## Contents

1 Contents

1.1 Install the MSL Package Manager .................................................. 3
1.2 Command Line Interface ................................................................. 3
1.3 API Usage ..................................................................................... 8
1.4 MSL Package Manager API Documentation ..................................... 10
1.5 “create” ReadMe .............................................................................. 18
1.6 MSL Developers Guide ............................................................... 23
1.7 License .......................................................................................... 34
1.8 Developers .................................................................................... 35
1.9 Release Notes ................................................................................. 35

2 Indices and tables

Python Module Index ........................................................................... 45

Index .................................................................................................. 47
The **MSL Package Manager** allows you to install, uninstall, update, list and create packages that are used at the Measurement Standards Laboratory of New Zealand.

All MSL packages that start with `msl` are part of the `msl` namespace. This allows one to split sub-packages and modules across multiple, separate distribution packages while still maintaining a single, unifying package structure.

All MSL packages are available as GitHub repositories and some have been published as PyPI packages.
1.1 Install the MSL Package Manager

To install the MSL Package Manager run:

```
pip install msl-package-manager
```

1.1.1 Dependencies

- Python 2.7, 3.5+
- setuptools
- colorama

1.2 Command Line Interface

Once the MSL Package Manager has been installed you will be able to install, uninstall, update, list and create MSL packages by using the command line interface.

*You can also directly call these functions through the API.*

**Attention:** Since MSL packages are part of a namespace, uninstalling MSL packages using `pip uninstall msl-<package name>` will break the namespace. Therefore, it is recommended to use `msl uninstall <package name>` to uninstall MSL packages.

**Note:** The information about the MSL repositories that are available on GitHub and the MSL packages on PyPI are cached for 24 hours after you request information about a repository or package. After 24 hours the information will be corrupted.
hours a subsequent request will automatically update the GitHub or PyPI cache. To force the cache to be updated immediately include the --update-cache flag.

To read the help documentation from the command line, run

```
msl --help
```

or, for help about a specific command (for example, the install command), run

```
msl install --help
```

### 1.2.1 install

Install all MSL packages that are available

```
msl install --all
```

Install all MSL packages without asking for confirmation

```
msl install --all --yes
```

Install a specific MSL package, for example msl-loadlib (you can ignore the msl- prefix)

```
msl install loadlib
```

Install a package from a specific GitHub branch (by default the master branch is used if the package is not available on PyPI)

```
msl install loadlib --branch develop
```

Install a package from a specific GitHub tag

```
msl install loadlib --tag v0.3.0
```

Install multiple MSL packages

```
msl install loadlib equipment qt
```

Install a specific version of a package (the package must be available as a PyPI package)

```
msl install loadlib==0.6.0
```

Specify a version range of a package – make sure to surround the package and version range in quotes (the package must be available as a PyPI package)

```
msl install "loadlib>=0.4,<0.6"
```

Install a package and its extra dependencies

```
msl install loadlib[com]
```

You can also use a wildcard, for example, to install all packages that start with pr-
**msl install pr-**

You can also include all options that the pip install command accepts, run pip help install for more details

**msl install loadlib equipment qt --user --retries 10**

### 1.2.2 uninstall

Uninstall all MSL packages (except for the msl-package-manager)

**msl uninstall --all**

**Tip:** You can also use remove as an alias for uninstall, e.g., msl remove --all

**Note:** To uninstall the MSL Package Manager run pip uninstall msl-package-manager. Use with caution. If you uninstall the MSL Package Manager and you still have other MSL packages installed then you may break the MSL namespace.

Uninstall all MSL packages without asking for confirmation

**msl uninstall --all --yes**

Uninstall a specific MSL package, for example msl-loadlib (you can ignore the msl- prefix)

**msl uninstall loadlib**

Uninstall multiple MSL packages

**msl uninstall loadlib equipment qt**

You can also include all options that the pip uninstall command accepts, run pip help uninstall for more details

**msl uninstall io qt --no-python-version-warning**

### 1.2.3 update

Update all MSL packages that are installed

**msl update --all**

**Tip:** You can also use upgrade as an alias for update, e.g., msl upgrade --all

Update all MSL packages without asking for confirmation

---

**1.2. Command Line Interface**
msl update --all --yes

Update a specific MSL package, for example `msl-loadlib` (you can ignore the `msl-` prefix)

```
msl update loadlib
```

Update to a package that was released <24 hours ago

```
msl update loadlib --update-cache
```

Update a package from a specific GitHub branch (by default the `master` branch is used if the package is not available on PyPI)

```
msl update loadlib --branch develop
```

Update a package from a specific GitHub tag

```
msl update loadlib --tag v0.3.0
```

Update multiple MSL packages

```
msl update loadlib equipment qt
```

You can also include all options that the `pip install` command accepts, run `pip help install` for more details (the `--upgrade` option is automatically included by default)

```
msl update loadlib io --no-deps
```

### 1.2.4 list

List all MSL packages that are installed

```
msl list
```

List all MSL repositories that are available on GitHub

```
msl list --github
```

List all MSL packages that are available on PyPI

```
msl list --pypi
```

Update the GitHub `cache` and then list all repositories that are available

```
msl list --github --update-cache
```

Update the PyPI `cache` and then list all packages that are available

```
msl list --pypi --update-cache
```

Show the information about the repositories (includes information about the branches and the tags) in `JSON` format
1.2.5 create

To create a new package called MyPackage, run

```bash
msl create MyPackage
```

This will create a new folder (in the current working directory) called msl-MyPackage.

To import the package you would use

```python
from msl import MyPackage
```

Running the `create` command attempts to determine your user name and email address from your git account to use as the author and email values in the files that it creates. You do not need git to be installed to use the `create` command, but it helps to make the process more automated. Optionally, you can specify the name to use for the author and the email address by passing additional arguments

```bash
msl create MyPackage --author Firstname Lastname --email my.email@address.com
```

You can also specify where to create the package (instead of the default location which is in the current working directory) by specifying a value for the `--dir` argument and to automatically accept the default author name and email address values by adding the `--yes` argument

```bash
msl create MyPackage --yes --dir D:\create\package\here
```

To create a new package that is part of a different namespace, you can run

```bash
msl create monochromator --namespace pr
```

To import this package you would use

```python
from pr import monochromator
```

To create a new package that is not part of a namespace, run

```bash
msl create mypackage --no-namespace
```

To import this package you would use

```python
import mypackage
```

1.2.6 authorize

When requesting information about the MSL repositories that are available on GitHub there is a limit to how often you can send requests to the GitHub API (this is the primary reason for caching the information). If you have a GitHub account and include your username and a personal access token with each request then this limit is increased. If you do not have a GitHub account then you could sign up to create an account.
By running this command you will be asked for your GitHub username and personal access token so that you send authorized requests to the GitHub API.

```
msl authorize
```

**Tip:** You can also use `authorise` as an alias for `authorize`, e.g., `msl authorise`

**Important:** Your GitHub username and personal access token are saved in plain text in the file that is created. You should set the file permissions provided by your operating system to ensure that your GitHub credentials are safe.

## 1.3 API Usage

In cases where using the *command-line interface* is not desired, you can use the *API* functions directly to install, uninstall, update, list and create MSL packages.

First, import the **MSL Package Manager**

```
>>> from msl import package_manager as pm
```

**Tip:** You can set what information is displayed on the screen by changing the *Logging Levels*

```
>>> import logging

>>> pm.set_log_level(logging.INFO)
```

### 1.3.1 install

*install* the **msl-network** and **msl-qt** packages

```
>>> pm.install('network', 'qt')
The following MSL packages will be INSTALLED:

msl-network 0.5.0  [PyPI]
msl-qt         [GitHub]
Proceed ([y]/n)?
```

### 1.3.2 uninstall

*uninstall* the **msl-loadlib** package

```
>>> pm.uninstall('loadlib')
The following MSL packages will be REMOVED:

msl-loadlib 0.6.0
```

(continues on next page)
1.3.3 update

Update the `msl-loadlib` package

```python
>>> pm.update('loadlib')
The following MSL packages will be UPDATED:

  msl-loadlib 0.6.0 --> 0.7.0 [PyPI]

Proceed ([y]/n)?
```

1.3.4 list

Display the information about the MSL packages that are installed, see `info()`

```python
>>> pm.info()
MSL Package          Version                      Description
------------------- ------- -----------------------------------------------
msl-loadlib 0.6.0    Load a shared library (and access a 32-bit
                     library from 64-bit Python)
msl-package-manager 2.4.0 Install, uninstall, update, list and create
                     MSL packages
```

Display the information about the MSL repositories that are available

```python
>>> pm.info(from_github=True)
MSL Repository       Version                      Description
--------------------- ------- -----------------------------------------------
GTC 1.2.1             The GUM Tree Calculator for Python
Quantity-Value 0.1.0  A package that supports physical quantity-
                     correctness in Python code
msl-equipment         Manage and communicate with equipment in the
msl-loadlib 0.7.0     Load a shared library (and access a 32-bit
                     library from 64-bit Python)
msl-network 0.5.0     Concurrent and asynchronous network I/O
msl-package-manager  2.4.0 Install, uninstall, update, list and create
                     MSL packages
msl-io                Read and write data files
msl-network 0.5.0     Concurrent and asynchronous network I/O
msl-package-manager  2.4.0 Install, uninstall, update, list and create
                     MSL packages
msl-qt                Custom Qt components for the user interface
```

Get a dictionary of all MSL packages that are `installed()`

```python
>>> pkgs = pm.installed()
>>> for pkg, info in pkgs.items():
    ...    print(pkg, info)
    ...
```

(continues on next page)
msl-loadlib {'version': '0.6.0', 'description': 'Load a shared library (and access a 32-bit library from 64-bit Python)', 'repo_name': 'msl-loadlib'}
msl-package-manager {'version': '2.4.0', 'description': 'Install, uninstall, update, list and create MSL packages', 'repo_name': 'msl-package-manager'}

Get a dictionary of all MSL repositories on GitHub, see `github()`

```python
>>> pkgs = pm.github()
>>> for key, value in pkgs['msl-package-manager'].items():
...     print('{}/{}: /'.format(key, value))
...
description: Install, uninstall, update, list and create MSL packages
version: 2.4.0
tags: ['v2.4.0', 'v2.3.0', 'v2.2.0', 'v2.1.0', 'v2.0.0', 'v1.5.1', 'v1.5.0', 'v1.4.1', 'v1.4.0', 'v1.3.0', 'v1.2.0', 'v1.1.0', 'v1.0.3', 'v1.0.2', 'v1.0.1', 'v1.0.0', 'v0.1.0']
branches: ['master']
```

Get a dictionary of all MSL packages on PyPI, see `pypi()`

```python
>>> pkgs = pm.pypi()
>>> pkgs['msl-package-manager']
{'description': 'Install, uninstall, update, list and create MSL packages', 'version': '2.4.0'}
```

1.3.5 create

create a new MSL-MyPackage package

```python
>>> pm.create('MyPackage', author='my name', email='my@email.com', directory='D:/create/here')
Created msl-MyPackage in 'D:/create/here\msl-MyPackage'
```

1.3.6 authorize

Create an authorization file for the GitHub API, see `authorize()`

```python
>>> pm.authorize()
Enter your GitHub username [default: ...]: ??
Enter your GitHub personal access token: ??
```

1.4 MSL Package Manager API Documentation

The root package is
msl.package_manager

Install, uninstall, update, list and create MSL packages.

which has the following functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorize</td>
<td>Create an authorization file for the GitHub API.</td>
</tr>
<tr>
<td>create</td>
<td>Create a new package.</td>
</tr>
<tr>
<td>github</td>
<td>Get the information about the MSL repositories that are available on GitHub.</td>
</tr>
<tr>
<td>info</td>
<td>Show information about MSL packages.</td>
</tr>
<tr>
<td>install</td>
<td>Install MSL packages.</td>
</tr>
<tr>
<td>installed</td>
<td>Get the information about the MSL packages that are installed.</td>
</tr>
<tr>
<td>set_log_level</td>
<td>Set the logging level.</td>
</tr>
<tr>
<td>pypi</td>
<td>Get the information about the MSL packages that are available on PyPI.</td>
</tr>
<tr>
<td>uninstall</td>
<td>Uninstall MSL packages.</td>
</tr>
<tr>
<td>update</td>
<td>Update MSL packages.</td>
</tr>
</tbody>
</table>

1.4.1 Package Structure

msl.package_manager package

Install, uninstall, update, list and create MSL packages.

The following constants are available.

msl.package_manager.version_info = version_info(major=2, minor=4, micro=1, releaselevel='dev0')
Contains the version information as a (major, minor, micro, releaselevel) tuple.

Type namedtuple

msl.package_manager.authorize module

Create an authorization file for the GitHub API.

msl.package_manager.authorize.authorize(username=None, token=None)
Create an authorization file for the GitHub API.

When requesting information about the MSL repositories that are available on GitHub there is a limit to how often you can send requests to the GitHub API. If you have a GitHub account and include your username and a personal access token with each request then this limit is increased.

Important: Calling this function will create a file that contains your GitHub username and a personal access token so that GitHub requests are authorized. Your username and personal access token are saved in plain text in the file that is created. You should set the file permissions provided by your operating system to ensure that your GitHub credentials are safe.

New in version 2.3.0.

1.4. MSL Package Manager API Documentation 11
Changed in version 2.4.0: Renamed the `password` keyword argument to `token` (using a password is deprecated for the GitHub API)

**Parameters**

- **username (str, optional)** – The GitHub username. If `None` then you will be asked for the `username`.

- **token (str, optional)** – A GitHub personal access token for `username`. If `None` then you will be asked for the `token`.

### msl.package_manager.cli module

Main entry point to either `install`, `uninstall`, `update`, `list` or `create` MSL packages using the command-line interface (CLI).

```python
msl.package_manager.cli.configure_parser()
    ArgumentParser: Returns the argument parser.
```

```python
msl.package_manager.cli.main(*args)
    Main entry point to either `install`, `uninstall`, `update`, `list` or `create` MSL packages using the CLI.
```

```python
msl.package_manager.cli.parse_args(args)
    Parse arguments.
```

**Parameters**

- **args (list of str)** – The arguments to parse.

**Returns**

An `argparse.Namespace` or `None` if there was an error.

### msl.package_manager.cli_argparse module

Custom argument parsers.

```python
class msl.package_manager.cli_argparse.ArgumentParser(*args, **kwargs)
    Bases: argparse.ArgumentParser
    A custom argument parser.
```

```python
    contains_package_names()
        bool: Whether package names were specified or the --all flag was used.
```

```python
    get_command_name()
        str: Returns the name of the command, e.g., install, list,...
```

```python
msl.package_manager.cli_argparse.add_argument_all(parser)
    Add an --all argument to the parser.
```

```python
msl.package_manager.cli_argparse.add_argument_branch(parser)
    Add a --branch argument to the parser.
```

```python
msl.package_manager.cli_argparse.add_argument_disable_mslpm_version_check(parser)
    Add a --disable-msplpm-version-check argument to the parser.
```

```python
msl.package_manager.cli_argparse.add_argument_package_names(parser)
    Add a --names argument to the parser.
```

```python
msl.package_manager.cli_argparse.add_argument_quiet(parser)
    Add a --quiet argument to the parser.
```
msl.package_manager.cli_argparse.add_argument_tag(parser)
Add a --tag argument to the parser.

msl.package_manager.cli_argparse.add_argument_update_cache(parser)
Add an --update-cache argument to the parser.

msl.package_manager.cli_argparse.add_argument_yes(parser)
Add a --yes argument to the parser.

msl.package_manager.cli_authorize module

Command line interface for the authorize command.

msl.package_manager.cli_authorize.add_parser_authorize(parser, name='authorize')
Add the authorize command to the parser.

msl.package_manager.cli_authorize.execute(args, parser)
Executes the authorize command.

msl.package_manager.cli_create module

Command line interface for the create command.

msl.package_manager.cli_create.add_parser_create(parser)
Add the create command to the parser.

msl.package_manager.cli_create.execute(args, parser)
Executes the create command.

msl.package_manager.cli_install module

Command line interface for the install command.

msl.package_manager.cli_install.add_parser_install(parser)
Add the install command to the parser.

msl.package_manager.cli_install.execute(args, parser)
Executes the install command.

msl.package_manager.cli_list module

Command line interface for the list command.

msl.package_manager.cli_list.add_parser_list(parser)
Add the list command to the parser.

msl.package_manager.cli_list.execute(args, parser)
Executes the list command.
**msl.package_manager.cli_uninstall module**

Command line interface for the `uninstall` command.

```python
msl.package_manager.cli_uninstall.add_parser_uninstall(parser, name='uninstall')
```

Add the `uninstall` command to the parser.

```python
msl.package_manager.cli_uninstall.execute(args, parser)
```

Executes the `uninstall` command.

**msl.package_manager.cli_update module**

Command line interface for the `update` command.

```python
msl.package_manager.cli_update.add_parser_update(parser, name='update')
```

Add the `update` command to the parser.

```python
msl.package_manager.cli_update.execute(args, parser)
```

Executes the `update` command.

**msl.package_manager.create module**

Create a new package.

```python
msl.package_manager.create.create(*names, **kwargs)
```

Create a new package.

**Parameters**

- **`*names`** – The name(s) of the package(s) to create.

- **`**kwargs`** –
  - `author` [str] The name of the author to use for the new package. If `None` then uses `utils.get_username()` to determine the author’s name. Default is `None`.
  - `directory` [str] The directory to create the new package(s) in. If `None` then creates the new package(s) in the current working directory. Default is `None`.
  - `email` [str] The author’s email address. If `None` then uses `utils.get_email()` to determine the author’s email address. Default is `None`.
  - `namespace` [str] The namespace that the package belongs to. If `namespace` is `None` or an empty string then create a new package that is not part of a namespace. Default is the `'msl'` namespace.
  - `yes` [bool] If `True` then don’t ask for verification for the `author` name and for the `email` address. This argument is only used if you do not specify the `author` or the `email` value. The verification step allows you to change the value that was automatically determined before the package is created. The default is to ask for verification before creating the package if the `author` or the `email` value was not specified. Default is `False`.

14 Chapter 1. Contents
msl.package_manager.install module

Install MSL packages.

```
msl.package_manager.install(*names, **kwargs)
```

Install MSL packages.

MSL packages can be installed from PyPI packages (only if a release has been uploaded to PyPI) or from GitHub repositories.

**Note:** If the MSL packages are available on PyPI then PyPI is used as the default URI to install the package. If you want to force the installation to occur from the master branch of the GitHub repository then set `branch='master'`. If the package is not available on PyPI then the master branch is used as the default installation URI.

**Parameters**

- `*names` — The name(s) of the MSL package(s) to install. If not specified then install all MSL packages that begin with the msl- prefix. The msl- prefix can be omitted (e.g., 'loadlib' is equivalent to 'msl-loadlib'). Also accepts shell-style wildcards (e.g., 'pr-*').

- `**kwargs` —
  - `branch` [str] The name of a GitHub branch to use for the installation. If None, and no tag value has been specified, then installs from the master branch. Default is None.
  - `tag` [str] The name of a GitHub tag to use for the installation. Default is None.
  - `update_cache` [bool] The information about the MSL packages that are available on PyPI and about the repositories that are available on GitHub are cached to use for subsequent calls to this function. After 24 hours the cache is automatically updated. Set `update_cache` to be True to force the cache to be updated when you call this function. Default is False.
  - `yes` [bool] If True then don’t ask for confirmation before installing. The default is False (ask before installing).
  - `pip_options` [list of str] Optional arguments to pass to the pip install command, e.g., ['--retries', '10', '--user']

**Attention:** Cannot specify both a branch and a tag simultaneously.

msl.package_manager.uninstall module

Uninstall MSL packages.

```
msl.package_manager.uninstall(*names, **kwargs)
```

Uninstall MSL packages.

**Parameters**
• **names** – The name(s) of the MSL package(s) to uninstall. If not specified then uninstall all MSL packages (except for the MSL Package Manager – in which case use pip uninstall msl-package-manager). The msl- prefix can be omitted (e.g., 'loadlib' is equivalent to 'msl-loadlib'). Also accepts shell-style wildcards (e.g., 'pr-*').

• **kwargs** –
  - **yes** [bool] If True then don’t ask for confirmation before uninstalling. The default is False (ask before uninstalling).
  - **pip_options** [list of str] Optional arguments to pass to the pip uninstall command, e.g., ['--no-python-version-warning']

msl.package_manager.update module

Update MSL packages.

msl.package_manager.update.update(*names, **kwargs)

Update MSL packages.

MSL packages can be updated from PyPI packages (only if a release has been uploaded to PyPI) or from GitHub repositories.

**Note:** If the MSL packages are available on PyPI then PyPI is used as the default URI to update the package. If you want to force the update to occur from the master branch of the GitHub repository then set branch='master'. If the package is not available on PyPI then the master branch is used as the default update URI.

Parameters

• **names** – The name(s) of the MSL package(s) to update. If not specified then update all MSL packages. The msl- prefix can be omitted (e.g., 'loadlib' is equivalent to 'msl-loadlib'). Also accepts shell-style wildcards (e.g., 'pr-*').

• **kwargs** –
  - **branch** [str] The name of a GitHub branch to use for the update. If None, and no tag value has also been specified, then updates the package using the master branch. Default is None.
  - **tag** [str] The name of a GitHub tag to use for the update. Default is None.
  - **update_cache** [bool] The information about the MSL packages that are available on PyPI and about the repositories that are available on GitHub are cached to use for subsequent calls to this function. After 24 hours the cache is automatically updated. Set update_cache to be True to force the cache to be updated when you call this function. Default is False.
  - **yes** [bool] If True then don’t ask for confirmation before updating. The default is False (ask before updating).
- **pip_options** [list of str] Optional arguments to pass to the `pip install --upgrade` command, e.g., `['--upgrade-strategy', 'eager']`

| Attention: | Cannot specify both a **branch** and a **tag** simultaneously. |

| Important: | If you specify a **branch** or a **tag** then the update will be forced. |

### `msl.package_manager.utils` module

Functions for the API.

#### `msl.package_manager.utils.get_email()`

Try to determine the user's email address.

- If `git` is installed then it returns the `user.email` parameter from the user's `git` account to use as the user's email address. If `git` is not installed then returns `None`.

**Returns** str or `None` – The user's email address.

#### `msl.package_manager.utils.get_username()`

Determine the name of the user.

- If `git` is installed then it returns the `user.name` parameter from the user's `git` account. If `git` is not installed or if the `user.name` parameter does not exist then `getpass.getuser()` is used to determine the username.

**Returns** str – The user's name.

#### `msl.package_manager.utils.github(update_cache=False)`

Get the information about the MSL repositories that are available on GitHub.

**Parameters**

- `update_cache` (bool, optional) – The information about the repositories that are available on GitHub are cached to use for subsequent calls to this function. After 24 hours the cache is automatically updated. Set `update_cache` to be `True` to force the cache to be updated when you call this function.

**Returns** dict – The information about the MSL repositories that are available on GitHub.

#### `msl.package_manager.utils.info(from_github=False, from_pypi=False, update_cache=False, as_json=False)`

Show information about MSL packages.

- The information about the packages can be either those that are installed or those that are available as repositories on GitHub or as packages on PyPI.

The default action is to show the information about the MSL packages that are installed.

**Parameters**

- `from_github` (bool, optional) – Whether to show the information about the MSL repositories that are available on GitHub.
• **from_pypi** *(bool, optional)* – Whether to show the information about the MSL packages that are available on PyPI.

• **update_cache** *(bool, optional)* – The information about the MSL packages that are available on PyPI and about the repositories that are available on GitHub are cached to use for subsequent calls to this function. After 24 hours the cache is automatically updated. Set `update_cache` to be `True` to force the cache to be updated when you call this function. If `from_github` is `True` then the cache for the repositories is updated. If `from_pypi` is `True` then the cache for the packages is updated.

• **as_json** *(bool, optional)* – Whether to show the information in JSON format. If enabled then the information about the MSL repositories includes additional information about the branches and tags.

**msl.package_manager.utils.installed()**

Get the information about the MSL packages that are installed.

**Returns** `dict` – The information about the MSL packages that are installed.

**msl.package_manager.utils.pypi(update_cache=False)**

Get the information about the MSL packages that are available on PyPI.

**Parameters** `update_cache` *(bool, optional)* – The information about the MSL packages that are available on PyPI are cached to use for subsequent calls to this function. After 24 hours the cache is automatically updated. Set `update_cache` to be `True` to force the cache to be updated when you call this function.

**Returns** `dict` – The information about the MSL packages that are available on PyPI.

**msl.package_manager.utils.set_log_level(level)**

Set the logging level.

**Parameters** `level` *(int)* – A value from one of the Logging Levels.

1.5 “create” ReadMe

The MSL package that is created by running the `msl create` command contains two scripts to help make development easier: `setup.py` and `condatests.py`.

1.5.1 setup.py

The `setup.py` file (that is created by running `msl create`) includes additional commands that can be used to run unit tests and to create the documentation for your MSL package.

**Note:** The Python packages that are required to execute the following commands (e.g., `pytest` and `sphinx`) are automatically installed (into the `.eggs` directory) if they are not already installed in your environment. Therefore, the first time that you run the following commands it will take longer to finish executing the command because these packages (and their own dependencies) need to be downloaded then installed. If you prefer to install these packages directly into your environment you can run `conda install pytest pytest-cov pytest-runner sphinx`
sphinx_rtd_theme, or if you are not using conda as your package manager then replace conda with pip.

The following command will run all test modules that pytest finds as well as testing all the example code that is located within the docstrings of the source code and in the .rst files in the docs/ directory. To modify the options that pytest will use to run the tests you can edit the [tool:pytest] section in setup.cfg. A coverage report is created in the htmlcov/index.html file. This report provides an overview of which parts of the code have been executed during the tests.

```
python setup.py tests
```

**Warning:** pytest can only load one configuration file and uses the following search order to find that file:

1. pytest.ini - used even if it does not contain a [pytest] section
2. tox.ini - must contain a [pytest] section to be used
3. setup.cfg - must contain a [tool:pytest] section to be used

See the configuration file section for an example if you want to run pytest with custom options without modifying any of these configuration files.

Create the documentation files, uses sphinx-build. The documentation can be viewed by opening docs/_build/html/index.html

```
python setup.py docs
```

Automatically create the API documentation from the docstrings in the source code, uses sphinx-apidoc. The files are saved to docs/_autosummary

```
python setup.py apidocs
```

**Attention:** By default, the docs/_autosummary directory that is created by running this command is automatically generated (overwrites existing files). As such, it is excluded from the repository (i.e., this directory is specified in the .gitignore file). If you want to keep the files located in docs/_autosummary you should rename the directory to, for example, docs/_api and then the changes made to the files in the docs/_api directory will be kept and can be included in the repository.

You can view additional help for setup.py by running

```
python setup.py --help
```

or

```
python setup.py --help-commands
```

1.5.2 condatests.py

1.5. “create” ReadMe
Important: The following assumes that you are using conda as your environment manager.

Additionally, there is a `condatests.py` file that is created by running `msl create`. This script will run the tests in all specified conda environments. At the time of writing this script, tox and conda were not compatible and so this script provided a way around this issue.

You can either pass options from the command line or by creating a configuration file.

**command line**

`condatests.py` accepts the following command-line arguments:

- `--create` - the Python version numbers to use to create conda environments (e.g., 2 3.6 3.7.2)
- `--include` - the conda environments to include (supports regex)
- `--exclude` - the conda environments to exclude (supports regex)
- `--requires` - additional packages to install for the tests (can also be a path to a file)
- `--command` - the command to execute with each conda environment
- `--ini` - the path to a configuration file
- `--list` - list the conda environments that will be used for the tests and then exit

You can view the help for `condatests.py` by running:

```
python condatests.py --help
```

Run the tests with all conda environment’s using the `python -m pytest` command. This assumes that a configuration file does not exist (which could change the default options).

```
python condatests.py
```

Run the tests with all conda environments that include `py` in the environment name

```
python condatests.py --include py
```

Run the tests with all conda environments but exclude those that contain `py26` and `py33` in the environment name

```
python condatests.py --exclude py26 py33
```

Tip: Since a regex search is used to filter the environment names that follow the `--exclude` (and also the `--include`) option, the above command could be replaced with `--exclude "py(26|33)"`. Surrounding the regex pattern with a `"` is necessary so that the OR, `|`, regex symbol is not mistaken for a pipe symbol.

Run the tests with all conda environments that include `dev` in the environment name but exclude those with `dev33` in the environment name

```
python condatests.py --include dev --exclude dev33
```
Create new conda environments for the specified Python versions (if the minor or micro version numbers are not specified then the latest Python version that is available to conda will be installed). After the test finishes the newly-created environment is removed. For example, the following command will create environments for the latest Python 2.x.x version, for the latest Python 3.6.x version and for Python 3.7.4 and exclude all environments that already exist

```
python condatests.py --create 2 3.6 3.7.4 --exclude .
```

You can also mix the --create, --include and --exclude arguments

```
python condatests.py --create 3.7 --include dev --exclude dev33
```

Run the tests with all conda environments using the command nosetests

```
python condatests.py --command nosetests
```

Run the tests with all conda environments using the command unittest discover -s tests/

```
python condatests.py --command "unittest discover -s tests/"
```

Run the tests with all conda environments using the command unittest discover -s tests/ and ensure that all the packages specified in a requirements file are installed in each environment

```
python condatests.py --command "unittest discover -s tests/" --requires my_˓→requirements.txt
```

List all conda environments that will be used for the tests and then exit

```
python condatests.py --list
```

You can also use --show as an alias for --list

```
python condatests.py --show
```

List the conda environments that include dev in the environment name and then exit

```
python condatests.py --include dev --list
```

Specify the path to a condatests.ini file

```
python condatests.py --ini C:\Users\Me\my_condatests_config.ini
```

**configuration file**

In addition to passing *command line* options, you can also save the options in an *condatests.ini* configuration file. This is a standard ini-style configuration file with the options create, include, exclude, command and requires specified under the [envs] section.

If a *condatests.ini* configuration file exists in the current working directory then it will automatically be loaded by running

```
python condatests.py
```

Alternatively, you can also specify the path to the configuration file from the command line
You can pass in command-line arguments as well as reading from the configuration file. The following will load the `condatests.ini` file in the current working directory, print the conda environments that will be used for the tests and then exit.

```shell
python condatests.py --show
```

Since every developer can name their environments to be anything that they want, the `condatests.ini` file is included in `.gitignore`.

The following are example `condatests.ini` files.

**Example 1:** Run `python -m pytest` (see `setup.py`) with all conda environments except for the base environment.

```
[envs]
exclude=base
```

**Example 2:** Run `python -m pytest` with all conda environments that include the text `py` in the name of the environment but exclude the environments that contain `py33` in the name (recall that a regex search is used to filter the environment names).

```
[envs]
include=py
exclude=py33
```

**Example 3:** Run `python -m pytest` only with newly-created conda environments, exclude all environments that already exist and ensure that `scipy` is installed in each new environment (if the minor or micro version numbers of the Python environments are not specified then the latest Python version that is available to conda will be installed).

```
[envs]
create=2 3.5 3.6 3.7
exclude=.
requires=scipy
```

**Example 4:** Run `python -m pytest` with newly-created conda environments and all conda environments that already exist that contain the text `dev` in the name of the environment except for the `dev33` environment.

```
[envs]
create=3.6 3.7.3 3.7.4
include=dev
exclude=dev33
```

**Example 5:** Run `unittest`, for all modules in the `tests` directory, with all conda environments that include the text `dev` in the environment name.

```
[envs]
include=dev
command=unittest discover -s tests/
```

**Example 6:** Run `pytest` with customized options (i.e., ignoring any `pytest.ini`, `tox.ini` or `setup.cfg` files that might exist) with the specified conda environments.
[envs]
create=3.7
include=dev27 myenvironment py36
command=pytest -c condatests.ini

[pytest]
addopts =
-x
--verbose

Note: The environment names specified in the create, include, exclude and requires option can be separated by a comma, by whitespace or both. So, include=py27,py36,py37, include=py27, py36, py37 and include=py27, py36, py37 are all equivalent.

1.6 MSL Developers Guide

This guide shows you how to:\(^1\):

- **Install and set up Python, Git and PyCharm**
- **Commit changes to a repository**
- **Use the setup.py and condatests.py scripts**
- **Edit source code using the style guide**

and describes one way to set up an environment to develop Python programs. The guide does not intend to imply that the following is the best way to develop programs in the Python language.

1.6.1 Install and set up Python, Git and PyCharm

This section uses the MSL-LoadLib repository as an example of a repository that one would like to clone and import into PyCharm.

The following instructions are written for a Windows x64 operating system. To install the same software on a Debian architecture, such as Ubuntu, run

```
sudo apt update
sudo apt install git snapd
sudo snap install pycharm-community --classic
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda*
```

and answer the questions that you are asked. After running these commands you can follow the appropriate steps below.

\(^1\) Software is identified in this guide in order to specify the installation and configuration procedure adequately. Such identification is not intended to imply recommendation or endorsement by the Measurement Standards Laboratory of New Zealand, nor is it intended to imply that the software identified are necessarily the best available for the purpose.
1. Download a 64-bit version of Miniconda.
2. Install Miniconda. It is recommended to Register Anaconda but not to Add it to your PATH.
3. Open the Anaconda Command Prompt
and then enter the following command to update all Miniconda packages:

```
conda update --all
```

4. It is usually best to create a new virtual environment for each Python project that you are working on to avoid possible conflicts between the packages that are required for each Python project or to test the code against different versions of Python (i.e., it solves the *Project X depends on version 1.x but Project Y depends on version 4.x* dilemma).

In the Anaconda Command Prompt create a new `py37` virtual environment (you can pick another name, `py37` is just an example of a name) and install the Python 3.7 interpreter in this environment *(NOTE: You can also create conda environment’s from within PyCharm if you are not comfortable with the command line, see Step 9)*

```
conda create --name py37 python=3.7
```

You may also want to create another virtual environment so that you can run the code against another Python version. For example, here is an example of how to create a Python 2.7 virtual environment named `py27`:

```
conda create --name py27 python=2.7
```

5. Create a GitHub account *(if you do not already have one)*.

6. Download and install `git (accept the default settings)`. This program is used as the version control system.
7. Download and install the Community Edition of PyCharm to use as an IDE. This IDE is free to use and it provides a lot of the features that one expects from an IDE. When asked to Create associations check the .py checkbox and you can also create a shortcut on the desktop (you can accept the default settings for everything else that you are asked during the installation).

8. Run PyCharm and perform the following:
   a) Import settings from a previous version of PyCharm *(if available)*

   ![Import PyCharm Settings From... dialog]

   b) Select the default editor theme *(you can also change the theme later)* and click Skip Remaining and Set Defaults
c) Select the **Git** option from **Check out from Version Control**

![Check out from Version Control](image)

![Git option](image)

d) Click the **Log in to Github...** button

![Log in to Github](image)
and then enter your GitHub account information (see Step 5 above) and click **Log In**

**e) Clone the MSL-LoadLib repository.** Specify the **Directory** where you want to clone the repository *(NOTE: the MSL-LoadLib repository will only appear if you are part of the MSLNZ organisation on GitHub. A list of your own repositories will be available.)*

**f) Open the MSL-LoadLib repository in PyCharm**

9. Add the **py37 virtual environment** that was created in Step 4 as the **Project Interpreter** *(NOTE: you can also create a new conda environment in Step 9d)*
   
a) Press **CTRL+ALT+S** to open the **Settings** window

b) Go to **Project Interpreter** and click on the **gear** button in the top-right corner
c) Select **Add**

d) Select **Conda Environment → Existing environment** and select the **py37 virtual environment** that was created in Step 4 and then click **OK**. You can also create a new environment if you want.
e) Click **Apply** then **OK**

f) If you created a py27 **virtual environment** then repeat Steps 9b-9d to add the Python 2.7 environment

10. The **MSL-LoadLib** project is now shown in the **Project** window and you can begin to modify the code.

### 1.6.2 Commit changes to a repository

The following is only a very basic example of how to upload changes to the source code to the **MSL-LoadLib repository** by using PyCharm. See this link for a much more detailed overview on how to use git.

**Note:** This section assumes that you followed the instructions from *Install and set up Python, Git and PyCharm.*

1. Make sure that the git **Branch** you are working on is up to date by performing a **Pull**.
   
   a) Click on the blue, downward-arrow button in the top-right corner to update the project
b) Select the options for how you want to update the project (the default options are usually okay) and click **OK**

![Update Project dialog]

2. Make changes to the code.

3. When you are happy with the changes that you have made you should **Push** the changes to the MSL-LoadLib repository.
   a) Click on the green, check-mark button in the top-right corner to commit the changes
b) Select the file(s) that you want to upload to the MSL-LoadLib repository, add a useful message for the commit and then select **Commit and Push**.
c) Finally, push the changes to the MSL-LoadLib repository.

1.6.3 Use the setup.py and condatests.py scripts

MSL packages come with two scripts to help make development easier: setup.py and condatests.py. See the “create” ReadMe page for an overview on how to use these scripts.

1.6.4 Edit source code using the style guide

Please adhere to the following style guides when contributing to MSL packages. With multiple people contributing to the code base it will be easier to understand if there is a coherent structure to how the code is written:

**Note:** This section assumes that you followed the instructions from *Install and set up Python, Git and PyCharm*.

- Follow the PEP 8 style guide when possible (by default, PyCharm will notify you if you do not).
- Docstrings must be provided for all public classes, methods and functions.
- For the docstrings use the NumPy Style format.
  - Press CTRL+ALT+S to open the Settings window and navigate to Tools → Python Integrated Tools to select the NumPy docstring format and then click Apply then OK.
Do not use `print()` statements to notify the end-user of the status of a program. Use `logging` instead. This has the advantage that you can use different `logging levels` to decide what message types are displayed and which are filtered and you can also easily redirect all messages, for example, to a GUI widget or to a file. The `django project` has a nice overview on how to use Python’s builtin logging module.

### 1.7 License

MIT License

Copyright (c) 2017 - 2021, Measurement Standards Laboratory of New Zealand

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.8 Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>

1.9 Release Notes

1.9.1 Version 2.4.1.dev0

- Added
  - support for Python 3.9
- Changed
  - no longer use the XMLRPC API to get the information about the MSL packages that are available on PyPI

1.9.2 Version 2.4.0 (2020.04.20)

- Added
  - the `pip_options` kwarg to the `install`, `update` and `uninstall` functions
  - support for Python 3.8
  - can now create a new package that is not part of a namespace
  - `authorise` as an alias for `authorize` for the CLI
  - the `--create`, `--requires` and `--ini` arguments to `condatests.py`
- Changed
  - make the order of the log messages consistent: pypi -> github -> local
  - use a personal access token instead of a password for authentication to the GitHub API (authenticating to the GitHub API using a password is deprecated)
  - omit the `examples` directory from the coverage report and from pytest
- Fixed
call getpass.getuser() if git is installed but the username parameter has not been defined

do not split the text in the Description field to the next line in the middle of a word for the info() function

can now run condatests.py from any conda environment not just the base environment

check if an MSL package was installed via pip in editable mode

issue #6 - add support for specifying a version number when installing/updating

issue #5 - add support for specifying an extras_require value when installing/updating

issue #4 - error updating a package if the installed name != repository name

the tests_require list in setup.py now specifies zipp<2.0, pyparsing<3.0 and pytest<5.0 for Python 2.7

• Removed

  support for Python 3.4

1.9.3 Version 2.3.0 (2019.06.10)

• Added

  ability to install, update, create and uninstall MSL packages that do not start with msl-

  the shorter -D flag for --disable-mslpm-version-check

  use of a shell-style wildcard when specifying the package name(s)

  authorize as an API function

• Changed

  renamed the optional --path argument to --dir in the create command

  renamed the path kwarg to directory in the create method

  renamed the -uc flag to -u for the --update-cache flag

• Fixed

  running the list command did not align the Description text if the text continued on the next line

  removed the --quiet flag in the pip search msl- query

  removed the --process-dependency-links flag when installing packages (for compatibility with pip v19.0)

1.9.4 Version 2.2.0 (2019.01.06)

• Added

  the --doctest-glob='*.rst' and doctest_optionflags = NORMALIZE_WHITESPACE options to the setup.cfg file that is generated when a new package is created
- a --disable-mslpm-version-check flag
- a -uc alias for --upgrade-cache

**Changed**

- renamed test_envs.py to condatests.py and made it compatible with an optional condatests.ini file
- disable pip from checking for version updates by using the --disable-pip-version-check flag
- rename the --detailed flag to be --json
- moved the GitHub authorization file to the .msl directory and renamed the file to be .mslpm-github-auth

**Fixed**

- improved error handling if there is no internet connection
- use threading.Thread instead of multiprocessing.pool.ThreadPool when fetching info from GitHub since using ThreadPool would cause some Python versions to hang (see https://bugs.python.org/issue34172)
- colorama was not resetting properly

### 1.9.5 Version 2.1.0 (2018.08.24)

**Added**

- autodoc_default_options to conf.py for Sphinx 1.8 support
- nitpicky to conf.py
- the version_info named tuple now includes a releaselevel
- can now update the MSL Package Manager using msl update package-manager

**Removed**

- support for Python 3.7

**1.9.6 Version 2.0.0 (2018.07.02)**

**Added**

- ability to make authorized requests to the GitHub API (created authorize command)
- create a 3x additive --quiet flag (for silencing WARNING, ERROR and CRITICAL logging levels)
- show a message if the current version of the MSL Package Manager is not the latest release
- .pytest_cache/ and junk/ directories are now in .gitignore

**Changed**
– use `pkg_resources.working_set` instead of `pip.get_installed_distributions` to get the information about the MSL packages that are installed
– use logging instead of print statements
– the function signature for `install, uninstall, update and create`
– rename function `print_packages()` to `info()`
– rename module `helper.py` to `utils.py`
– show the detailed info about the GitHub repos in JSON format
– many changes to the documentation

• Fixed
  – `ApiDocs` in `setup.py` failed to run with Sphinx >1.7.0
  – bug if the GitHub repo does not contain text in the Description field
  – searching PyPI packages showed results that contained the letters `msl` but did not start with `msl`

• Removed
  – the constants `IS_PYTHON2`, `IS_PYTHON3` and `PKG_NAME`

1.9.7 Version 1.5.1 (2018.02.23)

• Fixed
  – the `setup.py` file is now compatible with Sphinx 1.7.0

1.9.8 Version 1.5.0 (2018.02.15)

• Added
  – the default install/update URI is PyPI (and uses the GitHub URI if the package does not exist on PyPI)
  – `--update-pypi-cache` and `--pypi` flags for the CLI

• Changed
  – default “yes/no” choice for the CLI was changed to be “yes”
  – `test_envs.py` has been updated to properly color the output text from pytest (v3.3.1) using colorama

1.9.9 Version 1.4.1 (2017.10.19)

• Added
  – `pip` as a dependency
• Changed
  – modified the template that is used for creating a new package:
    * the setup.py file is now self-contained, i.e., it no longer depends on other files to be available
    * removed requirements.txt and requirements-dev.txt so that one must specify the dependencies in install_requires
    * added the ApiDocs and BuildDocs classes from docs/docs_commands.py and removed docs/docs_commands.py
  – print the help message if no command-line argument was passed in
  – updated the documentation and the docstrings

1.9.10 Version 1.4.0 (2017.09.19)

• Added
  – add a --branch and --tag argument for the install and update commands
  – add a --path and --yes argument for the create command
  – added more functions to the helper module for the API:
    * check_msl_prefix
    * create_install_list
    * create_uninstall_list
    * get_zip_name
    * print_error
    * print_info
    * print_warning
    * print_install_uninstall_message
    * sort_packages

• Changed
  – the print_list function was renamed to print_packages
  – updated the documentation and the docstrings

1.9.11 Version 1.3.0 (2017.08.31)

• Added
  – use a thread pool to request the version number of a release for MSL repositories on GitHub
  – cache the package information about the GitHub repositories
  – add an --update-github-cache flag for the CLI
  – update documentation and docstrings
MSL-Package-Manager Documentation, Release 2.4.1.dev0

• Fixed
  – the msl namespace got destroyed after uninstalling a package in Python 2.7
  – running python setup.py test now sets install_requires = []
  – the test_envs.py file would hang if it had to “install eggs”
• Removed
  – the --release-info flag for the CLI is no longer supported

1.9.12 Version 1.2.0 (2017.08.10)

• add the --all flag for the CLI
• include --process-dependency-links argument for pip install
• create upgrade alias for update
• bug fixes and edits for the print messages

1.9.13 Version 1.1.0 (2017.05.09)

• update email address to “measurement”
• previous release date (in CHANGES.rst) was yyyy.dd.mm should have been yyyy.mm.dd
• previous release should have incremented the minor number (new update feature)

1.9.14 Version 1.0.3 (2017.05.09)

• add update command
• run pip commands using sys.executable

1.9.15 Version 1.0.2 (2017.03.27)

• split requirements.txt using \n instead of by any white space
• remove unnecessary “import time”

1.9.16 Version 1.0.1 (2017.03.03)

• show help message if no package name was specified for “create” command
• remove unused ‘timeout’ argument from test_envs.py
• reorganize if-statement in “list” command to display “Invalid request” when appropriate
1.9.17 Version 1.0.0 (2017.03.02)

- separate \texttt{install}, \texttt{uninstall}, \texttt{create} and \texttt{list} functions into different modules
- fix MSL namespace
- edit test\_envs.py to work with colorama and update stdout in real time
- add \texttt{--yes} and \texttt{--release\_info} flags for CLI
- create documentation and unit tests
- many bug fixes

1.9.18 Version 0.1.0 (2017.02.19)

- initial release
CHAPTER 2

Indices and tables

- genindex
- modindex
m
msl.package_manager, 11
msl.package_manager.authorize, 11
msl.package_manager.cli, 12
msl.package_manager.cli_argparse, 12
msl.package_manager.cli_authorize, 13
msl.package_manager.cli_create, 13
msl.package_manager.cli_install, 13
msl.package_manager.cli_list, 13
msl.package_manager.cli_uninstall, 14
msl.package_manager.cli_update, 14
msl.package_manager.create, 14
msl.package_manager.install, 15
msl.package_manager.uninstall, 15
msl.package_manager.update, 16
msl.package_manager.utils, 17
A
add_argument_all() (in module msl.package_manager.cli_argparse), 12
add_argument_branch() (in module msl.package_manager.cli_argparse), 12
add_argument_disable_mslpm_version_check() (in module msl.package_manager.cli_argparse), 12
add_argument_package_names() (in module msl.package_manager.cli_argparse), 12
add_argument_quiet() (in module msl.package_manager.cli_argparse), 12
add_argument_tag() (in module msl.package_manager.cli_argparse), 13
add_argument_update_cache() (in module msl.package_manager.cli_argparse), 13
add_argument_yes() (in module msl.package_manager.cli_argparse), 13
add_parser_authorize() (in module msl.package_manager.cli_authorize), 13
add_parser_create() (in module msl.package_manager.cli_create), 13
add_parser_install() (in module msl.package_manager.cli_install), 13
add_parser_list() (in module msl.package_manager.cli_list), 13
add_parser_uninstall() (in module msl.package_manager.cli_uninstall), 14
add_parser_update() (in module msl.package_manager.cli_update), 14

ArgumentParser (class in msl.package_manager.cli_argparse), 12
authorize() (in module msl.package_manager.authorize), 11

C
configure_parser() (in module msl.package_manager.cli), 12
contains_package_names() (msl.package_manager.cli_argparse.ArgumentParser method), 12
create() (in module msl.package_manager.create), 14

E
execute() (in module msl.package_manager.cli_authorize), 13
execute() (in module msl.package_manager.cli_create), 13
execute() (in module msl.package_manager.cli_install), 13
execute() (in module msl.package_manager.cli_list), 13
execute() (in module msl.package_manager.cli_uninstall), 14
execute() (in module msl.package_manager.cli_update), 14

G
get_command_name() (msl.package_manager.cli_argparse.ArgumentParser method), 12
get_email() (in module msl.package_manager.utils), 17
get_username() (in module msl.package_manager.utils), 17
github() (in module msl.package_manager.utils), 17
info() (in module msl.package_manager.utils), 17
install() (in module msl.package_manager.install), 15
installed() (in module msl.package_manager.utils), 18

main() (in module msl.package_manager.cli), 12
msl.package_manager (module), 11
msl.package_manager.authorize (module), 11
msl.package_manager.cli (module), 12
msl.package_manager.cli_argparse (module), 12
msl.package_manager.cliAuthorize (module), 13
msl.package_manager.cli_create (module), 13
msl.package_manager.cli_install (module), 13
msl.package_manager.cli_list (module), 13
msl.package_manager.cli_uninstall (module), 14
msl.package_manager.cli_update (module), 14
msl.package_manager.create (module), 14
msl.package_manager.install (module), 15
msl.package_manager.uninstall (module), 15
msl.package_manager.update (module), 16
msl.package_manager.utils (module), 17

parse_args() (in module msl.package_manager.cli), 12
pypi() (in module msl.package_manager.utils), 18

Python Enhancement Proposals

PEP 8, 33
PEP 8

set_log_level() (in module msl.package_manager.utils), 18

uninstall() (in module msl.package_manager.uninstall), 15
update() (in module msl.package_manager.update), 16

version_info (in module msl.package_manager), 11

Index