utilia Documentation

Release 1.0.0dev201305191746

Eric A. McDonald

May 19, 2013

CONTENTS

CHAPTER

ONE

INTRODUCTION

Welcome!

The *utilia* software development project produces a software distribution of the same name and provides infrastructure for this distribution. The *utilia* software distribution consists of two parts, a library of Python modules and a collection of scripts.

CHAPTER

TWO

PROJECT RESOURCES

2.1 Documentation

The official documentation for the project can be found on Read the Docs. There is no wiki at this time.

2.2 Issue Trackers

Issues (bug reports and feature requests) are managed with the GitHub issue trackers for the project.

2.3 Forum and Mailing Lists

A Google Groups forum is provided for discussion. This forum can also be used as a traditional mailing list. The email address of the mailing list is utilia@googlegroups.com.

There is no separate mailing list for announcements at this time.

2.4 IRC and Twitter

There is neither an IRC channel nor a Twitter account at this time.

2.5 Source Code and Supporting Works

The official source code and supporting works (documentation, graphical images, etc...) for the software are found on GitHub.

The source code is released under the Apache License 2.0 and the supporting works are covered by the CC BY 3.0 License. For more information about the project's licensing, please read the licensing text.

For help with acquiring the software, please read the Installation Guide.

2.6 Project Status

The software undergoes continuous integration (CI) testing via Travis CI. The current results of the CI tests are tabulated below.

Repository Branch	Result
Main Development (`master`)	

Warning: This project is still in its nascent stages. An official release has not yet been made.

2.7 Indices and Search Facility

- *search*
- *genindex*
- *modindex*

CHAPTER

THREE

TABLE OF CONTENTS

3.1 Installation Guide

As the software released by the project is primarily based off of Python, it probably should go without saying that you need to have a working Python installation. The following instructions assume that you do.

3.1.1 Releases

Currently no releases have been made available.

Todo

Mention listing on PyPI.

Todo

Give examples of easy_install and pip. For both site-wide and user-specific installations.

3.1.2 Tracking Development

To follow the latest development on the project, you will need a Git client if you do not already have one.

General Git Clients: Linux, BSD, Solaris

You may have already have a Git client installed, if you are using a Linux or BSD-like OS distribution. If you need to install one and can acquire superuser privileges, then here are instructions for some popular distributions:

http://www.git-scm.com/download/linux

or you can build from sources and install into whatever location for which you have write permissions:

https://github.com/git/git/tags

(You will probably want to download the latest tagged release which does not have *rc* in its name.)

If you are running on another Unix-like OS, then building from sources may be your only option.

General Git Clients: MacOS X

If you are on MacOS X, then you can get a package to install from here:

http://www.git-scm.com/download/mac

(If you do not have the necessary privileges to install the package, you can also attempt to build from sources. Please the see the instructions for Linux and the other Unix-like systems above.)

If you are using MacPorts, then you can also install Git using the installer for that distribution.

General Git Clients: Windows

If you are on Windows, then you can get an installer from here:

http://www.git-scm.com/download/win

or from here:

http://msysgit.github.com

If you are a Windows user, familiar with Linux or another Unix-like operating system, then *msysGit* is recommended.

There are also a number of GUI Git clients for Windows, such as:

http://code.google.com/p/gitextensions/

http://code.google.com/p/tortoisegit/

GitHub GUI Clients: MacOS X and Windows

The master software repository is hosted on GitHub. Several special Git clients exist for working with repositories on GitHub.

If you are on MacOS X, then you may wish to try:

http://mac.github.com/

If you are on Windows, then you may wish to try:

http://windows.github.com/

Working with Command-Line Git Clients

Many of the Git clients listed above are operated from a command shell.

Users of Unix-like operating systems probably need no instruction on how to access a command line.

If you are on MacOS X, then you may need the `Terminal` application located in the `Utilities` folder inside the `Applications` folder, shown in a Finder window.

If you are Windows, then you may have shortcut to start a command console; this shortcut is likely located in a folder created by your Git client's installer program under the `All Programs` folder off of the Start Menu. If you are using *msysGit*, then you can also right-click in a folder displayed in an Explorer window to get a context menu with some Git-related options, one of which should be `Git Bash`.

Once you have a command shell in front of you and you have navigated into whatever directory you wish, then you should be able to use the following command to clone the official software repository off of GitHub:

```
git clone http://github.com/utilia/utilia.git
```

From time-to-time, you may wish to update to the latest sources in the directory where your clone of the official repository resides. You can use the following command to accomplish this:

```
git pull
```

Todo

Talk about tracking branches.

Todo

Add repo tracking instructions for popular GUI clients.

Todo

Detail procedure for building versions off of GitHub.

Todo

Talk about virtual environments.

3.2 `utilia` Package

3.2.1 Overview

All of the modules in the library, provided by this project, are contained in the `utilia` package. The modules have a wide variety of purposes, but are united by a common theme, which has the following features:

- Compatibility across multiple implementations and versions of Python. This compatibility is maintained **without** the use of automatic code translators, such as `2to3`. Presently, compatibility with the following implementations and versions of Python is maintained:
  - CPython (http://www.python.org), versions 2.6 and 2.7 and all versions from the 3.x series
- Well-documented programming interfaces, including examples.
- Useful, general purpose functionality, not provided by the Python standard library or available as robust, standalone implementations by third parties.
- Seamless interoperability amongst components, when such interoperability is desirable and meaningful.

3.2.2 Package Description

Provides a wide assortment of useful subpackages. The subpackages cover these areas:

- `compatibility between Python implementations`
- `configuration parsing`
- `exceptions`
- `file systems`
- `functional programming`

- operating system interfaces
- types

3.2.3 Subpackages and Modules

compat Subpackage

Subpackage Description

Provides a compatibility layer among the different implementations and versions of Python. This is achieved by providing access to modules, classes, and functions via a uniform naming convention.

The following modules are available:

- A command-line argument parser module, which provides a uniform implementation across Pythons, regardless of whether they have one in their standard library.
- A built-ins module, which shadows the appropriate Python one and provides missing pieces.
- A shadow of the Python collections module, which provides missing pieces.
- A configuration file parser module, which shadows the appropriate Python one.

General Compatibility Functions

utilia.compat.**iter_dict_keys**(*the_dict*)
Returns an iterator which generates keys from the given dictionary.

In Python 2, attempts a call to the iterkeys method of an object.

Parameters **the_dict** – an object conforming to the dictionary interface

Return type a Python 2 dictionary iterator

In Python 3, attempts a call to the keys method of an object and wraps the resulting view in an iterator.

Parameters **the_dict** – an object conforming to the dictionary interface

Return type a Python 3 dictionary iterator

utilia.compat.**iter_dict_values**(*the_dict*)
Returns an iterator which generates values from the given dictionary.

In Python 2, attempts a call to the itervalues method of an object.

Parameters **the_dict** – an object conforming to the dictionary interface

Return type a Python 2 dictionary iterator

In Python 3, attempts a call to the values method of an object and wraps the resulting view in an iterator.

Parameters **the_dict** – an object conforming to the dictionary interface

Return type a Python 3 dictionary iterator

utilia.compat.**iter_dict_items**(*the_dict*)
Returns an iterator which generates key-value pairs from the given dictionary.

In Python 2, attempts a call to the iteritems method of an object.

Parameters **the_dict** – an object conforming to the dictionary interface

Return type a Python 2 dictionary iterator

In Python 3, attempts a call to the `items` method of an object and wraps the resulting view in an iterator.

Parameters **the_dict** – an object conforming to the `dictionary` interface

Return type a Python 3 dictionary iterator

General Compatibility Classes

class `utilia.compat.`**`AbstractBase_BASE`**

Generalized abstract class, which has `abc.ABCMeta` as its metaclass.

Subclass this in any implementation of Python supported by the library to create abstract methods and properties and use the `__subclasshook__` override and the subclass registration machinery from `abc.ABCMeta`.

Subpackages and Modules

`argparse` Module

Module Description Provides the `argparse` module, as it is missing from the standard library of some Python implementations.

Note: The source code for this module is published independently by a third party. This code or a near variant of it was included into Python 2.7 and Python 3.2 per acceptance of **PEP 389**.

`builtins` Module

Module Description Imports the contents of the relevant `built-ins` module for the current Python implementation and provides some things which may be missing from it.

For Python 3, these augmentations are performed for backward-compatibility:

- `StandardError` is aliased to `Exception`.
- `reduce` is imported from `functools`.
- `xrange` is aliased to `range`.

`collections` Module

Module Description Imports the contents of the `collections` module from the Python standard library and provides pieces which are missing in some Python implementations.

For Python 2.6 and 3.0, these missing classes are provided:

- `OrderedDict`

Note: The source code for the `OrderedDict` class comes from a recipe provided by a third party. This code or a near variant of it was included into Python 2.7 and Python 3.1 per acceptance of **PEP 372**.

`configparser` Module

Module Description Imports the contents of the relevant `configuration file parser` module from the Python standard library.

`config_parsers` Subpackage

Subpackage Description

Provides parsers for command-lines and configuration files.

Subpackages and Modules

`exceptions` Subpackage

Subackage Description

Provides fundamental exceptions.

Abstract Signatures

class `utilia.exceptions.`**`Exception_BASE`**
Abstract base class for all `utilia` exceptions.

Use this for your exception handler signature if you wish to catch any exception raised from within `utilia`.

class `utilia.exceptions.`**`Error_BASE`**
Abstract base class for all `utilia` exceptions which are regarded as errors.

Use this for your exception handler signature if you wish to catch any error condition raised from within `utilia`.

Fundamental Classes

class `utilia.exceptions.`**`Exception_WithReason`**(*reason_format*, **reason_args*)
Mix-in class for all `utilia` exceptions which carry a translatable format string and a tuple of arguments for substitution into the format string.

Inherits from `object`.

Use this for your exception handler signature if you wish to catch any exception, which has a translatable reason string, raised from within `utilia`.

`reason_args`
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into `reason_format`.

`reason_format`
Format string, containing zero or more `format`-style substitution tokens, for presentation as the reason for the exception. The substitions come from `reason_args`.

`translated`(*translator*)
Returns a translated string expressing the reason for the exception.

class utilia.exceptions.**Exception_Exiting**(*reason_format*, **reason_args*)
Mix-in class for all utilia exceptions which carry a translatable format string, a tuple of arguments for substitution into the format string, and a return code that could be supplied to a SystemExit exception.

Inherits from Exception_WithReason.

Use this for your exception handler signature if you wish to catch any exception, which has a return code and a described reason, raised from within utilia.

rc
The return code.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

General Exceptions

class utilia.exceptions.**InvalidKeyError**(*reason_format*, **reason_args*)
Exception class representing the error condition where a key of a particular name is not permissible. (Note that this is different than the error condition where a key is expected but missing.)

Inherits from Exception_Exiting and KeyError.

rc
The return code.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

class utilia.exceptions.**UnknownKeyError**(*reason_format*, **reason_args*)
Exception class representing the error condition where a key of a particular name is expected but missing. (Note that this is different than the error condition where a key is forbidden.)

Inherits from Exception_Exiting and KeyError.

rc
The return code.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

class utilia.exceptions.**InvalidValueError**(*reason_format*, **reason_args*)
Exception class representing the error condition where a value is considered invalid in a particular context.

Inherits from Exception_Exiting and ValueError.

rc
The return code.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

class utilia.exceptions.**InvokedAbstractMethodError**(*reason_format*, **reason_args*)
Exception class representing the error condition where an abstract method should have not been invoked in a superclass.

Inherits from Exception_Exiting and RuntimeError.

rc
The return code.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

Subpackages and Modules

filesystem Subpackage

Subpackage Description

Provides various utilities pertaining to file systems. These modules provide uniform interfaces for querying information about and manipulating various aspects of file systems, regardless of file system type or operating system.

The following modules provide calculated paths:

- stdpath

Exception Classes

class utilia.filesystem.**Exception_BASE**
Base class for all utilia filesystem exceptions.

Use this for your exception handler signature if you wish to catch any filesystem-related exception raised from within `utilia`.

class `utilia.filesystem.`**`Error_BASE`**
Base class for all `utilia` filesystem exceptions which are regarded as errors.

Use this for your exception handler signature if you wish to catch any filesystem-related error condition raised from within `utilia`.

Subpackages and Modules

`stdpath` Module

Module Description Provides functionality for calculating paths which are compliant with the standard filesystem layout of a particular OS platform. Significant effort is made to comply with published filesystem standards as well as Python conventions. Currently, the following OS platforms are supported:

- Linux [1], [2]
- MacOS X [3], [4]
- Windows [5], [6], [7]

The functions contained in this module follow a regular naming convention, which provides hints as to their purpose. This convention can be expressed as follows.

- Functions, which return paths where the core OS files are typically located, have the word **oscore** in their names.
- Functions, which return paths where the OS distribution files are typically located, have the word **osdist** in their names.
- Functions, which return paths associated with the default locations where a superuser or systems adminstrator would install software not packaged as part of the OS distribution, have the word **common** in their names.
- The suffix **install_root** denotes that a returned path refers to a top-level directory under which other directories for things, such as configuration information and package resources, can be found.
- The suffix **pythonic** denotes that a returned path is calculated in accordance with **PEP 370** and may rely upon the `site` module.
- The suffix **base** denotes that a returned path refers to an upper-level directory of a certain flavor, such as for configuration information or package resources, which is potentially common to many pieces of software and not tied to a particular one.
- Functions, which return paths associated with the location of a particular piece of software, have the word **my** in their names.
- Functions, which return paths relative to the current user's home directory, have the word **user** in their names.

Most of the functions can be instructed to operate in a *return None on failure* mode or a *raise exception on failure* mode. By default, these functions return `None` on failure to determine a path. However, all functions raise a `UnsupportedFilesystemLayout` exception if they lack the logic necessary to support the filesystem layout of a particular OS platform.

[1] Linux Filesystem Hierarchy Standard 2.3
[2] XDG Base Directory Specification 0.8
[3] Mac OS X and iOS File System Basics
[4] BSD Filesystem Hierarchy for Darwin
[5] Windows XP Command Shell Overview
[6] Windows CSIDL Values
[7] Windows KNOWNFOLDERID Reference

Here is a list of higher level functions, which users of this module will most likely be interested in:

- whereis_my_user_config()
- whereis_my_user_resources()
- whereis_my_common_config()
- whereis_my_common_resources()
- whereis_my_temp()
- whereis_my_saved_data()

Please see their documentation and the *Examples* section for details on using them.

Exception Classes

class utilia.filesystem.stdpath.**UnsupportedFilesystemLayout**(*reason_format*, **reason_args*)

Error if the standard filesystem layout associated with the current OS is unknown or unsupported.

Inherits from utilia.filesystem.Error_BASE and utilia.Exception_WithReason.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

class utilia.filesystem.stdpath.**UndeterminedFilesystemPath**(*reason_format*, **reason_args*)

Error if unable to ascertain a reasonably standard path for something.

Inherits from utilia.filesystem.Error_BASE and utilia.Exception_WithReason.

reason_args
Arguments to be substituted into the format string for presentation as the reason for the exception. The arguments are substituted into reason_format.

reason_format
Format string, containing zero or more format-style substitution tokens, for presentation as the reason for the exception. The substitions come from reason_args.

translated(*translator*)
Returns a translated string expressing the reason for the exception.

Elementary Functions

utilia.filesystem.stdpath.**which_fs_layout**()

Returns a classification of the expected filesystem layout according to the OS in use, if there is a classifier for that OS.

The possible filesystem layout classifications are as follows:

Classification	Operating Systems
POSIX	Linux
MacOS X	Darwin
Windows	Windows

Return type string

Raises UnsupportedFilesystemLayout, if there is no classifier implemented for the OS in use.

utilia.filesystem.stdpath.**concatenated_software_path_fragment**(*software_name*, *vendor_name=None*, *version=None*, *error_on_none=False*)

Returns a concatenation of the name of a software product with an optional name of the software product's vendor and an optional version string for the software product as a filesystem path fragment typical for the operating system architecture.

Parameters

- **software_name** (string) – Consider the name of this software product when constructing a path fragment which identifies it.
- **vendor_name** (string) – Consider the name of this software vendor when constructing a path fragment which identifies a software product.
- **version** (string) – Consider this version string when constructing a path fragment which identifies a software product.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

Public Base Paths

utilia.filesystem.stdpath.**whereis_oscore_install_root**(*error_on_none=False*)

Returns the path to the installation root for the core OS components, if it can be determined. Returns None, otherwise.

Environment variables or API calls may help determine this path on certain operating systems. In other cases, this path is fixed.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/
MacOS X	/
Windows	C:\Windows\System32

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_osdist_install_root**(*error_on_none=False*)

Returns the path to the installation root of the OS distribution, if it can be determined. Returns None, otherwise.

Environment variables or API calls may help determine this path on certain operating systems. In other cases, this path is fixed.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/usr
MacOS X	/System
Windows	C:\Windows

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_common_install_root**(*error_on_none=False*)

Returns the path to the typical default root for a shared or sitewide software installation by the superuser or systems administrator, if it can be determined. Returns None, otherwise.

Environment variables or API calls may help determine this path on certain operating systems. In other cases, this path is fixed.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/usr/local
MacOS	/Library
Windows	C:\Program Files

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_common_temp_base**(*error_on_none=False*)

Returns the path to the temporary storage area available for use by everyone on the system.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/tmp
MacOS X	/tmp
Windows	

Parameters **error_on_none** (boolean) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- `UnsupportedFilesystemLayout`, if there is no path determination logic for the filesystem layout.
- `UndeterminedFilesystemPath`, if a path could not be determined and the `error_on_none` argument is `True`.

`utilia.filesystem.stdpath.`**`whereis_oscore_config_base`**(*error_on_none=False*)

Returns the path to the typical top-level directory under which the configuration information resides for the core OS components, if it can be determined. Returns `None`, otherwise.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/etc
MacOS X	/System/Preferences
Windows	C:\Windows\System32\Config

Parameters **error_on_none** (boolean) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- `UnsupportedFilesystemLayout`, if there is no path determination logic for the filesystem layout.
- `UndeterminedFilesystemPath`, if a path could not be determined and the `error_on_none` argument is `True`.

`utilia.filesystem.stdpath.`**`whereis_osdist_config_base`**(*error_on_none=False*)

Returns the path to the typical top-level directory under which the configuration information resides for the core OS components, if it can be determined. Returns `None`, otherwise.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/etc
MacOS X	/System/Preferences
Windows	

Parameters **error_on_none** (boolean) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_common_config_base**(*error_on_none=False*)

Returns the path to the typical top-level directory under which the configuration information resides for software installed by the superuser or systems administrator, if it can be determined. Returns None, otherwise.

Below is a table of typical paths by filesystem layout classification:

Classification	Path
POSIX	/usr/local/etc
MacOS X	/Library/Preferences
Windows	

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

User Base Paths

utilia.filesystem.stdpath.**whereis_user_home**(*error_on_none=False*)

Returns the path to the current user's home directory, if it can be determined. Returns None, otherwise.

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_user_temp_base**(*error_on_none=False*)

Returns the path to the current user's temporary storage area.

Parameters **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

Other Base Paths

utilia.filesystem.stdpath.**whereis_preferred_temp_base**(*prefer_common=False*, *error_on_none=False*)

Returns:

- the result from a call to whereis_common_temp_base(), if it is not None and the prefer_common argument is True;
- the result from a call to whereis_user_temp_base(), if it is not None;
- the result from a call to whereis_common_temp_base(), if it is not None;
- or None, if all else fails.

Parameters

- **prefer_common** (boolean) – If True, the common path, if it exists, will be preferred over the user path.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

Public Derived Paths

utilia.filesystem.stdpath.**whereis_my_common_config_at_base**(*base_path*, *specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared configuration information for the specific software is stored.

Parameters

- **base_path** (string) – Calculate path, using this path as the base.
- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_common_config**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared configuration information for the specific software is stored. (This path is relative to the current OS platform's standard shared location for configuration information.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.

- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_common_config_pythonic**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared configuration information for the specific software is stored. (This path is relative to the current Python's installation base directory.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_common_resources_at_base**(*base_path*, *specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared resources for the specific software are stored.

Parameters

- **base_path** (string) – Calculate path, using this path as the base.
- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_common_resources**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared resources for the specific software are stored. (This path is relative to the current OS platform's standard shared location for resources.)

Parameters

- **specific_path** (`string`) – Calculate path, using this path as the most specific part.
- **error_on_none** (`boolean`) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- `UnsupportedFilesystemLayout`, if there is no path determination logic for the filesystem layout.
- `UndeterminedFilesystemPath`, if a path could not be determined and the `error_on_none` argument is `True`.

utilia.filesystem.stdpath.**whereis_my_common_resources_pythonic**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the shared resources for the specific software are stored. (This path is relative to the current Python's installation base directory.)

Parameters

- **specific_path** (`string`) – Calculate path, using this path as the most specific part.
- **error_on_none** (`boolean`) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- `UnsupportedFilesystemLayout`, if there is no path determination logic for the filesystem layout.
- `UndeterminedFilesystemPath`, if a path could not be determined and the `error_on_none` argument is `True`.

User Derived Paths

utilia.filesystem.stdpath.**whereis_my_user_config_at_base**(*base_path*, *specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's configuration information for the specific software is stored.

Parameters

- **base_path** (`string`) – Calculate path, using this path as the base.
- **specific_path** (`string`) – Calculate path, using this path as the most specific part.
- **error_on_none** (`boolean`) – If `True`, cause an exception to be raised if the return value would be `None`.

Return type `string` or `None`

Raises

- `UnsupportedFilesystemLayout`, if there is no path determination logic for the filesystem layout.
- `UndeterminedFilesystemPath`, if a path could not be determined and the `error_on_none` argument is `True`.

utilia.filesystem.stdpath.**whereis_my_user_config**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's configuration information for the specific software is stored. (This path is relative to the current OS platform's standard per-user location for configuration information.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_user_config_pythonic**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's configuration information for the specific software are stored. (This path is relative to Python's user base directory, as specified by **PEP 370**.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_user_resources_at_base**(*base_path*, *specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's resources for the specific software are stored.

Parameters

- **base_path** (string) – Calculate path, using this path as the base.
- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.

- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_user_resources**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's resources for the specific software are stored. (This path is relative to the current OS platform's standard per-user location for resources.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_user_resources_pythonic**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory, where the current user's resources for the specific software are stored. (This path is relative to Python's user base directory, as specified by **PEP 370**.)

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

Other Derived Paths

utilia.filesystem.stdpath.**whereis_my_saved_data_at_base**(*base_path*, *specific_path=None*, *error_on_none=False*)

Returns a path to the directory where works, created by the current user, will be stored.

Parameters

- **base_path** (string) – Calculate path, using this path as the base.
- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_saved_data**(*specific_path=None*, *error_on_none=False*)

Returns a path to the directory where works, created by the user of the specific software, will be stored.

Uses whereis_user_home() internally.

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

utilia.filesystem.stdpath.**whereis_my_temp**(*specific_path=None*, *prefer_common=False*, *error_on_none=False*)

Returns the path to the preferred temporary storage for the specific software.

The path calculation relies on results from the whereis_preferred_temp_base() and the concatenated_software_path_fragment() functions.

Parameters

- **specific_path** (string) – Calculate path, using this path as the most specific part.
- **prefer_common** (boolean) – If True, the common path, if it exists, will be preferred over the user path.
- **error_on_none** (boolean) – If True, cause an exception to be raised if the return value would be None.

Return type string or None

Raises

- UnsupportedFilesystemLayout, if there is no path determination logic for the filesystem layout.
- UndeterminedFilesystemPath, if a path could not be determined and the error_on_none argument is True.

Examples

Todo

Create examples.

References

`functional` Subpackage

Subpackage Description

Provides assorted tools for functional programming. Included are:

- `logic functions`, which accept an arbitrary number of arguments.

Subpackages and Modules

`logic` Module

Module Description Provides common logic operators as functions.

The functions come in two forms: *objective* and *boolean*. Objective functions work in a manner similar to the `and` and `or` operators, built in to Python, in that return one of the original objects provided as an operand. Boolean functions return boolean values rather than the original objects. The names of objective functions start with `o_`.

The names of all functions in the module end with `f`, because some of the names would otherwise conflict with Python keywords. The `f` can be understood to mean *function* or *functional version* as opposed to an inline operator.

None of the functions in this module perform true *short-circuit* evaluation like the Python `and` and `or` operators do. This is because any expressions given as arguments to a function are evaluated before the function is called.

The documentation below and the *Examples* section provide more details.

Boolean Functions

`utilia.functional.logic.`**`andf`**`(*posargs)`

If no arguments are supplied, returns True. Else, returns the result of calling the `all` built-in function on the sequence of arguments.

Below is a truth table for this function with two arguments:

p	q	$p \wedge q$
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type `boolean`

`utilia.functional.logic.`**`nandf`**`(*posargs)`

If no arguments are supplied, returns False. Else, returns the negated result of calling the `all` built-in function on the sequence of arguments.

Below is a truth table for this function with two arguments:

p	q	$p \uparrow q$
`False`	`False`	`True`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`False`

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

utilia.functional.logic.**orf**(**posargs*)

If no arguments are supplied, returns False. Else, returns the result of calling the any built-in function on the sequence of arguments.

Below is a truth table for this function with two arguments:

p	q	$p \vee q$
False	False	False
False	True	True
True	False	True
True	True	True

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

utilia.functional.logic.**norf**(**posargs*)

If no arguments are supplied, returns True. Else, returns the negated result of calling the any built-in function on the sequence of arguments.

Below is a truth table for this function with two arguments:

p	q	$p \downarrow q$
False	False	True
False	True	False
True	False	False
True	True	False

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

utilia.functional.logic.**xorf**(**posargs*)

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the result of reducing the sequence of arguments with a logical xor function.

Below is a truth table for this function with two arguments:

p	q	pq
False	False	False
False	True	True
True	False	True
True	True	False

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

Raises TypeError, if a required argument is missing.

utilia.functional.logic.**xnorf**(**posargs*)

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the result of reducing the sequence of arguments with a logical xnor function.

Below is a truth table for this function with two arguments:

p	q	$p \Leftrightarrow q$
False	False	True
False	True	False
True	False	False
True	True	True

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

Raises TypeError, if a required argument is missing.

utilia.functional.logic.**impliesf**(**posargs*)

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the result of reducing the sequence of arguments with a logical implies function.

Below is a truth table for this function with two arguments:

p	q	$p \Rightarrow q$
False	False	True
False	True	True
True	False	False
True	True	True

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

Raises TypeError, if a required argument is missing.

utilia.functional.logic.**nimpliesf**(**posargs*)

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the negated result of reducing the sequence of arguments with a logical implies function.

Below is a truth table for this function with two arguments:

p	q	pq
False	False	False
False	True	False
True	False	True
True	True	False

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type boolean

Raises TypeError, if a required argument is missing.

utilia.functional.logic.**cimpliesf**(**posargs*)

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the result of reducing the sequence of arguments with a logical converse implies function.

Below is a truth table for this function with two arguments:

p	q	$p \Leftarrow q$
False	False	True
False	True	False
True	False	True
True	True	True

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type `boolean`

Raises `TypeError`, if a required argument is missing.

`utilia.functional.logic.`**`cnimpliesf`**`(*posargs)`

Converts sequence of arguments to booleans. If no arguments are supplied, raises an exception. Else, returns the negated result of reducing the sequence of arguments with a logical `converse implies` function.

Below is a truth table for this function with two arguments:

p	q	pq
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`False`
`True`	`True`	`False`

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type `boolean`

Raises `TypeError`, if a required argument is missing.

Objective Functions

`utilia.functional.logic.`**`o_andf`**`(*posargs)`

If no arguments are supplied, returns True. Else, returns the result of reducing the sequence of arguments with the logical `and` operator.

Below is a *truth-like* table for this function with two arguments:

p	q	$p \wedge q$
zeroish	zeroish	p
zeroish	non-zeroish	p
non-zeroish	zeroish	q
non-zeroish	non-zeroish	q

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type object of any type

`utilia.functional.logic.`**`o_orf`**`(*posargs)`

If no arguments are supplied, returns False. Else, returns the result of reducing the sequence of arguments with the logical `or` operator.

Below is a *truth-like* table for this function with two arguments:

p	q	$p \vee q$
zeroish	zeroish	q
zeroish	non-zeroish	q
non-zeroish	zeroish	p
non-zeroish	non-zeroish	p

Parameters **posargs** (*objects of any type*) – Arbitrary number of positional arguments.

Return type object of any type

Examples

Todo

Create examples.

`os` Subpackage

Subpackage Description

Provides convenience functions for working with various operating system facilities in a platform-independent manner.

Subpackages and Modules

`exit_codes` Module

Module Description Uniform list of exit codes with values that may vary according to any prevailing standards for a given platform [8], [9].

General Exit Codes

`utilia.os.exit_codes.`**`SUCCESS`**`()`
Successful completion of process.

`utilia.os.exit_codes.`**`FAILURE`**`()`
Unsuccessful completion of process without error.

(Example: POSIX **grep** returns 1 when no lines are matched.)

`utilia.os.exit_codes.`**`ERROR`**`()`
Exiting process because of a general error.

(Example: POSIX **grep** returns 2 on error.)

`utilia.os.exit_codes.`**`INTERNAL_SOFTWARE_ERROR`**`()`
Exiting process because the software failed an internal consistency check or assertion, encountered an unimplemented virtual method, etc....

References

`types` Subpackage

Subpackage Description

Provides various useful types, which either do not exist in the standard library of some of our supported versions of Python or have more features than their counterparts in the standard library of some of our supported versions of Python.

The following modules provide collections of types:

- `maps`

[8] BSD Exit Codes
[9] Python Exit Codes for Unix

Subpackages and Modules

`maps` Module

Module Description Provides a collection of various map types with a consistent interface across Python versions. The collection of map types consists of:

- `OrderedDict`

3.3 Scripts Collection

3.3.1 Overview

The collected scripts serve a wide variety of purposes, but have the following features in common:

- Written in Python with the same compatibility level as the accompanying modules library.
- Flexible execution in an OS-independent manner.
- Well-documented invocation arguments and options, operations, and exit codes. Usage examples are also included in the documentation.
- Useful, general purpose functionality.
- Seamless interoperability amongst scripts, when such interoperability is desirable and meaningful.

3.3.2 Table of Scripts

3.4 Development Guide

The target audience for this guide is people wishing to contribute source code to the project. However, users of the software may also be interested in the practices discussed in this guide.

3.4.1 Table of Contents

Source Code

Python

Table of Contents

Tuples The `tuple` type is somewhat controversial in Python. The following discussion is not intended to be an attack on this type, but will highlight some of the elements of the controversy.

Versus Lists From a strictly functional perspective, the major difference between a `tuple` and a `list` is that a `tuple` contains an immutable sequence whereas a `list` does not. Beyond this, some people in the Python community claim that the `list` type is intended for homogeneous collections whereas the `tuple` type is intended for heterogeneous collections with a fixed ordering. For example, see the discussions at:

http://third-bit.com/blog/archives/000450.html#comment-380

http://news.e-scribe.com/397

However, the Python language does not enforce the distinction between homogeneity and heterogeneity for the two types. Pretending that it is actually there seems a bit like wearing the emperor's new clothes. Furthermore, most examples cited to support the alleged fixed ordering property rely upon the immutability of the sequence to enforce this. But, as is often the case, the semantics of something is in the eye of the beholder.

Named Tuples In Python 2.6, the `namedtuple` type factory was introduced into the `collections` package of the standard library. Types, created with this factory, contain collections with a fixed ordering. Thus, the claims made about this property of the `tuple` type can be enforced in types out of the `namedtuple` factory. Therefore, if you need the fixed ordering property, it is recommended that you create types with this factory and use them. These types can be used in nearly any situation in which a plain `tuple` can be used.

Types, created with `namedtuple`, have some additional advantages over the `tuple` type:

- They are named, and their names are reflected in the strings returned by functions such as `repr` and `type`.
- They can be instantiated with positional or keyword arguments, where the keywords are the names of their sequence members.
- Sequence members of their instances can be accessed by name as well as by index.

Here are some examples of preferred and avoided usages:

```
from collections import namedtuple
Point = namedtuple( "Point", "x y" )
OneTuple = namedtuple( "OneTuple", "u" )

some_dict = { Point( 3, 4 ): "foo" }   # prefer
some_dict = { ( 3, 4 ): "foo" }        # avoid

p = Point( 5, 12 ); p.x**2 + p.y**2       # prefer
p = Point( 5, 12 ); p[ 0 ]**2 + p[ 1 ]**2 # avoid
p = ( 5, 12 ); p[ 0 ]**2 + p[ 1 ]**2      # avoid

OneTuple( 42 ) # sequence containing an integer
( 42 )         # uncontained integer
tuple( 42 )    # ERROR
```

Syntactic Sugar Avoid the syntactic sugar for tuples (parentheses) whenever possible, because of the following reasons:

- Parentheses are already used for expression grouping and invoking callables. Too many parentheses can make source code harder to read.
- The initialization of a 1-tuple cannot be disambiguated from a grouped expression, except with the inclusion of a trailing comma. Programming error can creep in when the size of a tuple initializer is reduced to one element

from a higher number of elements or increased to one element from no elements, as the trailing comma may be forgotten.

Here are some examples of preferred and avoided usages:

```
[ ]          # prefer
tuple( )     # prefer if sequence immutability is desired
( )          # avoid

[ 1 ]           # prefer
tuple( [ 1 ] ) # prefer if sequence immutability is desired
( 1, )          # avoid

[ 1, 2, 4 ]           # prefer
tuple( [ 1, 2, 4 ] ) # prefer if sequence immutability is desired
( 1, 2, 4 )           # avoid

return "a", 1, foo      # prefer
return ( "a", 1, foo )  # avoid

from utilia.compat import iter_dict_items
for key, val in iter_dict_items( some_dict )       # prefer
for ( key, val ) in iter_dict_items( some_dict )   # avoid
```

If you care about linguistic symmetry or code aesthetics, then consider the following contrasts:

```
[ 42 ]   # asymmetric with tuple, symmetric with set
( 42, )  # asymmetric with list and set
{ 42 }   # asymmetric with tuple, symmetric with list
# Note: Sugar for set is only available in Python 2.7 and 3.x.

tuple( [ 1, 2, 4 ] )      # symmetric with set
set( [ 1, 2, 4 ] )        # symmetric with tuple
```

Lists

Lists Of Tuples An `OrderedDict` can be used to accumulate key-value pairs in an order-preserving manner. These accumulated pairs can later be retrieved as tuples via a standard iteration method. This is cleaner than appending tuples to a list.

```
from utilia.compat import iter_dict_items
from utilia.compat.collections import OrderedDict

od = OrderedDict( )
od[ "foo" ] = 1
od[ "bar" ] = 2
# ...
od[ "baz" ] = 3

for key, value in iter_dict_items( od ):
   # Do stuff.
```

In cases where lists of tuples can be generated automatically, then the use of a tuple type, produced by the `namedtuple` factory, is preferred. If anonymity and mutability are acceptable, then using a list of lists is preferred.

```
from collections import namedtuple

# prefer: list of named tuples
```

```
Pair = namedtuple( "Pair", "x y" )
[ Pair( x, y ) for x in xrange( 10 ) for y in xrange( 10 ) ]
# prefer: list of lists
[ [ x, y ] for x in xrange( 10 ) for y in xrange( 10 ) ]
# avoid
[ tuple( [ x, y ] ) for x in xrange( 10 ) for y in xrange( 10 ) ]
# avoid
[ ( x, y ) for x in xrange( 10 ) for y in xrange( 10 ) ]
```

Sets

Syntactic Sugar As the present aim is to support Python 2.6 in addition to higher versions, we cannot use the syntactic sugar for the `set` type, which is available in Python 2.7 and 3.x. Therefore:

```
set( [ 1, 2, 4 ] )          # use
{ 1, 2, 4 }                 # do not use
```

Tuples The `tuple` type is somewhat controversial in Python. The following discussion is not intended to be an attack on this type, but will highlight some of the elements of the controversy.

Versus Lists From a strictly functional perspective, the major difference between a `tuple` and a `list` is that a `tuple` contains an immutable sequence whereas a `list` does not. Beyond this, some people in the Python community claim that the `list` type is intended for homogeneous collections whereas the `tuple` type is intended for heterogeneous collections with a fixed ordering. For example, see the discussions at:

http://third-bit.com/blog/archives/000450.html#comment-380

http://news.e-scribe.com/397

However, the Python language does not enforce the distinction between homogeneity and heterogeneity for the two types. Pretending that it is actually there seems a bit like wearing the emperor's new clothes. Furthermore, most examples cited to support the alleged fixed ordering property rely upon the immutability of the sequence to enforce this. But, as is often the case, the semantics of something is in the eye of the beholder.

Named Tuples In Python 2.6, the `namedtuple` type factory was introduced into the `collections` package of the standard library. Types, created with this factory, contain collections with a fixed ordering. Thus, the claims made about this property of the `tuple` type can be enforced in types out of the `namedtuple` factory. Therefore, if you need the fixed ordering property, it is recommended that you create types with this factory and use them. These types can be used in nearly any situation in which a plain `tuple` can be used.

Types, created with `namedtuple`, have some additional advantages over the `tuple` type:

- They are named, and their names are reflected in the strings returned by functions such as `repr` and `type`.
- They can be instantiated with positional or keyword arguments, where the keywords are the names of their sequence members.
- Sequence members of their instances can be accessed by name as well as by index.

Here are some examples of preferred and avoided usages:

```
from collections import namedtuple
Point = namedtuple( "Point", "x y" )
OneTuple = namedtuple( "OneTuple", "u" )
```

```
some_dict = { Point( 3, 4 ): "foo" }    # prefer
some_dict = { ( 3, 4 ): "foo" }         # avoid

p = Point( 5, 12 ); p.x**2 + p.y**2       # prefer
p = Point( 5, 12 ); p[ 0 ]**2 + p[ 1 ]**2 # avoid
p = ( 5, 12 ); p[ 0 ]**2 + p[ 1 ]**2      # avoid

OneTuple( 42 ) # sequence containing an integer
( 42 )         # uncontained integer
tuple( 42 )    # ERROR
```

Syntactic Sugar Avoid the syntactic sugar for tuples (parentheses) whenever possible, because of the following reasons:

- Parentheses are already used for expression grouping and invoking callables. Too many parentheses can make source code harder to read.
- The initialization of a 1-tuple cannot be disambiguated from a grouped expression, except with the inclusion of a trailing comma. Programming error can creep in when the size of a tuple initializer is reduced to one element from a higher number of elements or increased to one element from no elements, as the trailing comma may be forgotten.

Here are some examples of preferred and avoided usages:

```
[ ]         # prefer
tuple( )    # prefer if sequence immutability is desired
( )         # avoid

[ 1 ]           # prefer
tuple( [ 1 ] ) # prefer if sequence immutability is desired
( 1, )          # avoid

[ 1, 2, 4 ]           # prefer
tuple( [ 1, 2, 4 ] ) # prefer if sequence immutability is desired
( 1, 2, 4 )           # avoid

return "a", 1, foo      # prefer
return ( "a", 1, foo )  # avoid

from utilia.compat import iter_dict_items
for key, val in iter_dict_items( some_dict )       # prefer
for ( key, val ) in iter_dict_items( some_dict )   # avoid
```

If you care about linguistic symmetry or code aesthetics, then consider the following contrasts:

```
[ 42 ]   # asymmetric with tuple, symmetric with set
( 42, )  # asymmetric with list and set
{ 42 }   # asymmetric with tuple, symmetric with list
# Note: Sugar for set is only available in Python 2.7 and 3.x.

tuple( [ 1, 2, 4 ] )      # symmetric with set
set( [ 1, 2, 4 ] )        # symmetric with tuple
```

Lists

Lists Of Tuples An `OrderedDict` can be used to accumulate key-value pairs in an order-preserving manner. These accumulated pairs can later be retrieved as tuples via a standard iteration method. This is cleaner than appending tuples to a list.

```
from utilia.compat import iter_dict_items
from utilia.compat.collections import OrderedDict

od = OrderedDict( )
od[ "foo" ] = 1
od[ "bar" ] = 2
# ...
od[ "baz" ] = 3

for key, value in iter_dict_items( od ):
   # Do stuff.
```

In cases where lists of tuples can be generated automatically, then the use of a tuple type, produced by the `namedtuple` factory, is preferred. If anonymity and mutability are acceptable, then using a list of lists is preferred.

```
from collections import namedtuple

# prefer: list of named tuples
Pair = namedtuple( "Pair", "x y" )
[ Pair( x, y ) for x in xrange( 10 ) for y in xrange( 10 ) ]
# prefer: list of lists
[ [ x, y ] for x in xrange( 10 ) for y in xrange( 10 ) ]
# avoid
[ tuple( [ x, y ] ) for x in xrange( 10 ) for y in xrange( 10 ) ]
# avoid
[ ( x, y ) for x in xrange( 10 ) for y in xrange( 10 ) ]
```

Sets

Syntactic Sugar As the present aim is to support Python 2.6 in addition to higher versions, we cannot use the syntactic sugar for the `set` type, which is available in Python 2.7 and 3.x. Therefore:

```
set( [ 1, 2, 4 ] )        # use
{ 1, 2, 4 }               # do not use
```

Todo

Create discussions for C and C++ extensions.

Ideas for Further Development

Modules Library

- Enhanced support for finding and utilizing localization resources, such as translations.
- Generic tester functions, which may be used with testing frameworks or for other purposes.
- Unified parsing of command-line and config file options.
- Extensions to the `distutils.ccompiler` module for proper support of standalone Mingw32 and Mingw64.

- An extension to the `logging` module, which allows for logging to GUI message boxes. (Useful for showing critical errors before intended GUI subsytem is initialized or when operating in a headless or minimized-to-tray mode, in some cases.) Would try toolkit for current window manager first, before attempting to fallback to others.
- Wrapper for the PDCurses library to provide a more-portable *curses* implementation with some interesting back-end options (e.g., SDL).
- Wrapper for the AAlib or libcaca library.

Scripts Collections

- Replacement, augmentation, deletion, or creation of text blocks within all files matching a pattern within a directory hierarchy. Allow for lines, matching a regular expression, to be preserved during replacement.
- Searching and reporting on inventories used by Sphinx's `intersphinx` extension.
- Exploration and searching of the PE-COFF format.

To-Do List

Todo

Convert into wiki content.

(The *original entry* is located in ../answers/index.rst, line 8.)

Todo

Create discussion of version control.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/develop/index.rst, line 27.)

Todo

Create discussion on documentation.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/develop/index.rst, line 30.)

Todo

Create discussion of testing.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/develop/index.rst, line 33.)

Todo

Create discussions for C and C++ extensions.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/develop/source-code/index.rst, line 16.)

Todo

Mention listing on PyPI.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 20.)

Todo

Give examples of easy_install and pip. For both site-wide and user-specific installations.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 23.)

Todo

Talk about tracking branches.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 136.)

Todo

Add repo tracking instructions for popular GUI clients.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 139.)

Todo

Detail procedure for building versions off of GitHub.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 142.)

Todo

Talk about virtual environments.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/install/index.rst, line 145.)

Todo

Create examples.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/modules/filesystem/stdpath.rst, line 108.)

Todo

Create examples.

(The *original entry* is located in /var/build/user_builds/utilia/checkouts/latest/doc/guides/modules/functional/logic.rst, line 52.)

Todo

Create discussion of version control.

Todo

Create discussion on documentation.

Todo

Create discussion of testing.

3.5 Legal Stuff

3.5.1 Copyrights and Licensing

This text pertains to the *utilia* project ("the Project"), which creates and maintains the *utilia* software ("the Software") and the documentation and other works which are distributed with it.

Copyrights

All contributions to the Project are subject to the relevant licenses; please read the licensing details below.

As of this writing, no assignment of copyrights is required for the acceptance of contributions to the Software or other works which are distributed with it.

Attribution of Authorship

Version control systems provide a means of tracking the authorship of contributions to a software distribution. Because of this and the copyright law extant in most jurisdictions throughout the world, the Project forbids copyright notices in anything contributed to it, with an exception being made for those contributions which originate from separate, published collections or distributions. Instead:

- the official version control repository for the Software and the other works distributed with it shall serve as the official registry for tracking the authorship of contributions;
- the Software shall be distributed with a file, named `CONTRIB.txt`, which shall be the official list of all contributors who wish to be acknowledged as such.

The location of the official version control repository for the Software and the other works distributed with it can be found at the following Internet URL:

> http://github.com/utilia/utilia

For a more detailed legal discussion behind this practice, please read the following white paper from the Software Freedom Law Center:

> http://softwarefreedom.org/resources/2012/ManagingCopyrightInformation.html

For anything, which is not in the public domain and which is contributed to the Project from a separate, published collection or distribution, a notice about authorship and licensing shall appear in a file, named `NOTICE.txt`, which shall be distributed with the Software.

License for the Software

The Software is licensed under the Apache License, version 2.0 (the "Software License"). The text of the Software License may be found in the `licenses` subdirectory of the `doc` directory, distributed with the Software, or at the following Internet URL:

> http://www.apache.org/licenses/LICENSE-2.0

The contents of any file, distributed under the `src` directory, excluding files named with a `.dat` suffix, are licensed under the Software License. The contents of any file, distributed under the `src` directory and named with a `.dat` suffix, is in the public domain. Portions of the Software, which reside outside of the `src` directory hierarchy, are also licensed under the Software License. For example, scripts or source code fragments, residing in a database table, are considered part of the Software, even though the database, containing the table, may be distributed outside of the `src` directory hierarchy.

Exemptions to any of the foregoing are noted in the `NOTICE.txt` file, which is distributed with the Software.

License for Documentation and Other Works

Any copyrightable work, which is not part of the Software proper but which is distributed with the Software, including but not limited to icons, figures, graphical images, and separate documentation, is licensed under the Creative Commons Attribution 3.0 Unported License (the "Supporting Works License"). The text of the Supporting Works License may be found at the following Internet URL:

> http://creativecommons.org/licenses/by/3.0/

The contents of any file, distributed under the `doc` directory, excluding license files, are licensed under the Supporting Works License. Translations of message catalogs to various languages are also licensed under the Supporting Works License. Icons, figures, and graphical images are also licensed under the Supporting Works License. Other copyrightable works, not explicitly mentioned in the foregoing statements, are also licensed under the Supporting Works License. Documentation, which coexists within the same container as source code for the Software, is under the Software License and not the Supporting Works License. Licenses are reproduced and distributed with the Software for your convenience of reference; any applicable copyright law and any terms and conditions set forth by the author(s) of a reproduced license, regarding that license, are considered to be in full effect for that license.

Exemptions to any of the foregoing are noted in the `NOTICE.txt` file, which is distributed with the Software.

Apache License 2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
```

"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable

(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct
or contributory patent infringement, then any patent licenses
granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.

```
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

3.5.2 Additional Legal Notices

This text pertains to the *utilia* project ("the Project"), which creates and maintains the *utilia* software ("the Software") and the documentation and other works which are distributed with it.

Note: The contributions to the Project listed below have been granted exceptions to the standard licensing and legal notice policies of the Project. To review the standard legal notice and licensing policies of the Project, please read the `LEGAL.txt` file, distributed with the Software.

Licensed Works

Copies of some licensed, separately-published works are incorporated into the Project. The licenses and copyright holders (when known) for these works are indicated below.

agogo (Sphinx theme)

Publication Info: https://bitbucket.org/birkenfeld/sphinx/src

Original Author: Andi Albrecht

Copyright Holder: the Sphinx team

License: BSD (2-clause)

Location: `doc/guides/_themes/agogo`

Notes:

- The `static/agogo.css_t` file was modified by the Project.
- Please see the `AUTHORS` file and the `LICENSE` file in the Sphinx software distribution for more information about the pertinent copyright and license.

argparse

Publication Info: http://code.google.com/p/argparse/

Original Author: Steven J. Bethard

License: Python 2.0

Location: `src/lib/utilia/config_parsers/argparse.py`

distribute_setup

Publication Info: http://python-distribute.org/distribute_setup.py

License: [Uncertain, being researched. Probably Python 2.0.]

Location: `distribute_setup.py`

OrderedDict

Publication Info: http://code.activestate.com/recipes/576669/

Original Author: Raymond Hettinger

License: MIT

Location: `src/lib/utilia/types/ordered_dict.py`

Works in the Public Domain

Copies of works incorporated from the public domain by the Project are not captured under the Project's standard licenses; instead, they are left in the public domain.

There are no known works incorporated from the public domain at this time.

3.6 Acknowledgements

This text pertains to the *utilia* project ("the Project"), which creates and maintains the *utilia* software ("the Software") and the documentation and other works which are distributed with it.

3.6.1 Contributors to the Project

Below is a list of contributors, given by name, nickname, or handle, and an optional PGP/GPG public key ID.

Name / Nickname / Handle	PGP/GPG Key ID
Eric A. McDonald	0xcf627346fb2ad110

Note: If your name, nickname, or handle is not listed above and you have contributed, then please email the maintainer(s) of the Software with details on your contribution(s) and how you would like to be listed.

3.7 Questions and Answers

Todo

Convert into wiki content.

3.7.1 Purpose

Why does this project exist?

(*This question is perhaps best answered in the first-person perspective of its creator.*)

I created this public project, as a hobby, mainly out of four motivations:

- A desire to explore various interesting pieces of infrastructure that have appeared in the free and open source ecosystem. At work, I often get straight-jacketed into using certain tools and not using other ones, due to extant realities. Breaking free of that grind and exploring what is available out there is a way to remind myself that I can still enjoy software development. It also allows me to learn new things at my leisure, rather than upon demand like I often do at work. And, it provides a functional means of attempting to "stay current" and relevant.
- A desire to create something new *my way*. At work, I am almost always working with source codes which have been predominantly written by other people. Although I usually don't have any trouble adapting to other people's designs and assorted conventions, it is still quite liberating to sometimes push their impositions aside and make something which bears my own "signature".
- A desire to have my various useful pieces of Python code collected in a central location, which would be easy for me to access from anywhere.
- Ease-of-mind regarding backups for my growing code library. (Yes, I know that version control is no substitute for a good backup, but it *does* provide an additional level of safety and it *does* provide an automatic, off-site backup.)

I don't really care if no one else ever uses any of the tools I am providing, even though I think it would be swell if they did. If nothing else, I will (and already do) use them. That said, there is a lurking hope in the back of my mind for a couple of nice outcomes:

- I'll meet and collaborate with interesting, intelligent, polite people. I'm somewhat ill-at-ease with many traditional social networking channels. But, having technical (and philosophical) discussions with others is something with which I'm quite comfortable; this project provides a venue for such discussions to occur.
- A potential employer from industry will take notice and offer me an option to "escape" academia. I work in academia as a software engineer. I know I'm underpaid and not doing anything on the cutting edge, but I like the fairly-relaxed environment it has to offer. Also, my particular combination of "dev" and "ops" experience, involving scientific computing, can be a hard sell outside of the academic research environment. By doing something more general purpose and, hopefully, demonstrating skill in the process, I hope to increase my options. (Probably posting my solutions to some of the Project Euler problems would help too - I might do that as a separate project.)

So, that's the long-winded story behind this project.

3.7.2 Names and Nomenclature

What does *utilia* mean?

The word *utilia* is Latin for *useful things*. It can be used as either an adjective or a noun in Latin. The noun is a neuter, nominative or accusative, plural from the third declension. It is shorter to say or type than *rebus utilibus*, which some might consider to be more proper.

PYTHON MODULE INDEX