authlevel.AuthLevel attribute*), 55
NOUNDERLINE (*stationexec.utilities.colors.Colors attribute*), 66

O

OBJECT_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
OKBLUE (*stationexec.utilities.colors.Colors attribute*), 65
OKGREEN (*stationexec.utilities.colors.Colors attribute*), 65
on_close() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
on_close() (*stationexec.web.websocket.StationSocket method*), 72
on_connect() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
ON_CREATE_USER (*stationexec.station.events.StorageEvents attribute*), 40
ON_CUSTOM_STORAGE (*stationexec.station.events.StorageEvents attribute*), 40
ON_DATA_STORE (*stationexec.station.events.StorageEvents attribute*), 40
on_finish() (*stationexec.web.handlers.ExecutiveHandler method*), 71
ON_LOG_DATA (*stationexec.station.events.StorageEvents attribute*), 40
ON_MAINTENANCE_EVENT (*stationexec.station.events.StorageEvents attribute*), 40
on_message() (*stationexec.web.websocket.StationSocket method*), 72
ON_OPERATION_END (*stationexec.station.events.StorageEvents attribute*), 40
ON_OPERATION_START (*stationexec.station.events.StorageEvents attribute*), 40
ON_REGISTER_STATION (*stationexec.station.events.StorageEvents attribute*), 40
ON_RESULT_STORE (*stationexec.station.events.StorageEvents attribute*), 40
ON_SEQUENCE_END (*stationexec.station.events.StorageEvents attribute*), 40
ON_SEQUENCE_START (*stationexec.station.events.StorageEvents attribute*), 40
on_ui_command() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 47
on_ui_command() (*stationexec.toolbox.camerabase.CameraBase method*), 51
on_ui_command() (*stationexec.toolbox.tool.Tool method*), 44
ON_UPDATE_STATION (*stationexec.station.events.StorageEvents attribute*), 40
ON_UPDATE_USER (*stationexec.station.events.StorageEvents attribute*), 40

open() (*stationexec.web.websocket.StationSocket method*), 72
operand (*stationexec.sequencer.loop_condition.LoopCondition attribute*), 35
operand_is_result() (*stationexec.sequencer.loop_condition.LoopCondition method*), 35
Operation (*class in stationexec.sequencer.operation*), 29
operation_action() (*stationexec.sequencer.operation.Operation method*), 30
operation_has_result() (*in module stationexec.utilities.result_references*), 69
OperationLoop (*class in stationexec.sequencer.operationloop*), 35
OperationState (*class in stationexec.sequencer.operationstates*), 36
OPERATOR (*stationexec.authentication.authlevel.AuthLevel attribute*), 55

P

parse_rx_data() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
parse_url_path() (*stationexec.web.handlers.ExecutiveStaticFileHandler method*), 72
POPUP_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
post() (*stationexec.authentication.login.LoginHandler method*), 55
post() (*stationexec.sequencer.handlers.SequenceStartHandler method*), 33
post() (*stationexec.sequencer.handlers.SequenceStopHandler method*), 33
post() (*stationexec.station.handlers.StationCommandHandler method*), 42
post() (*stationexec.toolbox.handlers.ToolCommand method*), 51
post() (*stationexec.toolbox.tool_launch.ShutdownHandler method*), 54
post() (*stationexec.toolbox.tool_launch.ToolCommand method*), 54
post() (*stationexec.web.handlers.ShutdownHandler method*), 71
prepare() (*stationexec.sequencer.operation.Operation method*), 30
prepare() (*stationexec.web.handlers.ExecutiveHandler method*), 71
process_image() (*stationexec.toolbox.camerabase.CameraBase method*), 51
process_name() (*in module stationexec.utilities.shutdown*), 69
prompt_for_bool() (*in module stationexec.cli.common*), 60
prompt_for_list() (*in module stationexec.cli.common*), 59
prompt_for_text() (*in module stationexec.cli.common*), 59

R

read() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
read_ip() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
read_serial() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
receive() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
reference_parse_parts() (*in module stationexec.utilities.result_references*), 68
register_event() (*stationexec.station.data_storage.DataStorage method*), 40
register_for_event() (*in module stationexec.station.events*), 40
register_for_event_group() (*in module stationexec.station.events*), 40
register_for_events() (*in module stationexec.station.events*), 40
reload() (*stationexec.sequencer.sequencer.Sequencer method*), 26
RELOAD_SEQUENCE_OPERATIONS (*stationexec.station.events.ActionEvents attribute*), 41
RELOAD_STATION (*stationexec.station.events.ActionEvents attribute*), 41
RELOAD_TOOL_MANIFEST (*stationexec.station.events.ActionEvents attribute*), 41
remote_path_import() (*in module stationexec.utilities.config*), 67
REQUEUE (*stationexec.sequencer.operationstates.OperationState attribute*), 36
reset() (*stationexec.sequencer.loop_condition.LoopCondition method*), 35
reset() (*stationexec.sequencer.loop_condition.LoopRepeatOperand method*), 34
resolve_path() (*stationexec.web.handlers.ExecutiveTemplateLoader method*), 72
restrict_access() (*in module stationexec.authentication.auth*), 55

RESULTFAILURE (*stationexec.sequencer.operationstates.OperationState attribute*), 36
RetrievalEvents (*class in stationexec.station.events*), 40
return_tool() (*stationexec.sequencer.operation.Operation method*), 32
return_tool() (*stationexec.toolbox.toolbox.ToolBox method*), 46
ROOT (*stationexec.authentication.authlevel.AuthLevel attribute*), 55
RootDoesNotExist, 66
run() (*stationexec.sequencer.operation.Operation method*), 30
run() (*stationexec.sequencer.sequencer.Sequencer method*), 26
run() (*stationexec.toolbox.camerabase.CameraBase method*), 50
run_on_connect() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
run_on_connect() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 47
RUNNING (*stationexec.sequencer.operationstates.OperationState attribute*), 36

S

save() (*stationexec.web.websocket.SocketManager method*), 73
save_image() (*stationexec.toolbox.camerabase.CameraBase method*), 51
save_result() (*stationexec.sequencer.operation.Operation method*), 31
send() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
send_all() (*stationexec.web.websocket.SocketManager method*), 73
send_receive() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
SEQUENCE_ABORTED (*stationexec.station.events.InfoEvents attribute*), 41
sequence_cleanup() (*stationexec.executive.Executive method*), 24
SEQUENCE_FAILED (*stationexec.station.events.InfoEvents attribute*), 41
SEQUENCE_FINISHED (*stationexec.station.events.InfoEvents attribute*), 41
SEQUENCE_LOADED (*stationexec.station.events.InfoEvents attribute*), 42
sequence_setup() (*stationexec.executive.Executive method*), 24
SEQUENCE_STARTED (*stationexec.station.events.InfoEvents attribute*), 41
sequence_status (*stationexec.sequencer.handlers.SequenceStatusHandler attribute*), 33
sequence_status() (*stationexec.executive.Executive method*), 24
sequence_success_determinator() (*stationexec.sequencer.sequencer.Sequencer method*), 27
SEQUENCE_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
Sequencer (*class in stationexec.sequencer.sequencer*), 26
SequenceStartHandler (*class in stationexec.sequencer.handlers*), 33
SequenceStatusHandler (*class in stationexec.sequencer.handlers*), 33
SequenceStopHandler (*class in stationexec.sequencer.handlers*), 33
SERVER_STARTED (*stationexec.station.events.InfoEvents attribute*), 42
set_exposure() (*stationexec.toolbox.camerabase.CameraBase method*), 50
set_extra_headers() (*stationexec.web.handlers.ExecutiveStaticFileHandler method*), 72
set_offline() (*stationexec.toolbox.tool.Tool method*), 44
set_online() (*stationexec.toolbox.tool.Tool method*), 44
set_processor_callback() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 48
set_runtime_data() (*stationexec.sequencer.operation.Operation method*), 32
set_station_identity() (*in module stationexec.utilities.config*), 66
set_status() (*stationexec.sequencer.operation.Operation method*), 31
set_status() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
set_status() (*stationexec.toolbox.tool.Tool method*), 44

set_status_callback() (stationexec.toolbox.asynctoolbase.AsyncTool method), 48
set_stream_color_mode() (stationexec.toolbox.camerabase.CameraBase method), 51
set_tool_management() (stationexec.station.data_storage.DataStorage method), 40
setup_tool_server() (stationexec.toolbox.tool_launch.ToolLaunch method), 53
SHUTDOWN (stationexec.station.events.ActionEvents attribute), 41
shutdown() (stationexec.executive.Executive method), 23
shutdown() (stationexec.main.Main method), 24
shutdown() (stationexec.sequencer.operation.Operation method), 31
shutdown() (stationexec.sequencer.sequencer.Sequencer method), 26
shutdown() (stationexec.station.data_storage.DataStorage method), 40
shutdown() (stationexec.toolbox.asynctoolbase.AsyncTool method), 47
shutdown() (stationexec.toolbox.asynctoolbase.AsyncToolBase method), 46
shutdown() (stationexec.toolbox.camerabase.CameraBase method), 51
shutdown() (stationexec.toolbox.tool.Tool method), 43
shutdown() (stationexec.toolbox.tool_launch.ToolLaunch method), 53
shutdown() (stationexec.toolbox.toolbox.ToolBox method), 45
shutdown_requested (stationexec.sequencer.sequencer.Sequencer attribute), 27
ShutdownHandler (class in stationexec.toolbox.tool_launch), 54
ShutdownHandler (class in stationexec.web.handlers), 71
SHUTTING_DOWN (stationexec.station.events.InfoEvents attribute), 41
sig_handler() (in module stationexec.utilities.shutdown), 69
signal_list() (in module stationexec.utilities.shutdown), 69
Singleton (class in stationexec.utilities.singleton), 69
SocketManager (class in stationexec.web.websocket), 73
start() (stationexec.main.Main method), 24
start_capture() (stationexec.toolbox.camerabase.CameraBase method), 50
START_SEQUENCE (stationexec.station.events.ActionEvents attribute), 41
STATION_COMMAND (stationexec.station.events.InfoEvents attribute), 42
station_info (stationexec.authentication.login.LoginHandler attribute), 55
station_info (stationexec.executive.Executive attribute), 23
STATION_LOADED (stationexec.station.events.InfoEvents attribute), 42
station_name (stationexec.logger.logger.Logger attribute), 64
station_path (stationexec.web.handlers.ExecutiveStaticFileHandler attribute), 72
station_uuid (stationexec.executive.Executive attribute), 24
StationCommandHandler (class in stationexec.station.handlers), 42
StationError, 67
stationexec.authentication.auth (module), 55
stationexec.authentication.authlevel (module), 55
stationexec.authentication.login (module), 55
stationexec.authentication.logout (module), 56
stationexec.authentication.tokenmanager (module), 56
stationexec.authentication.userauthinfo (module), 57
stationexec.built_in.exampletool (module), 61
stationexec.built_in.exampletool2 (module), 61
stationexec.built_in.station_storage (module), 61
stationexec.cli.cli_tools (module), 59
stationexec.cli.common (module), 59
stationexec.cli.for_stations (module), 60
stationexec.cli.for_tools (module), 60
stationexec.executive (module), 23
stationexec.logger.log (module), 63
stationexec.logger.logger (module), 64
stationexec.main (module), 24
stationexec.sequencer.handlers (module), 33
stationexec.sequencer.loop_condition (module), 33

stationexec.sequencer.operation (*module*), 29
stationexec.sequencer.operationloop (*module*), 35
stationexec.sequencer.operationstates (*module*), 36
stationexec.sequencer.sequencer (*module*), 26
stationexec.station.data_storage (*module*), 37
stationexec.station.events (*module*), 40
stationexec.station.handlers (*module*), 42
stationexec.toolbox.asynctoolbase (*module*), 46
stationexec.toolbox.camerabase (*module*), 50
stationexec.toolbox.handlers (*module*), 51
stationexec.toolbox.tool (*module*), 43
stationexec.toolbox.tool_launch (*module*), 52
stationexec.toolbox.tool_utilities (*module*), 54
stationexec.toolbox.toolbox (*module*), 44
stationexec.utilities.byte_conversion (*module*), 65
stationexec.utilities.classproperty (*module*), 65
stationexec.utilities.colors (*module*), 65
stationexec.utilities.config (*module*), 66
stationexec.utilities.exceptions (*module*), 67
stationexec.utilities.gettool (*module*), 68
stationexec.utilities.result_references (*module*), 68
stationexec.utilities.shutdown (*module*), 69
stationexec.utilities.singleton (*module*), 69
stationexec.utilities.time (*module*), 69
stationexec.utilities.uuidstr (*module*), 70
stationexec.version (*module*), 24
stationexec.web.handlers (*module*), 71
stationexec.web.web_helpers (*module*), 72
stationexec.web.websocket (*module*), 72
StationHandler (*class in stationexec.station.handlers*), 42
StationHandlerSeqRecent (*class in stationexec.station.handlers*), 42
StationHandlerSeqSummary (*class in stationexec.station.handlers*), 42
StationSocket (*class in stationexec.web.websocket*), 72
StationStatusHandler (*class in stationexec.station.handlers*), 42
stop() (*stationexec.sequencer.sequencer.Sequencer method*), 26
stop_capture() (*stationexec.toolbox.camerabase.CameraBase method*), 50
stop_loop() (*in module stationexec.utilities.shutdown*), 69
stop_requested (*stationexec.sequencer.sequencer.Sequencer attribute*), 27
STOP_SEQUENCE (*stationexec.station.events.ActionEvents attribute*), 41
StorageEvents (*class in stationexec.station.events*), 40
StorageShutdown, 40
str2uuid() (*in module stationexec.utilities.uuidstr*), 70

T

TECHNICIAN (*stationexec.authentication.authlevel.AuthLevel attribute*), 55
terminate_sequence (*stationexec.sequencer.handlers.SequenceStopHandler attribute*), 33
terminate_sequence() (*stationexec.executive.Executive method*), 24
to_bytes() (*in module stationexec.utilities.byte_conversion*), 65
to_datetime() (*in module stationexec.utilities.time*), 69
to_timestamp() (*in module stationexec.utilities.time*), 69
Tool (*class in stationexec.toolbox.tool*), 43
TOOL_COMMAND (*stationexec.station.events.InfoEvents attribute*), 42
tool_command() (*stationexec.toolbox.tool_launch.ToolLaunch method*), 53
tool_command_build() (*stationexec.toolbox.tool_launch.ToolLaunch static method*), 52
tool_exists() (*stationexec.toolbox.toolbox.ToolBox method*), 45
tool_path (*stationexec.web.handlers.ExecutiveStaticFileHandler attribute*), 72
tool_run() (*stationexec.toolbox.tool_launch.ToolLaunch method*), 53
tool_setup() (*stationexec.toolbox.tool_launch.ToolLaunch method*), 52
tool_status_listener() (*stationexec.toolbox.tool_launch.ToolLaunch method*), 53

tool_status_listener() (*stationexec.toolbox.toolbox.ToolBox method*), 45
TOOL_UPDATE (*stationexec.station.events.InfoEvents attribute*), 41
ToolBox (*class in stationexec.toolbox.toolbox*), 44
ToolboxStatus (*class in stationexec.toolbox.handlers*), 51
ToolboxUI (*class in stationexec.toolbox.handlers*), 52
ToolboxUIFrame (*class in stationexec.toolbox.handlers*), 52
ToolCommand (*class in stationexec.toolbox.handlers*), 51
ToolCommand (*class in stationexec.toolbox.tool_launch*), 54
ToolExistsException, 67
ToolInUseException, 68
ToolLaunch (*class in stationexec.toolbox.tool_launch*), 52
ToolLockException, 67
ToolNotExistsException, 67
ToolNotFound, 54
TOOLS_LOADED (*stationexec.station.events.InfoEvents attribute*), 42
ToolUnavailableException, 68
trigger_event() (*stationexec.station.data_storage.DataStorage method*), 40

U

ui_command() (*stationexec.executive.Executive method*), 23
ui_log() (*stationexec.sequencer.operation.Operation method*), 32
ui_log() (*stationexec.toolbox.tool.Tool method*), 44
UNAUTHORIZED_ACCESS (*stationexec.station.events.InfoEvents attribute*), 41
UNDERLINE (*stationexec.utilities.colors.Colors attribute*), 66
USAGE (*stationexec.logger.log.LogKind attribute*), 63
USER_LOGGED_IN (*stationexec.station.events.InfoEvents attribute*), 41
USER_LOGGED_OUT (*stationexec.station.events.InfoEvents attribute*), 41
UserAuthInfo (*class in stationexec.authentication.userauthinfo*), 57
util (*stationexec.toolbox.tool_launch.MainHandler attribute*), 53
util (*stationexec.toolbox.tool_launch.ShutdownHandler attribute*), 54
util (*stationexec.toolbox.tool_launch.ToolCommand attribute*), 54
uuid2str() (*in module stationexec.utilities.uuidstr*), 70

V

valid_token() (*in module stationexec.authentication.tokenmanager*), 56
validate_absolute_path() (*stationexec.web.handlers.ExecutiveStaticFileHandler method*), 72
value (*stationexec.sequencer.loop_condition.LoopCondition attribute*), 35
value_is_result() (*stationexec.sequencer.loop_condition.LoopCondition method*), 34
value_to_ui() (*stationexec.toolbox.tool.Tool method*), 44
verify_paths_exist() (*in module stationexec.utilities.config*), 66
verify_status() (*stationexec.toolbox.asynctoolbase.AsyncToolBase method*), 46
verify_status() (*stationexec.toolbox.camerabase.CameraBase method*), 51
verify_status() (*stationexec.toolbox.tool.Tool method*), 43
verify_status_loop() (*stationexec.toolbox.toolbox.ToolBox method*), 46

W

WAITING_ON_TOOL (*stationexec.sequencer.operationstates.OperationState attribute*), 36
WARNING (*stationexec.logger.log.LogKind attribute*), 63
WARNING (*stationexec.utilities.colors.Colors attribute*), 65
warning() (*in module stationexec.logger.log*), 63
web_info (*stationexec.web.handlers.ExecutiveUI attribute*), 71
WEBSOCKET_INCOMING (*stationexec.station.events.InfoEvents attribute*), 41
write() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49
write_async() (*stationexec.toolbox.asynctoolbase.AsyncTool method*), 49