microdrop Documentation

Release 2.0.post90.dev66937752

Christian Fobel

September 12, 2016

Contents

Contents:

CHAPTER 1

Project Modules

1.1 microdrop Package

1.1.1 microdrop Package

microdrop.__init__.**base_path**()

microdrop.__init__.**glade_path**()
Return path to *.glade* files used by *gtk* to construct views.

1.1.2 __main__ Module

1.1.3 app Module

1.1.4 app_context Module

Copyright 2011 Ryan Fobel

This file is part of Microdrop.

Microdrop is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by Foundation, either version 3 of the License, or (at your option) any later version.

Microdrop is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

microdrop.app_context.**get_app**()

microdrop.app_context.**get_hub_uri**()

1.1.5 config Module

Copyright 2011 Ryan Fobel

This file is part of Microdrop.

Microdrop is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Microdrop is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

class microdrop.config.**Config**(*filename=None*)
Bases: object

Methods

default_config_directory = path('/home/docs/.microdrop')

default_config_path = path('/home/docs/.microdrop/microdrop.ini')

load(*filename=None*)
Load a Config object from a file.

Parameters **filename** – path to file. If None, try loading from the default location, and if there's no file, create a Config object with the default options.

Raises

- IOError – The file does not exist.
- ConfigObjError – There was a problem parsing the config file.
- *ValidationError* – There was a problem validating one or more fields.

save(*filename=None*)

spec = '\n [dmf_device]\n # name of the most recently used DMF device\n name = string(default=None)\n\n [protocol]\n #

exception microdrop.config.**ValidationError**
Bases: exceptions.Exception

1.1.6 dmf_device Module

Copyright 2011-2015 Ryan Fobel and Christian Fobel

This file is part of Microdrop.

Microdrop is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Microdrop is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

exception microdrop.dmf_device.**DeviceScaleNotSet**
Bases: exceptions.Exception

class microdrop.dmf_device.**DmfDevice**(*svg_filepath*, *name=None*, ***kwargs*)
Bases: object

Attributes

Methods

actuated_area(*state_of_all_channels*)
Compute area of all actuated electrodes.

Parameters **state_of_all_channels** (*np.array*) – An array-like instance containing an actuation level for each respective channel.

Returns Area of actuated electrodes in square millimeters.

Return type float

actuated_channels(*actuated_electrodes_index*)

Parameters **actuated_electrodes_index** (*list or array-like*) – Actuated electrode identifiers.

Returns Actuated channel index values, indexed by electrode identifier.

Return type pandas.Series

actuated_electrodes(*actuated_channels_index*)

Parameters **actuated_channels_index** (*list or array-like*) – Actuated channel indexes.

Returns Actuated electrode identifiers, indexed by channel index.

Return type pandas.Series

df_electrode_channels

diff_electrode_channels()
Identify electrodes with modified channel lists.

Returns Frame containing modified electrode channel lists. The two columns contain a list for the original and new assigned channels, respectively, indexed by `electrode_id`.

Return type pandas.DataFrame

dirty

electrodes

find_path(*source_id*, *target_id*)

Returns A list of nodes on the shortest path from source to target.

Return type list

get_actuated_electrodes_area(*electrode_states*)
Compute area of actuated electrodes.

Parameters **electrode_states** (*pandas.Series*) – Electrode states, indexed by electrode identifier. Any state greater than zero is considered actuated.

Returns Area of actuated electrodes in square millimeters.

Return type float

get_bounding_box()

Returns Tuple containing origin-*x*, origin-*y*, width and height, respectively.

Return type tuple

get_electrode_areas()

Returns Area of each electrode in square millimeters, indexed by electrode identifier.

Return type pandas.Series

get_electrode_channels()
Load the channels associated with each electrode from the device layer of an SVG source.

For each electrode polygon, the channels are read as a comma-separated list from the *"data-channels"* attribute.

Returns Each row corresponds to a channel connected to an electrode, where the `"electrode_id"` column corresponds to the `"id"` attribute of the corresponding SVG polygon.

Return type pandas.DataFrame

Notes

- Each electrode corresponds to a closed path in the device drawing.
- Each channel index corresponds to a DMF device channel that may be actuated independently.

get_svg_frame()
Return a `pandas.DataFrame` containing the vertices for electrode paths.

Each row of the frame corresponds to a single path vertex. The `groupby()` method may be used, for example, to apply operations to vertices on a per-path basis, such as calculating the bounding box.

classmethod load(*svg_filepath*, ***kwargs*)
Load a DmfDevice from a file.

Parameters **filename** – path to file.

Raises

- `TypeError` – file is not a DmfDevice.
- `FutureVersionError` – file was written by a future version of the software.

max_channel()

Returns Maximum channel index.

Return type int

set_electrode_channels(*electrode_id*, *channels*)
Set channels for electrode *electrode_id* to *channels*.

This includes updating *self.df_electrode_channels*.

Note: Existing channels assigned to electrode are overwritten.

Parameters

- **electrode_id** (*str*) – Electrode identifier.
- **channels** (*list*) – List of channel identifiers assigned to the electrode.

Returns `True` if channel mappings have changed.

Return type bool

to_svg()

Returns SVG XML source with up-to-date electrode channel lists.

Return type unicode

microdrop.dmf_device.**extract_channels**(*df_shapes*)

Load the channels associated with each electrode from the device layer of an SVG source.

For each electrode polygon, the channels are read as a comma-separated list from the *"data-channels"* attribute.

Parameters

- **svg_source** (*filepath*) – Input SVG file containing connection lines.
- **shapes_canvas** (*shapes_canvas.ShapesCanvas*) – Shapes canvas containing shapes to compare against connection endpoints.
- **electrode_layer** (*str*) – Name of layer in SVG containing electrodes.
- **electrode_xpath** (*str*) – XPath string to iterate throught electrodes.
- **namespaces** (*dict*) – SVG namespaces (compatible with *etree.parse*).

Returns Each row corresponds to a channel connected to an electrode, where the `"electrode_id"` column corresponds to the `"id"` attribute of the corresponding SVG polygon.

Return type pandas.DataFrame

1.1.7 experiment_log Module

Copyright 2011 Ryan Fobel

This file is part of Microdrop.

Microdrop is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Microdrop is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

class microdrop.experiment_log.**ExperimentLog**(*directory=None*)

Methods

add_data(*data*, *plugin_name='core'*)

add_step(*step_number*, *attempt=0*)

class_version = '0.3.0'

get(*name*, *plugin_name='core'*)

get_log_path()

classmethod load(*filename*)
Load an experiment log from a file.

Parameters **filename** – path to file.

Raises

- TypeError – file is not an experiment log.
- FutureVersionError – file was written by a future version of the software.

save(*filename=None*, *format='pickle'*)

start_time()

1.1.8 interfaces Module

class microdrop.interfaces.**IFoo**
Bases: pyutilib.component.core.core.Interface

class microdrop.interfaces.**ILoggingPlugin**
Bases: pyutilib.component.core.core.Interface

Methods

on_critical(*record*)

on_debug(*record*)

on_error(*record*)

on_info(*record*)

on_warning(*record*)

class microdrop.interfaces.**IPlugin**
Bases: pyutilib.component.core.core.Interface

Methods

get_schedule_requests(*function_name*)

Parameters **function_name** (*str*) – Plugin callback function name.

Returns List of scheduling requests (i.e., ScheduleRequest instances) for the function specified by function_name.

Return type list

get_step_form_class()

get_step_values(*step_number=None*)

on_app_exit()
Handler called just before the Microdrop application exits.

on_app_options_changed(*plugin_name*)
Handler called when the app options are changed for a particular plugin. This will, for example, allow for GUI elements to be updated.

Parameters **plugin** (*str*) – Plugin name for which the app options changed

on_dmf_device_changed(*dmf_device*)
Handler called when a DMF device is modified (e.g., channel assignment, scaling, etc.).

Parameters **dmf_device** (microdrop.dmf_device.DmfDevice) –

on_dmf_device_saved(*dmf_device*)
Handler called when a DMF device is saved.

Parameters **dmf_device** (microdrop.dmf_device.DmfDevice) –

on_dmf_device_swapped(*old_dmf_device*, *dmf_device*)
Handler called when a different DMF device is swapped in (e.g., when a new device is loaded).

Parameters

- **old_dmf_device** (microdrop.dmf_device.DmfDevice) – Original device.
- **dmf_device** (microdrop.dmf_device.DmfDevice) – New device.

on_experiment_log_changed(*experiment_log*)
Handler called when the current experiment log changes (e.g., when a protocol finishes running.

Parameters **experiment_log** (microdrop.experiment_log.ExperimentLog) – Reference to new experiment log instance.

on_experiment_log_selection_changed(*data*)
Handler called whenever the experiment log selection changes.

Parameters **data** – experiment log data (list of dictionaries, one per step) for the selected steps

on_export_experiment_log_data(*experiment_log*)
Handler called when the experiment log is exported.

Parameters **log** – experiment log data (list of dictionaries, one per step) for the selected steps

Returns A dictionary of pandas.DataFrame objects containing any relevant data that should be exported by the plugin, each keyed by a unique name.

on_metadata_changed(*schema*, *original_metadata*, *metadata*)
Handler called each time the experiment metadata has changed.

Parameters

- **schema** (*dict*) – jsonschema schema definition for metadata.
- **original_metadata** – Original metadata.
- **metadata** – New metadata matching `schema`

on_plugin_disable()
Handler called once the plugin instance is disabled.

on_plugin_disabled(*env*, *plugin*)
Handler called to notify that a plugin has been disabled.

Note that this signal is broadcast to all plugins implementing the *IPlugin* interface, whereas the *on_plugin_disable()* method is called directly on the plugin that is being disabled.

Parameters

- **env** (*str*) – `pyutilib` plugin environment.
- **plugin** (*str*) – Plugin name.

on_plugin_enable()
Handler called once the plugin instance is enabled.

Note: if you inherit your plugin from AppDataController and don't implement this handler, by default, it will automatically load all app options from the config file. If you decide to overide the default handler, you should call:

AppDataController.on_plugin_enable(self)

to retain this functionality.

on_plugin_enabled(*env*, *plugin*)
Handler called to notify that a plugin has been enabled.

Note that this signal is broadcast to all plugins implementing the *IPlugin* interface, whereas the *on_plugin_enable()* method is called directly on the plugin that is being enabled.

Parameters

- **env** (*str*) – pyutilib plugin environment.
- **plugin** (*str*) – Plugin name.

on_protocol_changed()
Handler called when a protocol is modified.

on_protocol_pause()
Handler called when a protocol is paused.

on_protocol_run()
Handler called when a protocol starts running.

on_protocol_swapped(*old_protocol*, *protocol*)
Handler called when a different protocol is swapped in (e.g., when a protocol is loaded or a new protocol is created).

Parameters

- **old_protocol** (microdrop.protocol.Protocol) – Original protocol.
- **protocol** (microdrop.protocol.Protocol) – New protocol.

on_step_complete(*plugin_name*, *return_value=None*)
Handler called whenever a plugin completes a step.

Returns

- 'Repeat': repeat the step
- 'Fail': unrecoverable error (stop the protocol)

Return type str or None

on_step_created(*step_number*)
Handler called whenever a new step is created.

Parameters **step_number** (*int*) – New step number.

on_step_options_changed(*plugin*, *step_number*)
Handler called when the step options are changed for a particular plugin. This will, for example, allow for GUI elements to be updated based on step specified.

Parameters

- **plugin** (*SingletonPlugin*) – Plugin instance for which the step options changed.
- **step_number** (*int*) – Step number that the options changed for.

on_step_options_swapped(*plugin*, *old_step_number*, *step_number*)
Handler called when the step options are changed for a particular plugin. This will, for example, allow for GUI elements to be updated based on step specified.

Parameters

- **plugin** (*SingletonPlugin*) – Plugin instance for which the step options changed.
- **old_step_number** (*int*) – Original step number.
- **step_number** (*int*) – New step number.

on_step_run()
Handler called whenever a step is executed. Note that this signal is only emitted in realtime mode or if a protocol is running.

Plugins that handle this signal must emit the *on_step_complete()* signal once they have completed the step. The protocol controller will wait until all plugins have completed the current step before proceeding.

Returns

- `'Repeat'`: repeat the step
- `'Fail'`: unrecoverable error (stop the protocol)

Return type str or None

on_step_swapped(*old_step_number*, *step_number*)
Handler called when the current step is swapped.

Parameters

- **old_step_number** (*int*) – Original step number.
- **step_number** (*int*) – New step number.

class microdrop.interfaces.**IWaveformGenerator**
Bases: pyutilib.component.core.core.Interface

Methods

set_frequency(*frequency*)
Set the waveform frequency.

Parameters **frequency** – frequency in Hz

set_voltage(*voltage*)
Set the waveform voltage.

Parameters **voltage** – RMS voltage

1.1.9 logger Module

class microdrop.logger.**CustomHandler**
Bases: logging.Handler

Attributes

Methods

emit(*record*)

1.1.10 microdrop Module

1.1.11 plugin_helpers Module

class microdrop.plugin_helpers.**AppDataController**
Bases: object

Methods

get_app_fields()

get_app_form_class()

get_app_value(*key*)

get_app_values()

get_default_app_options()

static get_plugin_app_values(*plugin_name*)

on_plugin_enable()
Handler called once the plugin instance has been enabled.

set_app_values(*values_dict*)

class microdrop.plugin_helpers.**PluginMetaData**
Bases: tuple

Attributes

Methods

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

as_dict()

static from_dict(*data*)

package_name
Alias for field number 0

plugin_name
Alias for field number 1

version
Alias for field number 2

class microdrop.plugin_helpers.**StepOptionsController**
Bases: object

Methods

get_default_step_options()

static get_plugin_step_values(*plugin_name*, *step_number=None*)

get_step(*step_number*)

get_step_fields()

get_step_form_class()

get_step_number(*default*)

get_step_options(*step_number=None*)

get_step_value(*name*, *step_number=None*)

get_step_values(*step_number=None*)

set_step_values(*values_dict*, *step_number=None*)
Consider a scenario where most step options are simple types that are supported by *flatland* and can be listed in *StepOptions* (e.g., *Integer*, *Boolean*, etc.), but there is at least one step option that is a type not supported by *flatland*, such as a *numpy.array*.

Currently, this requires custom handling for all methods related to step options, as in the case of the DMF control board. Instead, during validation of step option values, we could simply exclude options that are not listed in the *StepOptions* definition from the validation, but pass along *all* values to be saved in the protocol.

This should maintain backwards compatibility while simplifying the addition of arbitrary Python data types as step options.

microdrop.plugin_helpers.**from_dict**(*data*)

microdrop.plugin_helpers.**get_plugin_info**(*plugin_root*)

Return a named tuple: (package_name, plugin_name, version)

If plugin is not installed or invalid, returned tuple will be None.

microdrop.plugin_helpers.**hub_execute**(**args*, ***kwargs*)

microdrop.plugin_helpers.**hub_execute_async**(**args*, ***kwargs*)

1.1.12 plugin_manager Module

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

class microdrop.plugin_manager.**ScheduleRequest**
Bases: tuple

Attributes

Methods

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

after
Alias for field number 1

before
Alias for field number 0

microdrop.plugin_manager.**disable**(*name*, *env='microdrop.managed'*)

microdrop.plugin_manager.**emit_signal**(*function*, *args=None*, *interface=<class 'microdrop.interfaces.IPlugin'>*)

microdrop.plugin_manager.**enable**(*name*, *env='microdrop.managed'*)

microdrop.plugin_manager.**get_observers**(*function*, *interface=<class 'microdrop.interfaces.IPlugin'>*)

microdrop.plugin_manager.**get_plugin_names**(*env=None*)

microdrop.plugin_manager.**get_plugin_package_name**(*class_name*)

microdrop.plugin_manager.**get_schedule**(*observers*, *function*)

microdrop.plugin_manager.**get_service_class**(*name*, *env='microdrop.managed'*)

microdrop.plugin_manager.**get_service_instance**(*class_*, *env='microdrop.managed'*)

microdrop.plugin_manager.**get_service_instance_by_name**(*name*, *env='microdrop.managed'*)

microdrop.plugin_manager.**get_service_instance_by_package_name**(*name*, *env='microdrop.managed'*)

microdrop.plugin_manager.**get_service_names**(*env='microdrop.managed'*)

microdrop.plugin_manager.**load_plugins**(*plugins_dir='plugins'*)

microdrop.plugin_manager.**log_summary**()

microdrop.plugin_manager.**post_install**(*install_path*)

1.1.13 `protocol` Module

class `microdrop.protocol.`**`Protocol`**(*name=None*)

Attributes

Methods

`class_version` = **'0.2.0'**

`current_step`()

`delete_step`(*step_number*)

`delete_steps`(*step_ids*)

`first_step`()

`get_data`(*plugin_name*)

`get_step`(*step_number=None*)

`get_step_number`(*default*)

`get_step_values`(*plugin_name*)

`goto_step`(*step_number*)

`insert_step`(*step_number=None*, *value=None*, *notify=True*)

`insert_steps`(*step_number=None*, *count=None*, *values=None*)

`last_step`()

classmethod `load`(*filename*)
Load a Protocol from a file.

Parameters **`filename`** – path to file.

Raises

- `TypeError` – file is not a Protocol.
- `FutureVersionError` – file was written by a future version of the software.

`next_repetition`()

`next_step`()

`plugin_name_lookup`(*name*, *re_pattern=False*)

`plugins`

prev_step()

save(*filename*, *format='pickle'*)

set_data(*plugin_name*, *data*)

to_frame()

Returns

Data frame with multi-index columns, indexed first by plugin name, then by plugin step field name.

Note: If an exception is encountered while processing a plugin value, the plugin causing the exception is skipped and protocol values related to the plugin are not included in the result.

Return type pandas.DataFrame

See also:

to_json(), *to_ndjson()*

to_json()

Returns

json-encoded dictionary, with two top-level keys:

- **keys:**
  - Each key is a list containing a plugin name and a corresponding step field name.
- **values:**
  - Maps to list of records (i.e., lists), one per protocol step.

Each record in the `values` list may be *zipped together* with `keys` to yield a plugin field name to value mapping for a single protocol step.

Return type str

See also:

to_frame(), *to_ndjson()*

to_ndjson(*ostream=None*)

Write protocol as newline delimted JSON (i.e., ndjson, see specification).

Each subsequent line in the output is a nested JSON record, list), one line per protocol step. The keys of the top-level object of each record correspond to plugin names. The second-level keys correspond to the step field name.

Parameters **ostream** (*file-like, optional*) – Output stream to write to.

Returns If `ostream` parameter is `None`, return output as string.

Return type None or str

See also:

to_frame(), *to_json()*

class microdrop.protocol.**Step**(*plugin_data=None*)

Bases: object

Attributes

Methods

copy()

get_data(*plugin_name*)

plugin_name_lookup(*name*, *re_pattern=False*)

plugins

set_data(*plugin_name*, *data*)

microdrop.protocol.**protocol_to_frame**(*protocol_i*)

Parameters **protocol_i** (microdrop.protocol.Protocol) – Microdrop protocol.

> **Note:** A Microdrop protocol object is stored as pickled in the `protocol` file in each experiment log directory.

Returns

Data frame with rows indexed by 0-based step number and columns indexed (multi-index) first by plugin name, then by step field name.

> **Note:** Values may be Python objects. In future versions of Microdrop, values *may* be restricted to json compatible types.

Return type pandas.DataFrame

microdrop.protocol.**protocol_to_json**(*protocol*)

Parameters **protocol** (microdrop.protocol.Protocol) – Microdrop protocol.

> **Note:** A Microdrop protocol object is stored as pickled in the `protocol` file in each experiment log directory.

Returns

json-encoded dictionary, with two top-level keys:

- **keys:**
  - Each key is a list containing a plugin name and a corresponding step field name.
- **values:**
  - Maps to list of records (i.e., lists), one per protocol step.

Each record in the `values` list may be *zipped together* with `keys` to yield a plugin field name to value mapping for a single protocol step.

Return type str

1.1.14 Subpackages

bin Package

create_portable_config Module

microdrop.bin.create_portable_config.**main**(*output_dir*)

microdrop.bin.create_portable_config.**parse_args**(*args=None*)
Parses arguments, returns (options, args).

latest_versions Module

microdrop.bin.latest_versions.**get_latest_version_content**(*server_url='http://microfluidics.utoronto.ca/upda*

core_plugins Package

Subpackages

device_info_plugin Package

device_info_plugin Package

on_plugin_install Module

release Module

rename Module

electrode_controller_plugin Package

electrode_controller_plugin Package

on_plugin_install Module

release Module

rename Module

zmq_hub_plugin Package

zmq_hub_plugin Package

class microdrop.core_plugins.zmq_hub_plugin.**MicrodropHub**(*query_uri*, *name='hub'*)
Bases: zmq_plugin.hub.Hub

Attributes

Methods

on_command_recv(*msg_frames*)

class microdrop.core_plugins.zmq_hub_plugin.**ZmqHubPlugin**
Bases: pyutilib.component.core.core.SingletonPlugin, *microdrop.plugin_helpers.AppDataContr*

This class is automatically registered with the PluginManager.

Methods

AppFields
alias of Form

on_plugin_disable()
Handler called once the plugin instance is disabled.

on_plugin_enable()
Handler called once the plugin instance is enabled.

Note: if you inherit your plugin from AppDataController and don't implement this handler, by default, it will automatically load all app options from the config file. If you decide to overide the default handler, you should call:

AppDataController.on_plugin_enable(self)

to retain this functionality.

plugin_name = 'wheelerlab.zmq_hub_plugin'

on_plugin_install Module

release Module

rename Module

microdrop.core_plugins.zmq_hub_plugin.rename.**main**(*root*, *old_name*, *new_name*)

microdrop.core_plugins.zmq_hub_plugin.rename.**parse_args**(*args=None*)

Parses arguments, returns (options, args).

gui Package

app_options_controller Module

cairo_view Module

channel_sweep Module

config_controller Module

Copyright 2011 Ryan Fobel

This file is part of Microdrop.

Microdrop is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Microdrop is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Microdrop. If not, see <http://www.gnu.org/licenses/>.

class microdrop.gui.config_controller.**ConfigController**

Bases: pyutilib.component.core.core.SingletonPlugin

Methods

get_schedule_requests(*function_name*)

Returns a list of scheduling requests (i.e., ScheduleRequest instances) for the function specified by function_name.

on_app_exit()

on_app_options_changed(*plugin_name*)

on_dmf_device_changed(*dmf_device*)

on_dmf_device_swapped(*old_dmf_device*, *dmf_device*)

on_plugin_enable()

on_protocol_changed()

on_protocol_swapped(*old_protocol*, *protocol*)

dmf_device_controller Module

dmf_device_controller.video Module

dmf_device_view.video Module

experiment_log_controller Module

field_filter_controller Module

main_window_controller Module

plugin_download_dialog Module

plugin_manager_controller Module

plugin_manager_dialog Module

protocol_controller Module

protocol_grid_controller Module

tests Package

test_dmf_device Module

test_experiment_log Module

test_protocol Module

update_dmf_control_board Module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

m

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

S

T

V

Z