shallot - Scripting Documentation

Release 1.2.4764

Josef Hahn

Jul 26, 2020

CONTENTS

Shallot has a plugin interface which allows to add functionality to the core from outside. This document is for people who are going to create a new Shallot plugin in order to bring some new functionality to Shallot.

A Shallot plugin developer should have at least basic skills in the Python programming language and should have read the entire section h_scriptingmanual (From the h_featureoverview, it might be okay to read just the parts relevant for you).

After the h_scriptingmanual, you find the complete interface reference. You should use it for finding some technical details (which classes exist, which methods are there, which arguments do they have, ...). It lists the structure of the interface from different perspectives. Some of them are quite useful, others ones are more technical and only interesting in rare situations. Good starting points are those sections:

- The "Class List" provides an overview of all existing classes.
- The "shallot Namespace Reference" gives a coarser overview of all existing subparts.

CHAPTER

ONE

GENERAL NOTES

This section covers some notes and hints, which you should know about before you begin with developing Shallot plugins.

1.1 Plugin Management Assistant

Shallot includes a plugin manager. It helps to work with plugins as it provides an overview of all installed plugins and some management actions you can execute on them (e.g. deleting, disabling).

It is not enabled by default, since it is not required for day-to-day use. For plugin development, you should enable the Plugin Management in the *Tuning* dialog.

It also provides a collection of sample plugins. They can either be used as some kind of a tiny step-by-step tutorial or as templates for your new plugins.

1.2 General Plugin Design

Shallot plugins are Python scripts in a single file with the filename *[pluginname].py* (replacing [pluginname] with the actual name). They must *import shallot* and use the methods provided there for interacting with the Shallot core.

1.3 Implement & Register

For the most kinds of functionality you can add to Shallot, the development pattern is as follows (everything happening in your Python plugin file):

- At first you subclass one of the Shallot base classes and put your functionality into the class implementation. The documentation of the base class tells which methods may and/or must be implemented in your subclass.
- Then, in the *main* section of that plugin - this is the unindented root block as it is in typical executable Python scripts - the code must add this new subclass to the Shallot runtime. This happens with one of the *register** methods in the Shallot interface.

Which particular elements of the Shallot interface are used, depends on what kind of functionality you want to add. The section h_featureoverview below describes the details.

1.4 Position Indexes

For some kinds of pluggable functionality, Shallot keeps an *ordered* list of plugged objects. The semantic of the order can differ, e.g. it can be used for displaying purposes or for an execution order.

The technical mechanism is similar on most places in the interface. Particular functions (i.e. some constructors and register methods) have this two arguments amongst others:

- *positionGroup*: A coarse order information. This must be set to one of the elements in a particular enumeration (see the function documentation for details).
- *positionIndex*: An integer between 0 and (at least) 10000 which controls the order within the chosen group. Lower values mean sooner placement in the order. They are not required to be continuous but can have holes and also duplicates (although the order of duplicates to each other is then undefined). Existing larger indexes will also not shift (as they would do in a typical list) on insertion of a new item.

The documentation texts of them explain what the order is used for. Those two arguments together control the placement in the ordered object list. Typically both ones are optional and have a default:

- *positionGroup*: Some fixed default group. The function documentation might mention details.
- *positionIndex*: Some random-like integer, which is derived from your class name (and so is always the same in each run on each machine).

If so, you can decide to use those defaults, or to manually set either one or both of them.

If the order is critical (e.g. because it is the execution order of a process which only succeeds in a certain order), it is probably required to manually set them. Obviously you must then find out the values from the related objects (e.g. by trial-and-error) and tune your values according to them.

If not, you are fine with the defaults in many cases.

1.5 The '_ApiProxy' Class

Studying the reference, you will see the class *shallot._ApiProxy* be referred at many places. Whenever this is the case, e.g. as a superclass of some interface classes, this is mostly a technical information with no big impact on the developer. It is a common superclass for many classes in the Shallot scripting interface, since it gives them access to some internal parts of the infrastructure. You should not directly come in touch with any specifics of this class.

1.6 Exceptions

The base class for exceptions in Shallot is shallot.Exceptions.ScriptedException. The shallot.Exceptions namespace contains more interesting stuff.

Try to only throw those exceptions (either directly or indirectly) from your code to be failsafe. You should use one of the subclasses in *raise* statements. You should never use subclasses in *expect [Foo]* statements but only shallot.Exceptions.ScriptedException. Use shallot.Exceptions.ScriptedException.isExceptionClass for checking against a certain class (as string) and rethrow if needed. This is because a exception in Shallot typically has more classes than the direct Python side hierarchy would look like.

1.7 Localization

Shallot plugins can support more than just one language. For multi-language support, you should create a global instance of shallot.IntlStringMap at the beginning and take strings from there when they are used for displaying:

```
import shallot

Strings = shallot.IntlStringMap(
    HelloWorld = {"en":"Hello world!", "de":"Hallo Welt!"},
    SomethingElse = {"en":"Something different", "de":"Etwas anderes"},
)

shallot.Logging.log_info(Strings.HelloWorld)
```

1.8 Deployment

The deployment of a Shallot plugin is as easy as copying the plugin file. However, the destination differs among the various operating systems. Since there is a system-global location and a per-user location, there is one more degree of variation.

On the target system, you can find out the actual destination paths by means of the 'Find in filesystem' action in the h_pluginmanager.

Once the destination path is known, you can choose an arbitrary deployment technique - e.g. manual file copying, or using the native package manager from the operating system - for bringing the plugin file in place.

CHAPTER

TWO

FEATURE OVERVIEW

This sections introduces the different kinds of functionality you can add by means of this plugin interface. Those are the technical building blocks you have to use for implementing your plugin logic.

Most of the sections begin with an abstract explanation of the mechanisms. Read ahead! It should become clear later in the text. Each section includes links to the elements in the scripting interface reference which you need to know. You should read the documentations of this elements.

2.1 Actions

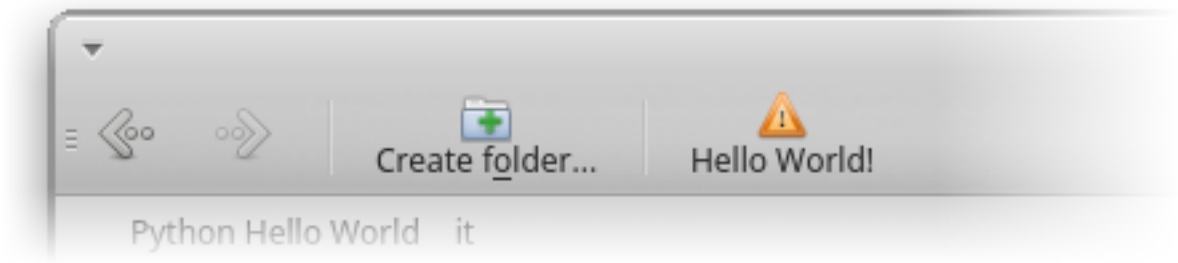
As a first approximation, an action is some piece of code for execution. Conceptually an action is similar to a simple Python function (although it technically is not just a function as you will read later). However, executing them brings a lot of additional functionality (e.g. dialog support for user communication) and infrastructure.

Actions are used in different ways in Shallot:

- The easiest thing to do with an action is to simply execute it. This is similar to directly executing the code, but using the additional functionality.
- Another common usage is to offer them in the user interface for some files or directories, so the user can decide to execute them.
- There are some other places in the plugin interface, which work with actions somehow. Read those documentations for details.

The first way gives a very versatile tool. In most places in the plugin code, you can put some parts of your routine into an action and execute it in order to use the additional functionality we will see later.

The second way enables your plugin to provide some code for a given selection of files and/or directories for the user, so the user can manually trigger it. Those are typically represented in context menus or the toolbar.



At this point, our first approximation must see some corrections:

An action is a (indirect) subclass of shallot.Actions.AbstractAction. It has the following subclasses, which are typical base classes for custom implementations:

- shallot.Actions.ActionAction: This kind of action is executable.
- shallot.Actions.SubmenuAction: A submenu is a list of other actions. It is not directly executable, but can be presented as a submenu to the user. This allows to bundle multiple shallot.Actions.ActionAction implementations which provide related functionality for better overview.

A shallot.Actions.ActionAction instance (i.e. an instance of your subclass) can be executed by initializing it at first (calling shallot.Actions.ActionAction.initialize_sync) and calling shallot.Actions.ActionAction.execute.

For sample code, read the sources of the 'action' sample plugin (in the h_pluginmanager).

In order to provide your action subclass in the user interface, call shallot.Actions.register.

For sample code, read the sources of the 'action.advanced' sample plugin.

The additional functionality comes with the *info* parameter you get in your shallot.Actions.ActionAction.action implementation. It is an instance of shallot.Actions.ExecutionInfo. Read this documentation for a full overview. One essential thing you can do with such an object is to have dialogs with the user. The next section shows how to do this.

The shallot.Actions namespace contains more interesting stuff.

2.2 User Feedback

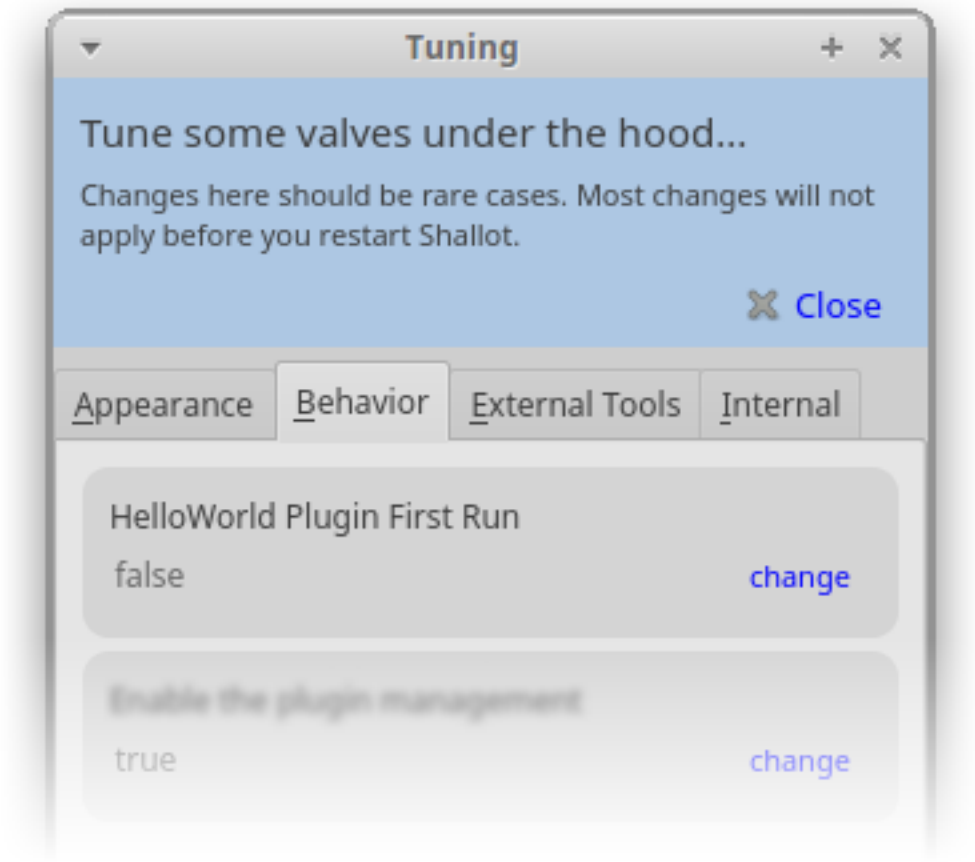
User feedback is everything about giving some information to the user and waiting for its answer in an interactive way.

Typical situations where user feedback operations take place are Action executions. There an instance of shallot.Actions.ExecutionInfo is available. shallot.Actions.ExecutionInfo.userfeedback instance gives you access to an instance of shallot.Actions.ExecutionUserFeedback, which is the key element to all kinds of feedback operations.

For sample code, read the sources of the 'userfeedback' sample plugin (in the h_pluginmanager).

2.3 Configuration Values

A plugin can use own configuration values which are stored by Shallot and are visible to the user in the *Tuning* dialog.



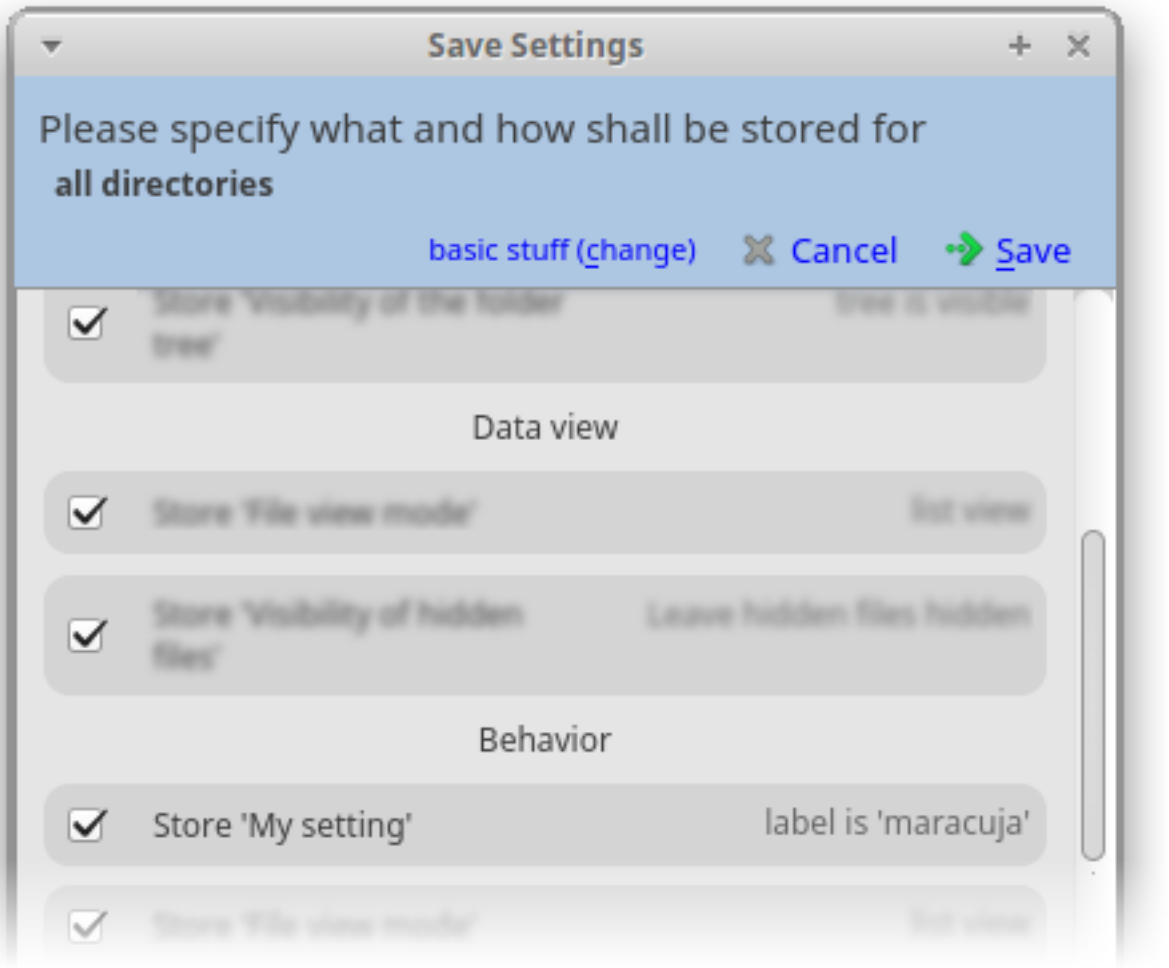
They are fine for making stuff configurable for special situations, which should typically should stay at default. For many other situations, e.g. when the user is expected to changes this value regularly, check if the Settings features fit better.

Using an own configuration value begins with subclassing shallot.ConfigurationValue. Afterwards, you should create one instance of this class globally and use its methods. No registration is required.

For sample code, read the sources of the 'configurationvalue' sample plugin (in the h_pluginmanager).

2.4 Settings

Settings are a common mechanism for various kinds of customizations the user can make in its Shallot environment. The user can store them, bind them to particular subdirectories and profiles and manage them in the *Settings* dialog.



A plugin can participate in this mechanism by adding own settings.

This begins with subclassing shallot.Setting. Afterwards, shallot.Setting.register must be used for registering it.

For sample code, read the sources of the 'setting' sample plugin (in the h_pluginmanager).

2.5 Main Window

The shallot.MainWindow class provides some navigation function and more. Read the class documentation for details. The main window object is globally available as shallot.MainWindow.current.

For sample code, read the sources of the 'mainwindow' sample plugin (in the h_pluginmanager).

2.6 EURLs

EURLs are addresses of objects in the filesystem. They are a very basic part of the Shallot foundation.

Conceptually EURLs are similar to [URLs](https://en.wikipedia.org/wiki/Uniform_Resource_Locator). In order to express cascades of locations, EURLs are defined as a superset of URLs. As a meaningful example, this EURL refers to a file in an archive file, which in turn is located on a remote server:

zip://[ftp://ftpserver/foo/bar.zip]//zippedfolder/zippedfile

This cascades can have arbitrary depths (although this obviously doesn't make sense in arbitrary combinations).

EURLs are used in many different parts in the scripting interface. They are instances of the class shallot.Eurl, which provide a rich set of getters and creator functions.

For sample code, read the sources of the 'eurl' sample plugin (in the h_pluginmanager).

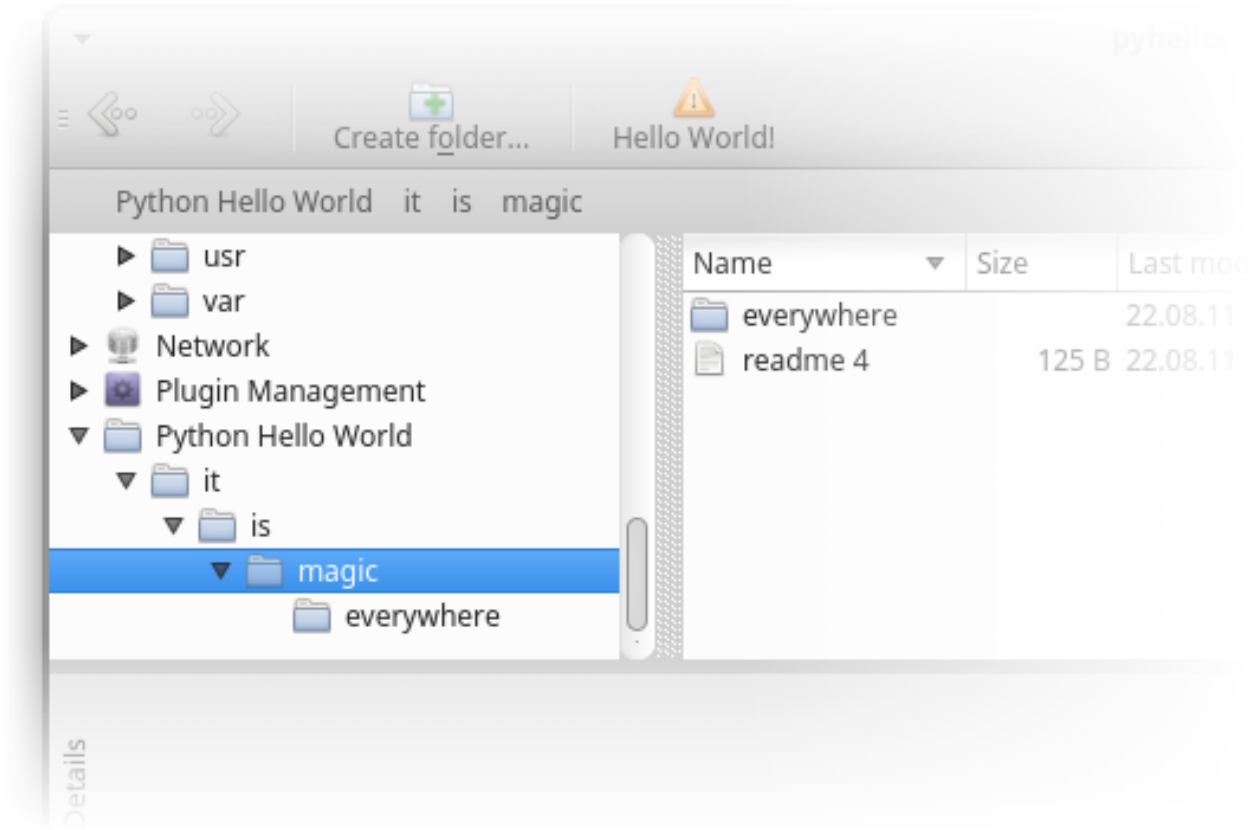
2.7 Filesystem

Shallot keeps an own model of the entire filesystem tree in memory (technically, it determines and stores stuff just on demand, of course). A Shallot plugin has different ways to work with this model.

At first, there are some methods for requesting some informations directly from the model. Find the static methods in shallot.Filesystem for more details (not all of them request things but some are relevant for the stuff explained later). A key class in interactions with the model, also used at various different parts in the scripting interface, is shallot.Filesystem.Node.

For sample code, read the sources of the 'filesystemnode' sample plugin (in the h_pluginmanager).

A different way of using it filesystem model is to provide custom implementations and own nodes in the tree.



This at least requires to subclass shallot.Filesystem.Handler and registering it with shallot.Filesystem.Handler.register.

In typical scenarios it also requires to create a shallot.Filesystem.Node which is associated with this handler and to insert it into the model. Some of the static methods of shallot.Filesystem are used for this procedure. Please note that it is just required to insert root nodes somewhere. The subtree of that node is specified by your handler implementation, but the node insertions (and removals) take place implicitely.

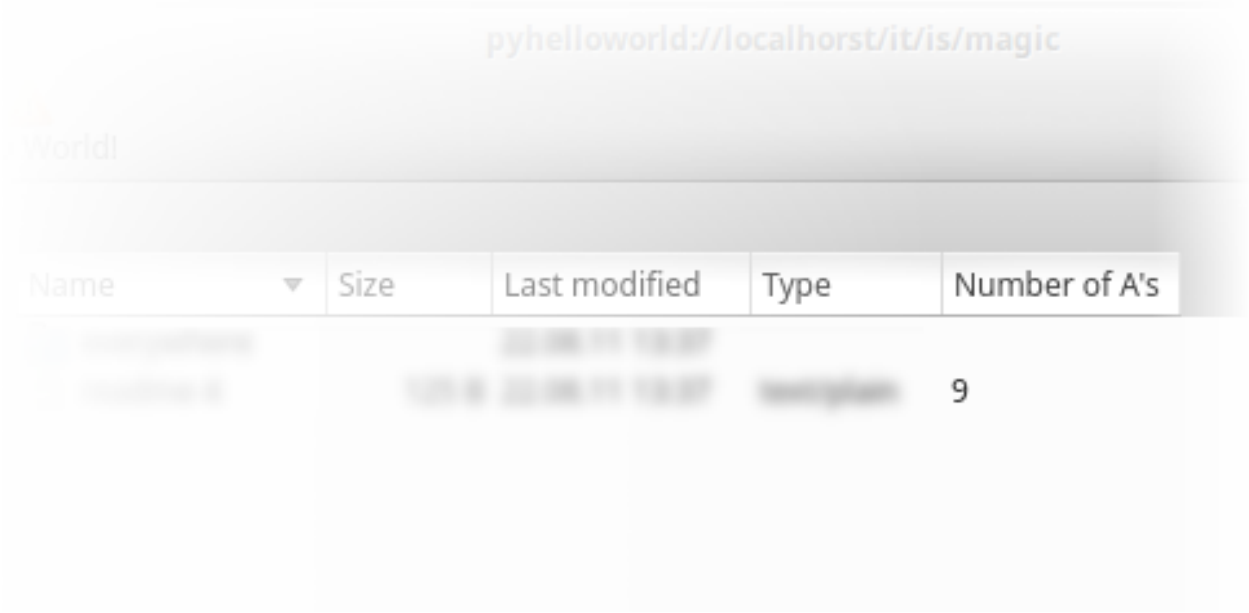
You should also read about Operations, since you might have to deal with the key object of this feature here as well.

For sample code, read the sources of the 'filesystem' sample plugin (in the h_pluginmanager).

A filesystem handler has a wide range of possibilities for extending the Shallot functionality. One of them are custom detail columns, which are subject of the next section.

2.8 Detail Columns

A custom detail column, as it is visible in file views, can offer additional information to the user.



Implementing custom detail columns begins with subclassing shallot.DetailColumn. You should then create one instance of this class and add it to some nodes with shallot.Filesystem.Node.add_detail.

For sample code, read the sources of the 'filesystem' sample plugin (in the h_pluginmanager).

2.9 Operations

Operations are a concept of bundling a bunch of small changes in the filesystem together to one large step. They are also essential for working in cascaded filesystems as they are expressed with EURLs. Operations are entirely a backend concept without any direct representations in the user interface.

A small example explains why this bundling is a required thing: There is a ftp server which contains two archive files; *a.zip* (A) and *b.zip* (B). They build the roots of the subtrees *zip:/[ftp://foo/a.zip]//* and *zip:/[ftp://foo/b.zip]//*, each containing the contents of the respective archive. The user requests a large file copy action for 1000 files in *zip:/[ftp://foo/a.zip]//* to somewhere in *zip:/[ftp://foo/b.zip]//*. In an unbundled way, Shallot would at first fetch A, then extract the first file, then fetch B, put the file content into the archive and upload the new version of B again. Afterwards, it would run the same loop for the second file, then for the third, ... In an operation, Shallot would fetch A and B just once, then works on the local versions and just uploads the final new version of B in the end.

This bundling is realized in the shallot.Operations.Operation class. You get an instance of it in those different ways:

- In the most situations, where filesystem operations potentially make sense, there is an instance directly available as a parameter in a method of your subclasses. The documentation of the Shallot interface classes should help.
- Sometimes there is an instance available, but it is not directly available from the function parameters. In such cases, the documentation should help as well. A typical example is the execution of Actions. Your implementation of shallot.Actions.ActionAction.action gets called with the parameter *info*, which is a shallot.Actions.ExecutionInfo instance. It has the member shallot.Actions.ExecutionInfo.operation which returns a operation object.

- In rare cases you have to explicitely create one. This is when you don't get an existing instance from somewhere, but you need one for some reasones. You can create a new instance with shallot.Operations.Operation.create, but then it must be manually committed with shallot.Operations.Operation.commit in order to transfer all the new versions to their real destinations.

The key method for using the bundling mechanism is shallot.Operations.Operation.fetch_file (and shallot.Operations.Operation.fetch_container_file, which is just a convenience variant with a subtle difference in the arguments). Whenever you need to read the content of a file in order to eventually replace it with a modified version, you should use it.

Please note that operations are also by the only convenient method you would even have whenever your code potentially works with cascades of filesystems. Whenever you need to get the actual data streams of the affected archives for some location like *zip://[zip://[ftp://foo/a.zip]//another.zip]//somephoto.jpeg*, calling shallot.Operations.Operation.fetch_file for some part of this address is an easy way (e.g. a shallot.Filesystem.Handler implementation for a custom archive format will need to do that all the time).

For sample code, read the sources of the 'operation' sample plugin (in the h_pluginmanager).

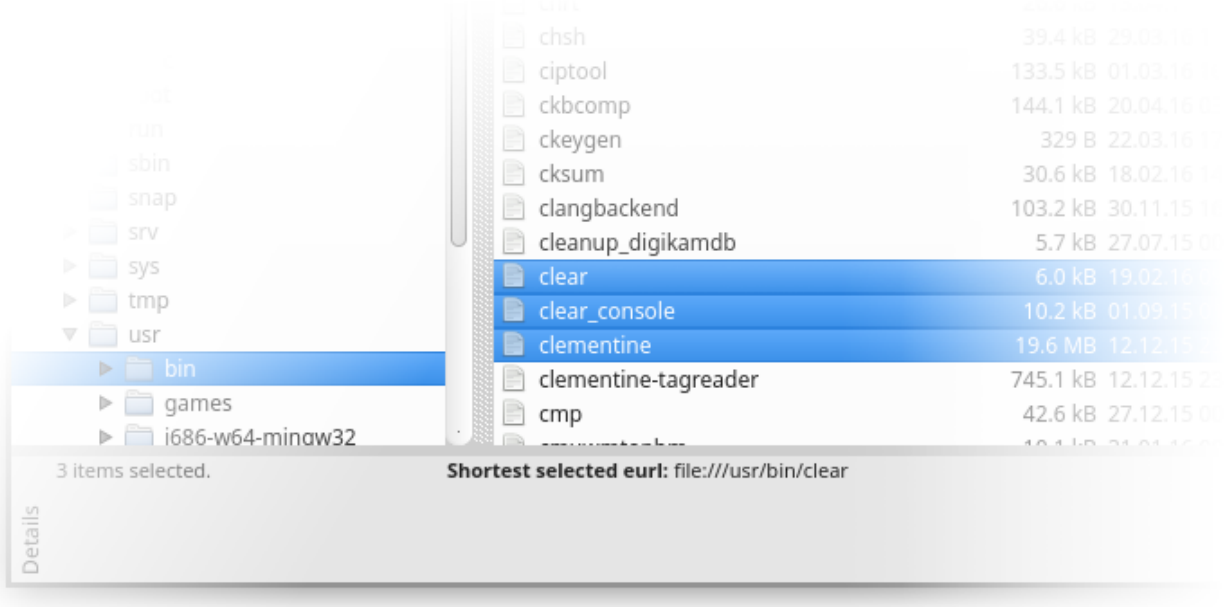
The shallot.Operations.Operation class provide some more. Amongst others, there is the shallot.Operations.Operation.filesystem member, which provides you access to an instance of shallot.Operations.FilesystemOperation. It provides some primitive steps you can also execute in a bundled way. See the class documentation for details.

For sample code, read the sources of the 'filesystemoperations' and 'filesystemoperations.advanced' sample plugins.

The shallot.Operations namespace contains more interesting stuff.

2.10 Panel Details

Panel details can be provided by a plugin in order to provide the user with additional information about a selection of one or more files in the *Details* part of the Shallot window.



This often is used for presenting some special kinds of file metadata.

The first step to a custom panel detail is to subclass either shallot.PanelDetails.SingleSelectionPanelDetail

or shallot.PanelDetails.MultiSelectionPanelDetail. An instance of it must be registered with shallot.PanelDetails.PanelDetail.register.

For sample code, read the sources of the 'paneldetails' sample plugin (in the h_pluginmanager).

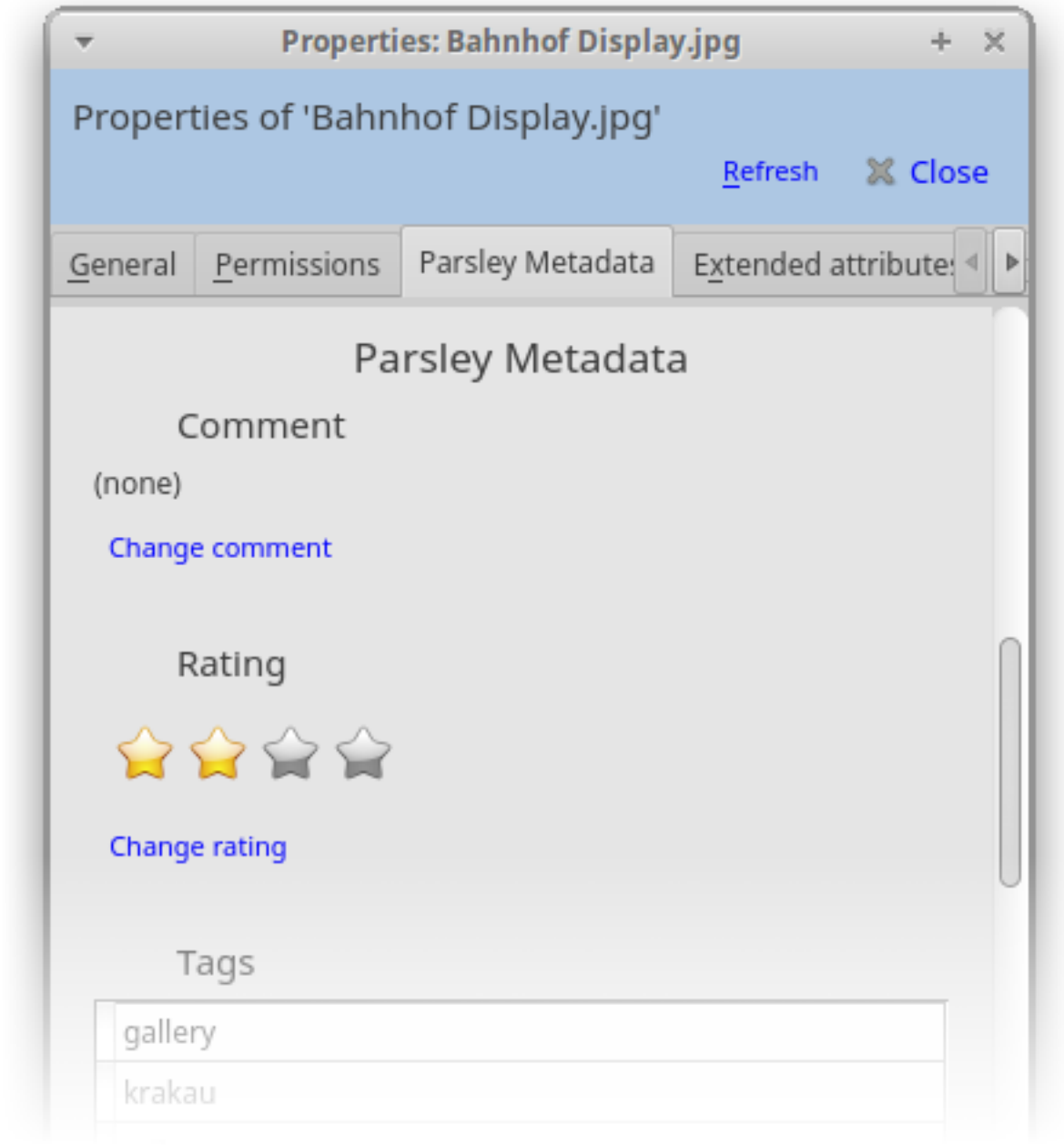
The shallot.PanelDetails namespace contains more interesting stuff.

2.11 File Property Dialog

The file property dialog is another place where plugins can provide custom pieces of information (and user interactions).

2.11.1 Dialog Tabs

A custom tab in the property dialog can show one or more information pieces in different views. It can also offer buttons for user interaction.



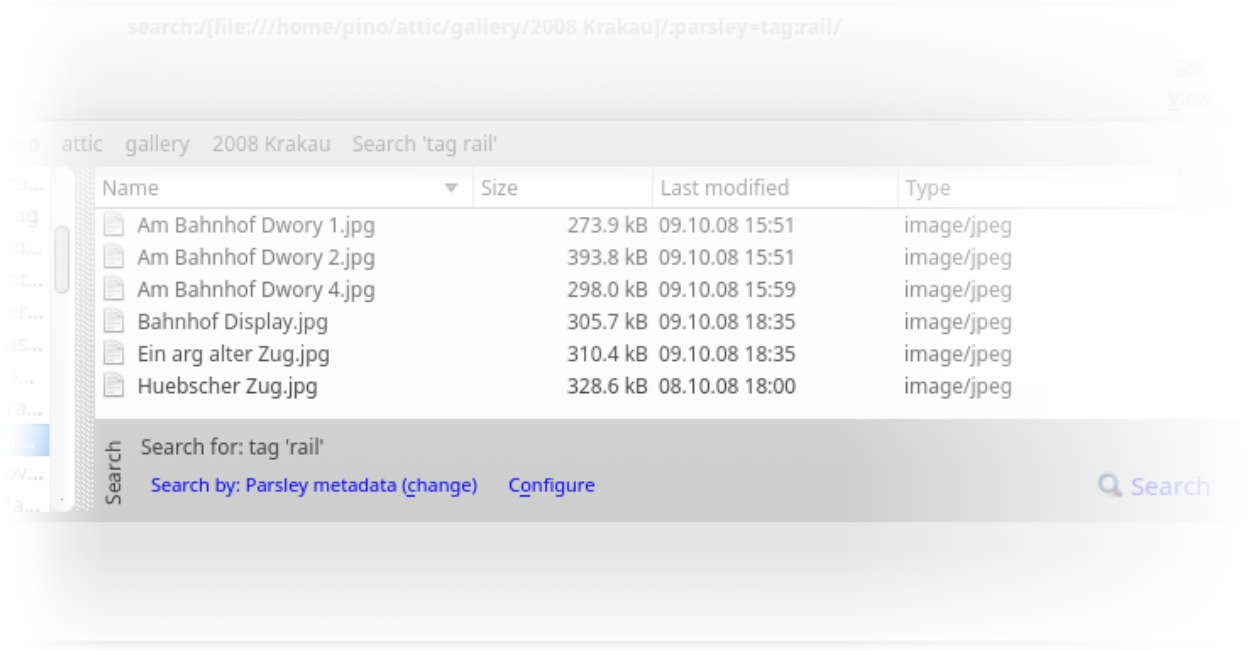
The first step is to subclass shallot.FilePropertyDialog.Tab and to register this class with shallot.FilePropertyDialog.Tab.register.

For sample code, read the sources of the 'filepropertydialogtab' sample plugin (in the h_pluginmanager).

The shallot.FilePropertyDialog namespace contains more interesting stuff.

2.12 File Searches

Custom search criterion implementations bring additional functionality to Shallot file searches.



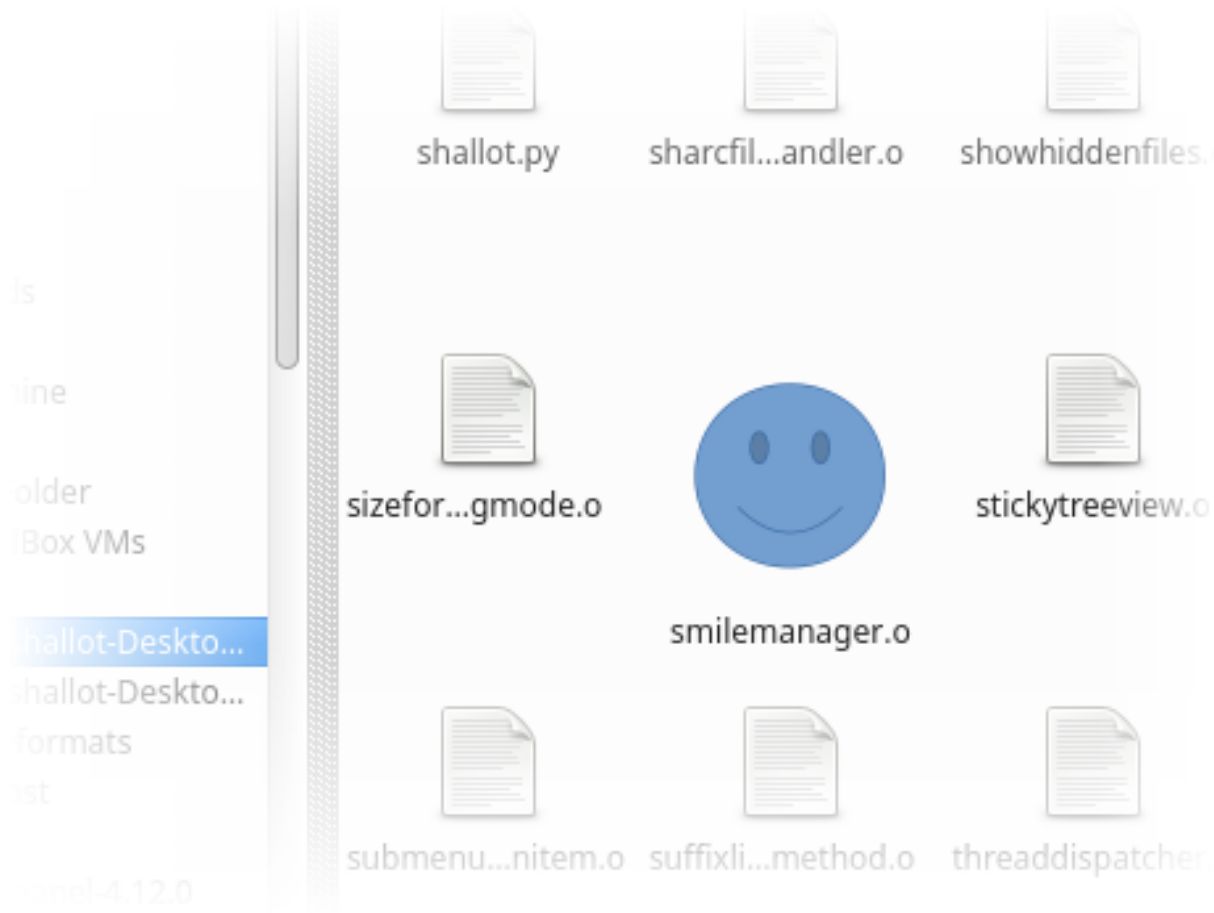
The first step is to subclass shallot.FileSearch.SearchCriterion and also shallot.FileSearch.SearchCriterionFactory (the latter one has to refer to the former one in the constructor call; see class documentations). Register the latter one with shallot.FileSearch.SearchCriterionFactory.register.

For sample code, read the sources of the 'searchcriterion' sample plugin (in the h_pluginmanager).

The shallot.FileSearch namespace contains more interesting stuff.

2.13 Thumbnails

Thumbnails are visible in some view modes only. In order to support a new file format or to show certain custom thumbnail content in particular situations, a plugin can add a thumbnail provider implementation to the thumbnail creation mechanism.

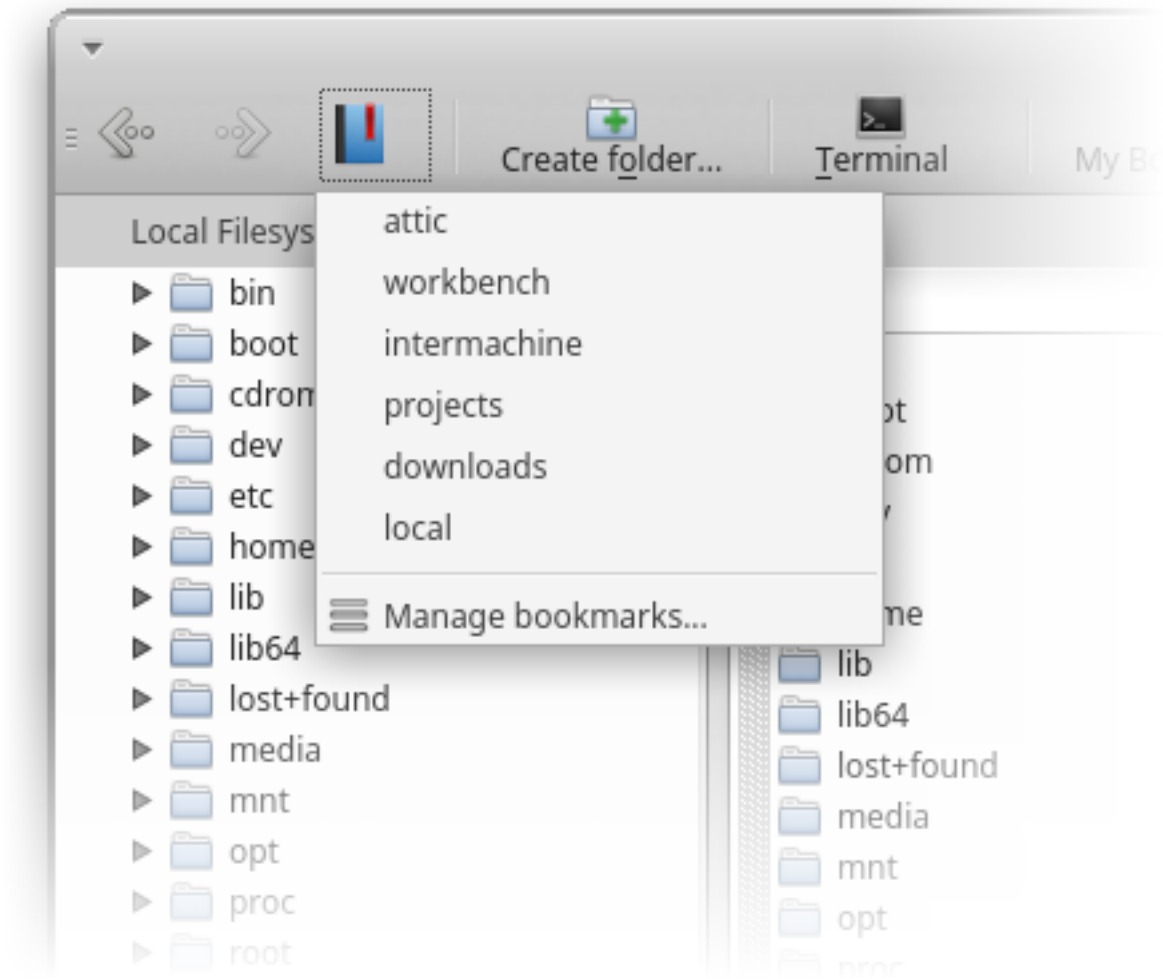


The first step is to subclass shallot.ThumbnailProvider and to register an instance of this class with shallot.ThumbnailProvider.register.

For sample code, read the sources of the 'thumbnailprovider' sample plugin (in the h_pluginmanager).

2.14 Bookmarks

Bookmarks are shortcuts somewhere in the user interface, which allow the user to fastly jump to some common filesystem locations.



The bookmark menu becomes visible once some bookmarks exist. A plugin can deal with bookmarks as well. It can create and manage them with the methods in shallot.Bookmarks.

For sample code, read the sources of the 'bookmarks' sample plugin (in the h_pluginmanager).

2.15 Logging

Plugins can write all kinds of information to the Shallot log in order to find potential issues. See shallot.Logging for details.

2.16 Utilities

There are some utilities for working with the environment available in shallot.Environment. This includes shallot.Environment.Thread for running code asynchronously, shallot.Environment.Timer for running a code in regular intervals and more.

CHAPTER

THREE

API REFERENCE

3.1 shallot module

class shallot.**Actions**
Bases: object

Everything about actions.

class AbstractAction
Bases: object

Abstract base class for executable actions or submenu structures of them. See the subclasses of this class. They can be registered with shallot.Actions.register. They can be returned from shallot.Filesystem.Handler.get_actions.

enabled()
Checks if this action is enabled.

initialize()
Initialize the action. This should make the time-consuming parts, e.g. for determining a label or enabled state. *Override this method in custom subclasses or leave the default implementation.*

set_enabled(*value*)
Sets if the item is enabled.
Parameters **value** – The new value.

set_visible(*value*)
Sets the visibility of this item.
Parameters **value** – The new value.

visible()
Checks the visibility of this item (non-recursively).

class ActionAction(*text*, *enabled=True*, *icon=''*, *defaultActionPrecedence=0*)
Bases: *shallot._ApiProxy*, *shallot.Actions.AbstractAction*

Abstract base class for an executable action, which can be made visible in menus or the toolbar or executed directly. See shallot.Actions.AbstractAction for more.

Parameters

- **text** – A label text (can be "" if you want to directly execute it instead of adding it to some menu structure).
- **enabled** – If this action is enabled.
- **icon** – Name of an icon.

- **defaultActionPrecedence** – Precedence value for being a default action. This int value must be higher than all others for becoming the default action. See shallot.Actions.DefaultPrecedenceValues.

action(*info*)
The action implementation, i.e. what the actions should actually do. *Override this method in custom subclasses.*
Parameters **info** – A shallot.Actions.ExecutionInfo execution info object.

execute()
Executes this action.

initialize_sync()
Initializes the action. This can be called from outside in order to do the initialization.

class ByRegExpPredicate(*pattern*)
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only when the selection paths matches a regular expression. See base class for more information.

class DefaultPrecedenceValues
Bases: object

Reference values for calculating default precedence values of open actions.

class DontResolveLinksPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Disables links resolving. See base class for more information.

class ExecutionInfo
Bases: object

An object for signalling action execution state changes between an action implementation and the Shallot core (mostly to the core). Most calls lead to changes in the information presented by the progress dialog.

Can't be constructed directly.

add_changed_eurl(*eurl*)
Requests updating the filesystem model info for an item.
Parameters **eurl** – The item location to be updated as shallot.Eurl.

cancel()
Cancel the action.

from_objectname()
Gets the current object name on from-side.

from_verb()
Gets the current verb on from-side.

head()
Gets the head text.

is_cancelled()
Is the action cancelled?

is_manual_intervention_needed()
Is manual intervention needed?

is_visualprocessfeedback_active()
Is visual feedback visibility enforced?

operation()
Gets the shallot.Operations.Operation transactional operation object.

progress_all()
How many items are to do in sum?

progress_done()
How many items are done?

progress_text()
The textual representation of the progress.

respect_cancel()
Respect a cancel request. This should be called from time to time (inside loops for example).

set_details(*fromverb*, *fromobjectname*, *toverb*, *toobjectname*)
Sets the progress details.
Parameters
- **fromverb** – The verb on from-side (the source).
- **fromobjectname** – (file path) The objectname on from-side (the source).
- **toverb** – The verb on to-side (the destination).
- **toobjectname** – (file path) The objectname on to-side (the destination).

set_head(*txt*)
Sets the head text.

set_manual_intervention_needed(*v*)
Set if manual intervention is needed.

set_progress(*done*, *all*, *label*)
Sets a current determinate progress.
Parameters
- **done** – The number of finished items.
- **all** – The gross number of items.
- **label** – Progress description text.

set_progress_indeterminate(*label*)
Sets a current indeterminate progress.
Parameters **label** – Progress description text.

set_visualprocessfeedback_active(*v*)
Sets the visibility of a visual feedback.

to_objectname()
Gets the current object name on to-side.

to_verb()
Gets the current verb on to-side.

userfeedback()
The shallot.Actions.ExecutionUserFeedback user feedback object.

class ExecutionUserFeedback
Bases: `object`

An object for communication with the user in action implementations. It can be used to ask the user for some information or to just give some information to the user.

Can't be constructed directly.

class MessageBoxButton
Bases: `object`

Buttons in a message box from shallot.Actions.ExecutionUserFeedback.

messagebox(*message*, *buttons*, *icon*, *defaultbutton*, *abortbutton*)
A message box. Returns the index of the chosen button.

Parameters
- **message** – The message text.
- **buttons** – A list of buttons as string list.
- **icon** – An icon name.
- **defaultbutton** – Answer for keyboard Enter as shallot.Actions.ExecutionUserFeedback.MessageBoxButton.
- **abortbutton** – Answer for keyboard Esc as shallot.Actions.ExecutionUserFeedback.MessageBoxButton.

simple_inputbox(*question*, *defaulttext*)
A simple input box.

Parameters
- **question** – The question.
- **defaulttext** – The default answer.

simple_messagebox(*message*, *buttons*, *icon*, *defaultbutton*, *abortbutton*)
A simple message box. Returns the chosen button.

Parameters
- **message** – The message text.
- **buttons** – A list of buttons as or-combination of shallot.Actions.ExecutionUserFeedback.MessageBoxButton.
- **icon** – An icon name.
- **defaultbutton** – Answer for keyboard Enter as shallot.Actions.ExecutionUserFeedback.MessageBoxButton.

May be 0 for no definition. :param abortbutton: Answer for keyboard Esc as shallot.Actions.ExecutionUserFeedback.MessageBoxButton. May be 0 for no definition.

class HideOnCurrentDirectoryLevelPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only when not searching for 'current directory level' actions. See base class for more information.

class HideOnSelectionLevelPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only when not searching for 'selection level' actions. See base class for more information.

class KeyShortcutPredicate(*shortcut*, *triggers_on_currentdirectory_level*)
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Sets a keyboard shortcut. See base class for more information.

class OnDirectoriesPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only on directories. See base class for more information.

class OnFilesPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only on files. See base class for more information.

class OnLinksPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only on links. See base class for more information.

class OnSingleEntrySelectionPredicate
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Shows actions only on single-entry selections. See base class for more information.

class PositionIndexPredicate(*index*)
Bases: *shallot.Actions.Predicate*, *shallot._ApiProxy*

Sets a positioning information index. See base class for more information.

class Predicate
Bases: object

Controls when and how an action is created. Used in shallot.Actions.register. See base class for more information.

class SubmenuAction(*text*, *enabled=True*, *icon=''*, *defaultActionPrecedence=0*)
Bases: *shallot._ApiProxy*, *shallot.Actions.AbstractAction*

Abstract base class for a submenu action, which can be made visible in menus or the toolbar. See shallot.Actions.AbstractAction for more.

Parameters

- **text** – A label text.
- **enabled** – If this action is enabled.
- **icon** – Name of an icon.
- **defaultActionPrecedence** – Precedence value for being a default action. This int value must be higher than all others for becoming the default action. See shallot.Actions.DefaultPrecedenceValues.

set_subitems(*subitems*)
Sets the subitems.
Parameters **subitems** – A list of shallot.Actions.AbstractAction instances.

static _factory(**a*)

static register(*actiontype*, *category='manage'*, *predicates=[]*)
Registers a subclass of shallot.Actions.AbstractAction. This makes it permanently visible in the context menu of selections and/or in the toolbar. If and where the action is visible depends on the action implementation (see constructor parameters), the situation (what is selected in the user interface?) and the registration *mode*.

Parameters

- **actiontype** – The class implementing shallot.Actions.AbstractAction. Its constructor must be callable with just *actiontype(nodes)*. The *nodes* parameter contains a list of shallot.Filesystem.Node containing the node selection.
- **category** – The action category. Used for grouping in menus. Typically available are *"create"*, *"open"* and *"manage"*.
- **predicates** – A list of shallot.Actions.Predicate controlling when and how the action is offered.

class shallot.**Bookmarks**
Bases: object

Everything about TODO h_bookmarks "Bookmarks".

class Bookmark
Bases: object

A bookmark. Change it with the methods in shallot.Bookmarks.

Can't be constructed directly.

eurl()
The location this bookmark points to.

folder()
The subcollection (as list of strings).

id()
The bookmark id. It is not display anywhere but only used for the management methods in shallot.Bookmarks.

label()
A textual description of this bookmark.

tags()
The tags strings. This value is solely used by plugins for their bookkeeping. It can allow to programmatically find a particular bookmark (if the creator wrote meaningful information in it).

static add_bookmark(*eurl*, *, *folder=[]*, *label=None*, *tags=None*)
Adds a new bookmark.

Parameters

- **eurl** – The location to bookmark.
- **folder** – A string list describing in which subcollection this bookmark should be placed.
- **label** – The textual description of the new bookmark.
- **tags** – A string for internal bookkeeping. Use it for recognizing your own bookmarks later on. See shallot.Bookmarks.Bookmark.tags.

static change_bookmark(*id*, *eurl*, *label=None*)
Changes bookmark data.

Parameters

- **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).
- **eurl** – The new location.
- **label** – The new textual description.

static change_bookmark_tags(*id*, *tags*)
Changes bookmark tags. See shallot.Bookmarks.Bookmark.tags.

Parameters

- **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).
- **tags** – The new tags string.s

static get_bookmarks()
Returns a list of all existing shallot.Bookmarks.Bookmark instances.

static has_bookmarks()
Returns True if there are any bookmarks stored.

static move_bookmark_down(*id*)
Moves a bookmark downwards in its collection.

Parameters **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).

static move_bookmark_to_folder(*id*, *folder*)
Moves a bookmark to another subcollection.

Parameters

- **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).
- **folder** – The new subcollection (as list of strings).

static move_bookmark_up(*id*)
Moves a bookmark upwards in its collection.

Parameters **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).

static remove_bookmark(*id*)
Removes a bookmark.

Parameters **id** – The bookmark id this call is about (See shallot.Bookmarks.Bookmark.id).

class shallot.**ConfigurationValue**(*name*, *defaultvalue*, *category=0*, *description=''*, *longdescription=''*, *changehint=None*)
Bases: *shallot._ApiProxy*

Abstract base class for a configuration value.

Parameters

- **name** – The config value name.
- **defaultvalue** – The default value.
- **category** – The category. One of shallot.ConfigurationValue.Category.
- **description** – The short description.
- **longdescription** – The long description.
- **changehint** – A label hint for change buttons in user interfaces.

class Category
Bases: object

Categories of shallot.ConfigurationValue implementations. They are used for grouping them in the dialogs. There is no difference in behavior implied by this choice.

set_value(*value*)
Sets the value.

Parameters **value** – The new value.

value()
Returns the current value.

class shallot.**DetailColumn**(*name*, *displayname*, *positionGroup=None*, *positionIndex=None*, *sort_doTypediff=True*, *defaultWidth=-1*, *isRightAligned=False*)
Bases: *shallot._ApiProxy*

Abstract base class for a detail column for shallot.Filesystem.Node instances.

Those can e.g. be seen in the file list view, but can also be used internally by other places.

It encapsulates the retrieval logic and metadata for one piece of additional information a shallot.Filesystem.Node can have (e.g. the filesize). Retrieving the values is designed to be asynchronous. Each instance represents one column, while the actual logic is implemented in subclasses. For a new detail column, subclass this class and implement at least shallot.DetailColumn.determine_value.

Parameters

- **name** – The internal name (must be unique).
- **displayname** – The label text.
- **positionGroup** – Controls display order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.DetailColumn.
- **positionIndex** – Controls display order. See Position Indexes for details.
- **sort_doTypediff** – Shall sorting differ between files and directories?
- **defaultWidth** – The default width in pixels (optional).
- **isRightAligned** – If the column values are right aligned (optional).

apply_value(*eurl*, *operation*, *value*)
Set the detail value for a given eurl. Used for transferring details in file transfers (see shallot.DetailColumn.register_as_transferrable). *Override this method in custom subclasses or leave the default implementation.*

Parameters

- **eurl** – The shallot.Eurl for which the value must be determined.
- **operation** – The shallot.Operations.Operation operation object.
- **value** – The detail value (as string).

compute_value(*node*, *operation*)
Queries the column value for a node.

Parameters

- **node** – The shallot.Filesystem.Node node for which the value must be determined.
- **operation** – The shallot.Operations.Operation operation object.

determine_value(*node*, *operation*)
Determines the column value for a node. *Override this method in custom subclasses.* Only used internally. For querying foreign detail columns, use compute_value instead.

Parameters

- **node** – The shallot.Filesystem.Node node for which the value must be determined.
- **operation** – The shallot.Operations.Operation operation object.

static find_by_name(*name*)
Finds a shallot.DetailColumn implementation by name.

Parameters **name** – The detail column name.

static register_as_transferrable(*index*, *detailcolumn*)
Registers a shallot.DetailColumn for transferring those details in file transfers (e.g. when the user copies files). This requires to implement some methods.

Parameters

- **index** – An integer which controls the order of transferring.
- **detailcolumn** – The shallot.DetailColumn detail column.

class shallot.**Environment**
Bases: object

Environment information and some utilities.

class Thread

Bases: *shallot._ApiProxy*

A thread executes code asynchronously. See also the Python documentation about threading. Although this thread class is not related to the Python thread class (in terms of object orientation), you should understand the general pitfalls which come with multithreading and you might find the Python synchronization tools useful for avoiding those pitfalls. Note: It is strictly forbidden to 'park' a thread and wait for some external event most of the time. Use a shallot.Environment.Timer for recurring tasks.

static execute_threaded(*fct*, **args*, ***kwargs*)

Executes a function in a new shallot.Environment.Thread (and returns that). This avoids subclassing and makes the code more compact. 'args' and 'kwargs' are additional parameters which become the parameters in the actual call of 'fct'.

Parameters **fct** – The function to execute.

run()

Contains the code to execute in this thread. *Override this method in custom subclasses.*

start()

Starts the execution of this thread and calls 'run' in that thread. Returns immediately.

class Timer

Bases: *shallot._ApiProxy*

A timers executes some code recurringly in some time interval.

is_started()

Returns if the timer is currently active.

run()

Contains the code to execute. *Override this method in custom subclasses.*

start(*interval*)

Starts the timer, which executes 'run' each 'interval' milliseconds. Returns immediately.

Parameters **interval** – The interval in milliseconds.

static start_in_timer(*fct*, *interval*, **args*, ***kwargs*)

Starts a function in a shallot.Environment.Timer (and returns that). This avoids subclassing and makes the code more compact. 'args' and 'kwargs' are additional parameters which become the parameters in the actual call of 'fct'.

Parameters

- **fct** – The function to execute.
- **interval** – The interval in milliseconds.

stop()

Stops the timer.

class shallot.**Eurl**

Bases: object

A EURL.

You can get instances with global methods like shallot.Eurl.create and shallot.Eurl.from_string.

Please note: Instances can be associated with any kind of elements in the filesystem (files, directories, links, ...). It might also point to something which does not exist at all. The documentation sometimes explicitly makes a difference between those kinds (files, directories, links, ...; often called 'node type'). But it often uses the term 'file' implicitly while meaning all kinds of elements; assuming that e.g. a directory is just a special kind of a file. It should be clear from the particular context which meaning applies.

Can't be constructed directly.

as_string()
Returns the textual value. This is what shallot.Eurl.from_string would expect as parameter.

basename()
Returns the last path segment. This is the text behind the last slash. Examples: *"baz"* for *file:///foo/bar/baz*. *""* for *file:///*.

static create(*scheme*, *hostname*, *path*)
Returns a shallot.Eurl from three parts. The structure is *scheme://hostname/pa/th*.

Parameters

- **scheme** – The scheme part.
- **hostname** – The hostname.
- **path** – The path.

enwrap_with_outer_url(*scheme*, *hostname*, *path*)
Returns a new shallot.Eurl containing this one packed as embedding and new outer parts *scheme*, *hostname* and *path*. Example: *scheme:/[file:///a/b/c.zip]/hostname/p/a/t/h* for *file:///a/b/c.zip*.

Parameters

- **scheme** – The scheme part of the new outer eurl.
- **hostname** – The hostname of the new outer eurl.
- **path** – The path of the new outer eurl.

static from_string(*s*)
Returns a shallot.Eurl from a string.

Parameters **s** – The eurl string. This is what shallot.Eurl.as_string would return.

has_inner_urls()
Checks if this shallot.Eurl has embeddings. Examples: *true* for *foo:/[bar:///foo]/host/*. *false* for *foo://host/*.

has_parent_segment()
Checks if this shallot.Eurl has a parent segment. This indicates if shallot.Eurl.parent_segment would return *0*.

hostname()
Returns the hostname part (from the outer url of this shallot.Eurl). Examples: *"livingroom-pc"* for *smb://livingroom-pc/foo/bar/baz*. *""* for *file:///foo/bar/baz*.

is_prefix_of(*eurl*)
Checks if this shallot.Eurl is a prefix of another one. This is not an equivalent to a string comparison but it checks parent relationships according to shallot.Eurl.parent_segment.

Parameters **eurl** – The longer shallot.Eurl.

outer_url()
Returns a new shallot.Eurl containing only the outer part of this one (strips the embeddings). Example: *foobar://host/foo/bar/baz* for *foobar:/[zip:/[file:///a/b/c.zip]//d/e]//foo/bar/baz*.

outer_url_is_root_directory()
Checks if this shallot.Eurl is a root path (with or without embeddings). Examples: *True* for *foo://host/*. *False* for *foo://host/a*. *True* for *foo:/[bar:///goo]/host/*. *False* for *foo:/[bar:///goo]/host/a*. *True* for *foo:/[bar:///goo]//*. *False* for *foo:/[bar:///goo]//a*.

outermost_inner_eurl()
Returns a new shallot.Eurl containing only the embedding of this one. Example: *zip:/[file:///a/b/c.zip]//d/e* for *foobar:/[zip:/[file:///a/b/c.zip]//d/e]//foo/bar/baz*.

parent_segment()
Returns the parent shallot.Eurl. At first, this traverses path segments. For a root path eurl with embeddings, it returns the embedding. If none are available, it returns *0*. Examples: *foo://host/foo* for *foo://host/foo/bar*. *foo://host/* for *foo://host/boo*. *foo:/[bar://host/goo]/anotherhost/* for *foo:/[bar://host/goo]/anotherhost/boo*. *bar://host/foo* for *foo:/[bar://host/foo]/host/*. *0* for *foo://host/*.

path()
Returns the path part (from the outer url of this shallot.Eurl). Examples: *"/foo/bar/baz"* for *smb://livingroom-pc/foo/bar/baz*. *"/foo/bar/baz"* for *file:///foo/bar/baz*. *"/"* for *file:///*.

root()
Returns the root shallot.Eurl from this one. Example: *zip:/[file:///a/b/c.zip]//* for *zip:/[file:///a/b/c.zip]//foo/bar/baz*.

scheme()
Returns the scheme (from the outer url of this shallot.Eurl). This is what comes before the *://*. Example: *"file"* for *file:///foo/bar/baz*.

with_appended_segment(*segment*)
Returns a new shallot.Eurl from this one with "/basename" appended. The parameter is assumed to be a single path segment.

Parameters **segment** – The basename to be appended.

with_appended_segments(*path*)
Returns a new shallot.Eurl with path segments "/pa/t/h/" appended. The parameter may contain `"/"`s for dividing path segments.

Parameters **path** – The path segments to be appended, like *"foo/bar"*.

class shallot.**Exceptions**
Bases: object

Everything about exceptions.

exception ArgumentException(*details=None*, *message=None*, ***, *isResumeable=True*, *detailsAreInteresting=None*, *_class=''*)
Bases: *shallot.Exceptions.ProgramException*

Shallot exception for failed operation due to invalid arguments given to some program part. It typically allows resume but not retry (special cases may override each of them). Read more about Shallot Exceptions.

See shallot.Exceptions.ScriptedException.__init__ for details.

exception IOException(*details=None*, *message=None*, ***, *isRetryable=None*, *autoRetryRecommendedIn=- 1*, *detailsAreInteresting=None*, *_class=''*)
Bases: *shallot.Exceptions.RuntimeException*

Shallot exception in IO. It allows resume and typically allows retry (special cases may override each of them). Read more about Shallot Exceptions.

See shallot.Exceptions.ScriptedException.__init__ for details.

exception ProgramException(*details=None*, *message=None*, ***, *isResumeable=True*, *detailsAreInteresting=None*, *_class=''*)
Bases: *shallot.Exceptions.ScriptedException*

Shallot exception for program errors (typically logical stuff) in the program or plugin. It typically allows resume but no retry (special cases may override each of them). Read more about Shallot Exceptions.

See shallot.Exceptions.ScriptedException.__init__ for details.

exception RuntimeException(*details=None*, *message=None*, ***, *isRetryable=None*, *autoRetryRecommendedIn=- 1*, *detailsAreInteresting=None*, *_class=''*)

Bases: *shallot.Exceptions.ScriptedException*

Shallot exception for failed operation due to (often external) runtime effects. It allows resume and optionally allows retry (special cases may override each of them). Read more about Shallot Exceptions.

See shallot.Exceptions.ScriptedException.__init__ for details.

exception ScriptedException(*details=None*, *message=None*, ***, *isRuntime=None*, *isResumeable=None*, *isRetryable=None*, *autoRetryRecommendedIn=- 1*, *detailsAreInteresting=None*, *_class=''*)

Bases: `Exception`

The Shallot exception class.

Parameters

- **details** – Detail text.
- **message** – Message text.
- **isRuntime** – Is a runtime error (instead of a program logic error).
- **isResumeable** – Is resumeable.
- **isRetryable** – Is retryable.
- **detailsAreInteresting** – If the details contain information which is directly interesting for (and consumable by) the end user.
- **autoRetryRecommendedIn** – Recommended retry interval in milliseconds.
- **_class** – Only used internally.

isExceptionClass(*classname*)

Checks if this exception is instance of a given exception class.

Parameters **classname** – A exception class name (as string).

class shallot.**FilePropertyDialog**

Bases: `object`

The file property dialog.

class Tab(*title*, *properties*)

Bases: *shallot._ApiProxy*

Abstract base class for a tab in the Properties dialog. It provides content and can also offer user interactions. See shallot.FilePropertyDialog.Tab.register for registering custom implementations to shallot.

Parameters

- **title** – The title text for this tab.
- **properties** – A list of shallot.FilePropertyDialog.Tab.PropertyConfig instances. Each instances specifies one piece of information

you want to provide.

class PropertyButtonConfig(*label*, *fct*)

Bases: `object`

Definitions for buttons in a shallot.FilePropertyDialog.Tab.PropertyConfig.

Parameters

- **label** – The button text.

- **fct** – The function to execute when the user clicks on the button.

_idcounter = 0

class PropertyConfig(*title*, *propertytype*, *buttons=[]*)
Bases: object

Definitions for properties, which present a piece of information in a shallot.FilePropertyDialog.Tab.

Parameters
- **title** – The property title/name.
- **propertytype** – The value type. This controls what you have to return in shallot.FilePropertyDialog.Tab.update_widget and how it is displayed. See the PROPERTYTYPE_* constants.
- **buttons** – A list of shallot.FilePropertyDialog.Tab.PropertyButtonConfig specifying which buttons to show for offering user interaction.

class PropertyType
Bases: object

A type of a single property. Used in shallot.FilePropertyDialog.Tab.PropertyConfig.

_buttonTriggered(*k*)

_updateWidget(*i*, *operation*, *widgetptr*)

nodes()
Returns a list of shallot.Filesystem.Node containing the nodes to show (typically one).

refresh()
Reloads the content in the dialog (typically called after the external situation has changed so the dialog shows outdated information).

static register(*tabclass*, *positionGroup=None*, *positionIndex=None*)
Registers a shallot.FilePropertyDialog.Tab implementation for making its content available in the Properties dialog.

Parameters
- **positionGroup** – Controls display order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.FilePropertyDialog.Tab.
- **positionIndex** – Controls display order. See Position Indexes for details.
- **tabclass** – A subclass of shallot.FilePropertyDialog.Tab.

selected_index_for_property(*index*)
For a shallot.FilePropertyDialog.Tab.PropertyType.StringMap property, this returns the index of the row selected by the user in the dialog (or -1 for no selection). It is not allowed to call it for other properties.

update_widget(*i*, *operation*)
Provides the actual content. Returns a value depending on the type choice in shallot.FilePropertyDialog.Tab.PropertyConfig. *Override this method in custom subclasses.*

Parameters
- **i** – The index (in the same order as the *properties* constructor parameter).
- **operation** – A shallot.Operations.Operation instance.

class TabPropertyIconTextBanner
Bases: object

Represents values for image/text banners as used for shallot.FilePropertyDialog.Tab.PropertyType.IconTextBanner.

add_icon(*iconname*, *size=1.0*)
Adds an icon to the banner.

Parameters

- **iconname** – The name of the icon to append.
- **size** – Icon size (not in pixels but relative to the default).

add_text(*text*)
Adds text to the banner.
Parameters **text** – The text to append.

static _tabFactory(*tabclass*)

class shallot.**FileSearch**
Bases: object

File searches.

class SearchCriterion(*factory*)
Bases: *shallot._ApiProxy*

Abstract base class for a search criterion. It implements custom file search filters. Implement this class as well as shallot.FileSearch.SearchCriterionFactory and register this pair with shallot.FileSearch.SearchCriterionFactory.register.

Each implementation must offer a constructor with this signature. It must forward the *factory* parameter to this constructor.

_configure(*info*)

configure(*info*)
Asks some questions to the user (with *info* parameter) in order to determine a configuration and returns it as a list of strings. Some other methods must be able to interpret this list in order to get back this configuration. *Override this method in custom subclasses.*
Parameters **info** – A shallot.Actions.ExecutionInfo execution info object.

match(*operation*, *eurl*)
Determines if a file matches the configured search criteria (returns bool). *Override this method in custom subclasses.* Use searchspec for getting the current configuration.
Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl on which the filter logic must act.

searchspec()
Returns the current search configuration as string list.

class SearchCriterionFactory(*key*, *ctype*, *description*)
Bases: *shallot._ApiProxy*

A factory for a shallot.FileSearch.SearchCriterion class.

Call this constructor from subclasses.

Parameters

- **key** – A short string used as key for this criterion.
- **ctype** – The type of your shallot.FileSearch.SearchCriterion implementation to create.
- **description** – The short description (as shown in menus).

_construct()

get_searchspec_description(*val*)
Returns the description of a search configuration (for displaying in the user interface). *Override this method in custom subclasses or leave the default implementation.*
Parameters **val** – The search configuration as string list.

is_visible_for(*operation*, *node*)
Determines if this search criterion should be offered for a node. *Override this method in custom subclasses or leave the default implementation.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **node** – The shallot.Filesystem.Node node for which the visibility must be determined.

static register(*criterionfactory*, *positionGroup=None*, *positionIndex=None*)
Registers a custom shallot.FileSearch.SearchCriterionFactory instance for offering custom file searching filters.

Parameters
- **criterionfactory** – An instance of shallot.FileSearch.SearchCriterionFactory.
- **positionGroup** – Controls display order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.FileSearch.SearchCriterionFactory.
- **positionIndex** – Controls display order. See Position Indexes for details.

class shallot.**Filesystem**
Bases: object

The filesystem.

class Handler
Bases: *shallot._ApiProxy*

Abstract base class for a custom filesystem handler. Subclass and register it for implementing a new virtual filesystem, which lets new nodes appear somewhere in the filesystem tree and controls how to handle them. Use shallot.Filesystem.Handler.register for registration.

_configureItems(*op*, *itemlist*)

_getActions(*eurls*)

_getCustomAttributes(*op*, *eurl*)

_getFileContent(*op*, *eurl*)

_getMtime(*op*, *eurl*)

_setMtime(*op*, *eurl*, *itime*)

can_create_directory(*operation*, *eurl*)
Is it allowed to create subdirectories in a certain directory? *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

can_create_file(*operation*, *eurl*)
Is it allowed to create files in a certain directory? *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

can_create_link(*operation*, *eurl*)
Is it allowed to create a link in a certain directory? *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

can_delete_item(*operation*, *eurl*)
Is it allowed to delete a certain file/directory/link/...? *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

can_get_filecontent (*operation*, *eurl*)
Is it allowed to get the content of a certain file? *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

can_rename_item (*operation*, *src*)
Is it allowed to move a certain file/directory/link/...? *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **src** – The shallot.Eurl address pointing to the potential item to be moved.

configure_item (*operation*, *node*)
Configure a newly created node (e.g. setting another icon or changing the display name). *Override this method in custom subclasses or leave the default implementation.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **node** – The shallot.Filesystem.Node filesystem node to configure.

create_directory (*operation*, *eurl*)
Create a directory. *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

create_file (*operation*, *eurl*, *outstream*, *handlertransfer*)
Creates a file with some content. *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **outstream** – the content stream to be transferred into the new file
- **handlertransfer** – shallot.Operations.HandlerTransfer may be used for some better integration, like cancel support and progress monitoring.

create_link (*operation*, *eurl*, *tgt*)
Create a link. *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **tgt** – The link destination path (as string).

delete_item (*operation*, *eurl*)
Delete a file/directory/link/... *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_actions (*eurls*)
Which actions (see former example) are to be offered for certain files? *Override this method in custom subclasses.*

Parameters **eurls** – A list of shallot.Eurl addresses this call is referred to.

get_customattributes (*operation*, *eurl*)
Returns a dict<string,string> with custom attributes for a node. *Override this method in custom subclasses or leave the default implementation.* In contrast to extended attributes, those ones are not

directly stored in the filesystems, but are managed in a handler specific way (e.g. file permissions).

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_extendedattribute(*operation*, *eurl*, *attribname*)

Gets the value for a particular extended attribute. *Override this method in custom subclasses or leave the default implementation.* Extended attributes are all kinds of properties of a file which aren't handled otherwise. It can contain filesystem's extended attributes, permission information and more.

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **attribname** – The attribute name.

get_extendedattribute_size(*operation*, *eurl*, *attribname*)

Returns the size (in byte) of the value for a particular extended attribute. *Override this method in custom subclasses or leave the default implementation.* Extended attributes are all kinds of properties of a file which aren't handled otherwise. It can contain filesystem's extended attributes, permission information and more.

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **attribname** – The attribute name.

get_file_content(*operation*, *eurl*)

Get the content of a file as a shallot.Streaming.ReadDataDevice. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_linktarget(*operation*, *eurl*)

Resolve a link. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_mimetype(*operation*, *eurl*)

Determine mime type of a file. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_mtime(*operation*, *eurl*)

Determine the mtime of a file. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_size(*operation*, *eurl*)

Determine the size of a file. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

get_type(*operation*, *eurl*)

Determine if a file is regular, or a dir, or a link, ... *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

itemlist(*operation*, *eurl*, *nodetype*, *nodelist*)

Determine a list of subelements in a certain directory. *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **nodetype** – Which kind of items are requested. See shallot.Filesystem.NodeType.
- **nodelist** – A shallot.Filesystem.NodeList object which controls the entry list for the referred directory. Operate on this object for adding children.

list_extendedattributes(*operation*, *eurl*)

Returns a list of extended attributes which exist for a location. *Override this method in custom subclasses or leave the default implementation.* Extended attributes are all kinds of properties of a file which aren't handled otherwise. It can contain filesystem's extended attributes, permission information and more.

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.

static register(*scheme*, *handler*)

Registers a shallot.Filesystem.Handler filesystem handler implementation.

Parameters

- **scheme** – The scheme name (first part in a shallot.Eurl, like *file*).
- **handler** – The shallot.Filesystem.Handler filesystem handler.

remove_extendedattribute(*operation*, *eurl*, *attribname*)

Removes an extended attribute. *Override this method in custom subclasses or leave the default implementation.* Extended attributes are all kinds of properties of a file which aren't handled otherwise. It can contain filesystem's extended attributes, permission information and more.

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **attribname** – The attribute name.

rename_item(*operation*, *src*, *destpath*)

Move a file/directory/link/... to somewhere else (can be a simple renaming). *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **src** – The shallot.Eurl address pointing to the item to be moved.
- **destpath** – The new path for the src item.

set_customattribute(*operation*, *eurl*, *key*, *value*)

Sets a custom attribute to an entry. *Override this method in custom subclasses or leave the default implementation.* See also get_customattributes.

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **key** – The attribute key.
- **value** – The attribute value.

set_extendedattribute(*operation*, *eurl*, *attribname*, *value*)

Sets the value for a particular extended attribute. *Override this method in custom subclasses or leave the default implementation.* Extended attributes are all kinds of properties of a file which aren't handled otherwise. It can contain filesystem's extended attributes, permission information and more.

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **attribname** – The attribute name.
- **value** – The new attribute value (as byte string).

set_mtime(*operation*, *eurl*, *mtime*)
Set the mtime of a file. *Override this method in custom subclasses.*

Parameters
- **operation** – The shallot.Operations.Operation operation object.
- **eurl** – The shallot.Eurl address this call is referred to.
- **mtime** – The datetime.datetime, which specifies the new modification time.

class Node
Bases: object

A model representation of a location in the filesystem tree. It represents stuff like a file or a directory. Those instances are used at many places for all kinds of operations.

Please note: Instances can be associated with any kind of elements in the filesystem (files, directories, links, ...). It might also point to something which does not exist at all (see parentnode). The documentation sometimes explicitly makes a difference between those kinds (files, directories, links, ...; often called 'node type'). But it often uses the term 'file' implicitly while meaning all kinds of elements; assuming that e.g. a directory is just a special kind of a file. It should be clear from the particular context which meaning applies.

Can't be constructed directly.

add_detail(*column*)
Adds a detail to this node.

Parameters **column** – The shallot.DetailColumn instance to add.

eurl()
Return the shallot.Eurl entry address.

ishidden()
Returns if the node is hidden.

Note: It's not difficult for the user to also show the hidden items.

nodetype()
Returns the shallot.Filesystem.NodeType node type.

set_displayname(*displayname*)
Sets the displayed name of the node.

set_hidden(*value*)
Sets if the node is hidden.

See also ishidden().

Parameters **value** – If the node shall be visible (boolean).

set_icon(*icon*)
Sets the node icon.

class NodeList
Bases: object

A list editor for a filesystem node list. Thay are mainly used for specifying child nodes in shallot.Filesystem.Handler.itemlist and provide some help there. In most cases, you should just use shallot.Filesystem.NodeList.set_items.

Can't be constructed directly.

add_item(*item*)
Adds a new children to the list, directly showing that in the user interface, while you can proceed filling the list. Please read shallot.Filesystem.NodeList.begin_iterative_adding as well! In most cases, you don't need this function.
Parameters **item** – A the new children node. This must be the basename.

begin_iterative_adding()
Must be called before you begin to iteratively fill the children list (e.g. with shallot.Filesystem.NodeList.add_item). You must also call shallot.Filesystem.NodeList.end_iterative_adding afterwards.

end_iterative_adding()
Must be called after you iteratively filled the children list with shallot.Filesystem.NodeList.add_item. This automatically removes all the old nodes, which you haven't added in this session.

set_items(*items*)
Sets a new list content. This function is all you need in most situations. For iteratively adding nodes, which makes the intermediate results visible in the user interface, see shallot.Filesystem.NodeList.add_item.
Parameters **items** – A list of strings providing the new children nodes. The list must contain the basenames of all existing children (with the node type you got as argument).

class NodeType
Bases: object

Enumeration of types a shallot.Filesystem.Node can have.

static create_node(*eurl*, *handler*, *nodetype*, *parentnode*, *doinsert=True*, *showInitialLoadingLabel=True*, *hidden=False*)
Creates a new shallot.Filesystem.Node for placing it into the model. If such a node (with the same eurl for the same parent node) does not exist, it generates a new one. If there already is such a node alive, but currently not placed in the model, it recycles this one. This can happen when references exist to a node which is not yet inserted or which is removed meanwhile. It is not allowed to call this method when such a node already exists in the model. Use this function for getting a shallot.Filesystem.Node, which is to be added to the model now or later. It is typically used within a shallot.Filesystem.Handler implementation.

Parameters

- **eurl** – The shallot.Eurl of the new item.
- **handler** – The shallot.Filesystem.Handler of the new node. This must always be the same one as the handler registered for the scheme of the eurl!
- **nodetype** – The shallot.Filesystem.NodeType node type.
- **parentnode** – The shallot.Filesystem.Node parent node.
- **doinsert** – If this function shall actually add the node to the model (to be exact, in some situations, this will not actually take place nonetheless).
- **showInitialLoadingLabel** – If a 'loading…' label shall be visible at beginning.
- **hidden** – If the node shall be created as hidden one. See also shallot.Filesystem.Node.ishidden.

static find_nodes_for_eurl(*eurl*)
Get a list of shallot.Filesystem.Node for a shallot.Eurl. If it is unknown to the model so far, it tries to build them. In typical cases, this list either contains one element, or is empty if the filesystem handlers decide that this file does not exist. But in some cases, there is also more than one node for one shallot.Eurl (when

the Eurl appears on more than one place in the tree). It only returns nodes, which are 'alive', i.e. which have a living parent and which are a child of this parent.

static get_or_create_node(*eurl*, *handler*, *nodetype*, *parentnode*, *doinsert=True*, *showInitialLoadingLabel=True*, *hidden=False*)

Returns a shallot.Filesystem.Node for using it as a child node in the model. It either creates a new one, if there currently is no node for this eurl in this parentnode, or returns the existing one. Even for existing ones, this call can change the nodetype of that node.

Parameters

- **eurl** – The shallot.Eurl of the new node.
- **handler** – The shallot.Filesystem.Handler of the new node. This must always be the same one as the handler registered for the scheme of the eurl!
- **nodetype** – The shallot.Filesystem.NodeType node type.
- **parentnode** – The shallot.Filesystem.Node parent node.
- **doinsert** – If this function shall actually add the node to the model (to be exact, in some situations, this will not actually take place nonetheless).
- **showInitialLoadingLabel** – If a 'loading…' label shall be visible at beginning.
- **hidden** – If the node shall be created as hidden one. See also shallot.Filesystem.Node.ishidden.

static refresh(*eurl*, *forceFindParent=False*, *withDetails=True*)

Request to refresh the internal model information for a shallot.Eurl. This may be a place which is already known (then a change of some metadata or the removal is detected) or a formerly unknown place (then new nodes get inserted in the model).

Parameters

- **eurl** – The shallot.Eurl of the node.
- **forceFindParent** – Forcefully create such model information if unknown before.
- **withDetails** – If detail columns assigned to an existing node should also be updated.

static try_get_nodes_for_eurl(*eurl*)

Returns a list of shallot.Filesystem.Node for a shallot.Eurl. It only considers the current state of the in-memory model. It will only return nodes which are already known to the model so far. This operation is cheaper and handling only the known nodes is sufficient in many situations. In typical cases, this list either contains one element, or is empty. But in some cases, there is also more than one node for one shallot.Eurl (when the Eurl appears on more than one place in the tree). It only returns nodes, which are 'alive', i.e. which have a living parent and which are a child of this parent.

class shallot.**IntlStringMap**(***stringdicts*)

Bases: object

A multi-language string map used for localization of plugins.

Internally it uses shallot.IntlStringMap.IntlString instances, but is more convenient for dealing with a larger amount of strings.

Creates a map of internationalized strings. For each keyword-argument, this map will get a member with the key name as member name. The values are dictionaries in the same form as for shallot.IntlStringMap.IntlString. The members will be the best available language variants as plain Python strings.

Example: *Strings = shallot.IntlStringMap(Foo = {"en":"foo", "it":"Fuh"}, Bar = {"en":"bar", "it":"Barra"})*

Then *Strings.Foo* might contain the Python string "Fuh".

class IntlString(*strings*)
Bases: object

A multi-language string.

Creates a multi-language string by a dictionary, considering the keys as language code and the values as the string in that language.

Example: *shallot.IntlStringMap.IntlString({'en':'yes', 'de':'ja'})*

static _normalizecode(*code*)

get()
Returns the best variant as plain Python string, depending on the current user interface language.

map
A map of shallot.IntlStringMap.IntlString instances.

class shallot.**Logging**
Bases: object

Methods for logging.

static log_debug(*s*)
Logs a message with debug severity.

Parameters **s** – The message.

static log_error(*s*)
Logs a message with message severity.

Parameters **s** – The message.

static log_info(*s*)
Logs a message with info severity.

Parameters **s** – The message.

static log_warning(*s*)
Logs a message with warning severity.

Parameters **s** – The message.

class shallot.**MainWindow**
Bases: object

A Shallot main window.

static current()
Returns the current shallot.MainWindow.

get_current_directory_node()
Gets the shallot.Filesystem.Node selected in the current view.

jump_to_eurl(*eurl*)
Let the current view jump to another location.

Parameters **eurl** – The shallot.Eurl address to jump to.

static open_items(*eurls*)
Opens a list of files, largely as if the user had doubleclicked on them.

Parameters **eurls** – a list of shallot.Eurl.

class shallot.**Operations**
Bases: object

Everything about operations on the filesystem.

class FilesystemOperation

Bases: object

A high-level provider of filesystem operations. It is always based on the transaction of a shallot.Operations.Operation and also accessible from such an instance.

Some operations allow to specify an shallot.Operations.FilesystemOperationProgressMonitor. You can specify an instance for getting some additional functionaly, like progress notification, conflict resolution and more. Please note that not each operation uses each aspect of shallot.Operations.FilesystemOperationProgressMonitor (e.g. some will not do any conflict resolution).

Can't be constructed directly.

can_copy_item(*src*, *dest*)

Can we copy a given entry?

Parameters

- **src** – The shallot.Eurl of the item to be copied.
- **dest** – The shallot.Eurl destination path. If it is not known, use *None*.

can_create_directory(*eurl*)

Can we create a given directory?

Parameters **eurl** – The shallot.Eurl of the considered item.

can_create_file(*eurl*)

Can we create a given file?

Parameters **eurl** – The shallot.Eurl of the considered item.

can_create_link(*eurl*)

Can we create a given link?

Parameters **eurl** – The shallot.Eurl of the considered item.

can_delete_item(*eurl*)

Can we delete a given entry?

Parameters **eurl** – The shallot.Eurl of the considered item.

can_get_filecontent(*eurl*)

Can we get the file content for an entry?

Parameters **eurl** – The shallot.Eurl of the considered item.

can_move_item(*src*, *dest*)

Can we move a given entry?

Parameters

- **src** – The shallot.Eurl of the item to be moved.
- **dest** – The shallot.Eurl destination path. If it is not known, use *None*.

copy_item(*src*, *tgt*, *progressmon*)

Copy an entry.

Parameters

- **src** – The shallot.Eurl of the item to be copied.
- **tgt** – The shallot.Eurl of the new destination.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

copy_items(*src*, *tgt*, *progressmon*)

Copy entries.

Parameters

- **src** – List of shallot.Eurl of the item to be copied.

- **tgt** – The shallot.Eurl of the new common parent destination.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

create_directory(*eurl*, *progressmon*)

Creates a directory.

Parameters

- **eurl** – The shallot.Eurl of the new directory.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

create_file(*eurl*, *contentdevice*, *progressmon*)

Create a file.

Parameters

- **eurl** – The shallot.Eurl of the new file.
- **contentdevice** – a shallot.Streaming.ReadDataDevice content device.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

create_link(*eurl*, *tgt*, *progressmon*)

Creates a link.

Parameters

- **eurl** – The shallot.Eurl of the new link.
- **tgt** – Target path of this link.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

delete_directory_if_empty(*eurl*, *progressmon*)

Delete a directory entry if empty.

Parameters

- **eurl** – The shallot.Eurl of the directory to delete.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

delete_item(*eurl*, *progressmon*)

Delete an entry.

Parameters

- **eurl** – The shallot.Eurl of the item to delete.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

get_extendedattribute(*eurl*, *attribute*)

Returns the value (as byte string) of an extended attribute value stored in the filesystem for an entry.

Parameters

- **eurl** – The shallot.Eurl of the considered item.
- **attribute** – The attribute key.

get_extendedattribute_size(*eurl*, *attribute*)

Returns the size of an extended attribute value stored in the filesystem for an entry.

Parameters

- **eurl** – The shallot.Eurl of the considered item.

- **attribute** – The attribute key.

get_file_content(*eurl*)
Gets the file content for an entry as shallot.Streaming.ReadDataDevice.
Parameters **eurl** – The shallot.Eurl of the considered item.

get_filesize(*eurl*)
Gets the file size in bytes for an entry.
Parameters **eurl** – The shallot.Eurl of the considered item.

get_linktarget(*eurl*)
Gets the link target for an entry (if it is a link).
Parameters **eurl** – The shallot.Eurl of the considered item.

get_type(*eurl*)
Gets the shallot.Filesystem.NodeType node type for an entry.
Parameters **eurl** – The shallot.Eurl of the considered item.

itemlist(*eurl*)
Gets a list of shallot.Eurl in a directory.
Parameters **eurl** – The shallot.Eurl of the directory to list.

itemlist_by_type(*eurl*, *nodetype*)
Gets a list of shallot.Eurl in a directory which have a certain type.
Parameters
- **eurl** – The shallot.Eurl of the directory to list.
- **nodetype** – The shallot.Filesystem.NodeType node type to fetch.

list_extendedattributes(*eurl*)
Lists all keys of extended attributes stored in the filesystem for an entry.
Parameters **eurl** – The shallot.Eurl of the considered item.

move_item(*src*, *tgt*, *progressmon*)
Move an entry.
Parameters
- **src** – The shallot.Eurl of the item to be moved.
- **tgt** – The shallot.Eurl of the new destination.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

move_items(*src*, *tgt*, *progressmon*)
Move entries.
Parameters
- **src** – List of shallot.Eurl of the item to be moved.
- **tgt** – The shallot.Eurl of the new common parent destination.
- **progressmon** – A shallot.Operations.FilesystemOperationProgressMonitor instance for some additional functionality (or *None*). See shallot.Operations.FilesystemOperation for details.

remove_extendedattribute(*eurl*, *attribute*)
Removes an extended attribute stored in the filesystem for an entry.
Parameters
- **eurl** – The shallot.Eurl of the considered item.
- **attribute** – The attribute key.

resolve_eurl_link(*eurl*)
Resolve a link as Eurl.
Parameters **eurl** – A shallot.Eurl to resolve.

resolve_eurl_link_nonrecursive(*eurl*)
Resolve a link as Eurl without recursion.
Parameters **eurl** – A shallot.Eurl to resolve.

resolve_node_link(*node*)
Resolve a link as node.
Parameters **node** – A shallot.Filesystem.Node to resolve.

resolve_node_link_nonrecursive(*node*)
Resolve a link as node without recursion.
Parameters **node** – A shallot.Filesystem.Node to resolve.

set_extendedattribute(*eurl*, *attribute*, *value*)
Store the value of an extended attribute in the filesystem for an entry.
Parameters
- **eurl** – The shallot.Eurl of the considered item.
- **attribute** – The attribute key.
- **value** – The attribute value (as byte string).

class FilesystemOperationProgressMonitor(*actionExecution=None*)
Bases: *shallot._ApiProxy*

Abstract base class for progress monitors, used for monitoring progress of some operations in shallot.Operations.FilesystemOperation and for conflict resolution.

It makes sense to directly instantiate this class in some cases.

Parameters **actionExecution** – Optional instance of shallot.Actions.ExecutionInfo. If available, it provides some additional features

(e.g. the user may cancel the transfer).

_resolveConflicts(*steps*)

all_bytes()
How many bytes are to be transferred in total.

all_items()
How many items are to be transferred in total.

changed()
Reacts on progress changes. *Override this method in custom subclasses or leave the default implementation.*

done_bytes()
How many bytes are transferred so far.

done_items()
How many items are transferred so far.

estimation()
The current performance estimation as textual representation.

get_item_info_from()
The current source as textual representation.

get_item_info_to()
The current destination as textual representation.

has_bytes_info()
If the monitor currently has information about progress in terms of transferred bytes.

has_item_info()
If the monitor currently has information about progress in terms of item counts.

resolve_conflicts(*steps*)
Resolves upcoming filesystem conflicts. *Override this method in custom subclasses or leave the default implementation.*

Parameters steps – A list of shallot.Operations.FilesystemOperationStep which stay in conflict.

class FilesystemOperationStep
Bases: object

One step in a larger filesystem transfer running inside some operations in shallot.Operations.FilesystemOperation.

Can't be constructed directly.

class ConflictResolution
Bases: object

Enumeration of conflict resolution strategies.

conflict_description()
The conflict descrition text.

conflict_resolution()
The currently selected shallot.Operations.FilesystemOperationStep.ConflictResolution conflict resolution.

conflict_resolution_different_destination_name_to()
The new destination name (if conflict_resolution is shallot.Operations.FilesystemOperationStep.ConflictResolution.Usel

conflict_resolution_rename_destination_before_to()
The new destination name (if conflict_resolution is shallot.Operations.FilesystemOperationStep.ConflictResolution.Rena

effective_destination()
The effective destination shallot.Eurl location with applied conflict resolution strategies.

original_destination()
The original destination shallot.Eurl location.

set_conflict_resolve_merge_directories()
Set the conflict resolution to shallot.Operations.FilesystemOperationStep.ConflictResolution.MergeDirectories.

set_conflict_resolve_merge_directories_check_allowed()
Checks if it is allowed to solve this conflict by merging directories.

set_conflict_resolve_overwrite_destination()
Set the conflict resolution to shallot.Operations.FilesystemOperationStep.ConflictResolution.OverwriteDestination.

set_conflict_resolve_rename_destination_before(*newname*)
Set the conflict resolution to shallot.Operations.FilesystemOperationStep.ConflictResolution.RenameDestinationBefore

set_conflict_resolve_skip()
Set the conflict resolution to shallot.Operations.FilesystemOperationStep.ConflictResolution.Skip.

set_conflict_resolve_use_different_destination_name(*newname*)
Set the conflict resolution to shallot.Operations.FilesystemOperationStep.ConflictResolution.UseDifferentDestinationNa

source()
The source shallot.Eurl location.

sourcetype()
The shallot.Filesystem.NodeType of the source.

class HandlerTransfer
Bases: object

Used for some additional functionality in some transfer operations in shallot.Filesystem.Handler instances.

Can't be constructed directly.

increment_transferred_bytes(*donebytes*)
Notify that a certain amount of date (in byte) are transferred. This is used for progress monitoring.

respect_cancel()
Called from time to time for allowing the user to cancel the transfer.

class Operation
Bases: object

Transactional read and write filesystem accesses. Read more.

abort()
Aborts transaction dropping all pending changes.

commit()
Commits transaction applying all pending changes.

enable_detailscache()
Enable details caching.

fetch_container_file(*eurl*)
Fetches the container file for a eurl locally and returns the local path. Use this with care and only if needed. In many cases working with streams is the better way! It returns nothing for an eurl without embedded parts (like foo://[bar:///y]/x). If the outermost inner eurl represents a local file path, it is directly returned without copying. You should always regard the available disk space. See get_free_cachespace().
Parameters **eurl** – The shallot.Eurl location to fetch the container for.

fetch_container_file2(*eurl*, *namehint*)
Fetches the container file for a eurl locally and returns the local path. This is like fetch_container_file() but with more options.
Parameters
- **eurl** – The shallot.Eurl location to fetch the container for.
- **namehint** – A small file name prefix for temporary files, like *"tmp"*.

fetch_file(*eurl*)
Fetches a file locally and returns the local path. Use this with care and only if needed. In many cases working with streams is the better way! If the eurl represents a local file path, it is directly returned without copying. You should always regard the available disk space. See get_free_cachespace().
Parameters **eurl** – The shallot.Eurl location to fetch the container for.

fetch_file2(*eurl*, *namehint*)
Fetches a file locally and returns the local path. This is like fetch_file() but with more options.
Parameters
- **eurl** – The shallot.Eurl location to fetch the container for.
- **namehint** – A small file name prefix for temporary files, like *"tmp"*.

filesystem()
Gets the shallot.Operations.FilesystemOperation filesystem operation object.

get_custom_data(*eurl*, *key*)
Gets stored custom data.
Parameters
- **eurl** – The shallot.Eurl instance you want to get data about (or None).

- **key** – Custom data key name.

get_detail_from_cache(*eurl*, *column*)
Gets a column value from cache.
Parameters
- **eurl** – The shallot.Eurl to get a detail value for.
- **column** – The shallot.DetailColumn to get a detail value for.

get_free_cachespace()
Returns the free disk space (in bytes) available for operations like fetch_container_file() or fetch_file().

is_detailscache_enabled()
Is details caching enabled?

set_custom_data(*eurl*, *key*, *value*)
Stores custom data.
Parameters
- **eurl** – The shallot.Eurl instance you want to store data about (or None).
- **key** – Custom data key name.
- **value** – Custom data value.

store_detail_in_cache(*eurl*, *column*, *value*)
Stores a column detail in cache.
Parameters
- **eurl** – The shallot.Eurl to store a detail value for.
- **column** – The shallot.DetailColumn to store a detail value for.
- **value** – The column value for this entry.

static create()
Creates a new shallot.Operations.Operation.

class shallot.**PanelDetails**
Bases: object

Everything about panel details.

class AbstractPanelDetailValueElement(*type*, *value*)
Bases: object

Abstract base class for an element of a panel detail row's value. This can be something like a text, an icon or a link button. See the subclasses.

class MultiSelectionPanelDetail(*positionGroup=None*, *positionIndex=None*, *valueWidthHint=4*)
Bases: *shallot._ApiProxy*, *shallot.PanelDetails.PanelDetail*

Abstract base class for a detail panel entry, which occurs when multiple items are selected. See shallot.PanelDetails.PanelDetail.register for registering custom implementations to shallot.

Parameters

- **positionGroup** – Controls display order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.PanelDetails.PanelDetail.
- **positionIndex** – Controls display order. See Position Indexes for details.
- **valueWidthHint** – A width in centimeters to reserve for printing the values.

link_triggered(*nodes*, *linktarget*)
This method is called whenever the user triggers a link. Override in custom implementations if links are used.
Parameters
- **nodes** – The selected list of shallot.Filesystem.Node object.

- **linktarget** – The link target string, as specified in shallot.PanelDetails.PanelDetailValueElementButton.__init__.

set_value(*detail*, *nodes*, *operation*)
This method is called whenever an output must be determined for certain nodes. It must return a list of 2-tuples (one for each row). Those are each name/value pairs with the label of that row and the value (which is a list of shallot.PanelDetails.AbstractPanelDetailValueElement). For large waiting times, you should just return an empty value list for your rows, start an asynchronous execution and use shallot.PanelDetailInst.set_row in the end. *Override this method in custom subclasses.*

Parameters
- **detail** – A shallot.PanelDetailInst object for printing the details. Only used in advanced cases.
- **nodes** – The selected list of shallot.Filesystem.Node object.
- **operation** – The shallot.Operations.Operation operation object.

class PanelDetail
Bases: `object`

Abstract base class for a detail panel entry. See the subclasses.

_setValue(*detail*, *nodes*, *op*)

static register(*detail*)
Registers a shallot.PanelDetails.PanelDetail in the Shallot core.
Parameters **detail** – The shallot.PanelDetails.PanelDetail instance.

class PanelDetailInst(*_detail*)
Bases: `object`

The storage for detail value rows, which are the actual content of a shallot.PanelDetails.PanelDetail for an actual file selection.

Can't be constructed directly.

set_row(*row*, *val*)
Sets the value for one row.
Parameters
- **row** – The label of that row.
- **val** – The value (which is a list of shallot.PanelDetails.AbstractPanelDetailValueElement).

class PanelDetailValueElementButton(*text*, *target*, *autohide=True*)
Bases: *shallot.PanelDetails.AbstractPanelDetailValueElement*

A link button (for executing some code on user behalf) as element of a panel detail row's value. Use shallot.PanelDetails.SingleSelectionPanelDetail.link_triggered or shallot.PanelDetails.MultiSelectionPanelDetail.link_triggered for handling user actions.

Parameters
- **text** – The text.
- **target** – The link target name (will be passed to the handler function).
- **autohide** – If the button shall only be visible when the mouse cursor is in the details panel.

class PanelDetailValueElementIcon(*icon*)
Bases: *shallot.PanelDetails.AbstractPanelDetailValueElement*

An icon as element of a panel detail row's value.

Parameters **icon** – The icon name.

class PanelDetailValueElementString(*text*)
Bases: *shallot.PanelDetails.AbstractPanelDetailValueElement*

A piece of text as element of a panel detail row's value.

Parameters **text** – The text.

class PanelDetailValueElementWaiting
Bases: *shallot.PanelDetails.AbstractPanelDetailValueElement*

A 'waiting' placeholder as element of a panel detail row's value.

class SingleSelectionPanelDetail(*positionGroup=None*, *positionIndex=None*, *valueWidthHint=4*)
Bases: *shallot._ApiProxy*, *shallot.PanelDetails.PanelDetail*

Abstract base class for a detail panel entry, which occurs when one item is selected. See shallot.PanelDetails.PanelDetail.register for registering custom implementations to shallot.

Parameters

- **positionGroup** – Controls display order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.PanelDetails.PanelDetail.
- **positionIndex** – Controls display order. See Position Indexes for details.
- **valueWidthHint** – A width in centimeters to reserve for printing the values.

link_triggered(*node*, *linktarget*)
This method is called whenever the user triggers a link. Override in custom implementations if links are used.

Parameters
- **node** – The selected shallot.Filesystem.Node.
- **linktarget** – The link target string, as specified in shallot.PanelDetails.PanelDetailValueElementButton.__init__.

set_value(*detail*, *node*, *operation*)
This method is called whenever an output must be determined for a certain node. It must return a list of 2-tuples (one for each row). Those are each name/value pairs with the label of that row and the value (which is a list of shallot.PanelDetails.PanelDetails.AbstractPanelDetailValueElement). For large waiting times, you should just return a value list only containing a shallot.PanelDetails.PanelDetailValueElementWaiting for your rows, start an asynchronous execution and use shallot.PanelDetailInst.set_row in the end. *Override this method in custom subclasses.*

Parameters
- **detail** – A shallot.PanelDetailInst object for printing the details. Only used in advanced cases.
- **node** – The selected shallot.Filesystem.Node.
- **operation** – The shallot.Operations.Operation operation object.

class shallot.**Setting**(*name*, *description*, *group*, *isAdvancedSetting=False*, *isGlobal=False*, *isPerFileview=False*)
Bases: *shallot._ApiProxy*

Abstract base class for a scripted Shallot setting (those who have to be stored manually). See Shallot documentation for details.

Parameters

- **name** – Internal name (must be unique).
- **description** – Description text.
- **group** – Group. One of shallot.Setting.GroupInfo.

- **isAdvancedSetting** – Is this an advanced setting?
- **isGlobal** – Does this setting regard global aspects (instead of per-directory aspects)?
- **isPerFileview** – Does this setting regard per-fileview aspects (instead of per-mainwindow)?

class GroupInfo
Bases: object

Enumeration of groups to which a setting can belong. This is just a matter of grouping the for presentation.

_setvalue1(*value*)

_setvalue2(*viewindex*, *value*)

get_value(*viewindex*)
Get the currently set value. *Override this method in custom subclasses.*

Parameters **viewindex** – View index. Will be *Nothing* for per-mainwindow settings. Otherwise a fileview index.

static register(*setting*)
Registers a shallot.Setting instance in Shallot.

Parameters **setting** – The shallot.Setting setting object.

set_value(*value*, *viewindex=None*)
Called from Shallot core when the value was set. *Override this method in custom subclasses.*

Parameters

- **value** – The value.
- **viewindex** – View index. Will be *Nothing* for per-mainwindow settings. Otherwise a fileview index.

value_description(*val*)
Gets a human readable description text for a value. *Override this method in custom subclasses.*

Parameters **val** – The value.

class shallot.**Streaming**
Bases: object

Streams of binary data.

class BlobReadDataDevice(*content*)
Bases: *shallot.Streaming.ReadDataDevice*

A binary source backed by static binary content in a byte array.

For really large content (many megabytes), you should avoid creating complete copies but use one of the other subclasses or implementing an own one.

Parameters **content** – The byte array containing the content.

getdata()
See shallot.Streaming.ReadDataDevice.

class ReadDataDevice
Bases: *shallot._ApiProxy*

A scripted source of binary content. Override this class for providing own binary content. You might often find it more convenient to use one of the non-abstract subclasses.

_getdata()

getdata()
Returns a Python byte array containing the next available chunk of content data. This may either be the complete content, or just the next part in an arbitrary size which is convenient in your situation. It is allowed to return empty arrays whenever there is temporarily no data available. Return *None* when the end of stream is reached. *Override this method in custom subclasses.*

read()
Reads a chuck of data as Python byte array. An empty array signals the stream ended.

readall()
Reads the complete data as Python byte array.

class StreamReadDataDevice(*stream*)
Bases: *shallot.Streaming.ReadDataDevice*

A binary source backed by a file object.

The file object must be opened in blocking mode (which is the typical one).

Parameters **stream** – The file object containing the content.

getdata()
See shallot.Streaming.ReadDataDevice.

class ThreadedReadDataDevice
Bases: *shallot.Streaming.ReadDataDevice*

A binary source dynamically created piecewise in a separate thread. Override this class and provide the content generator within it.

_stop()
This is part of a particular piece of internal infrastructure and is typically not required to be used directly.

getdata()
See shallot.Streaming.ReadDataDevice.

run()
This is the content generator function. *Override this method in custom subclasses.* Generate the content in an arbitrary way and call write() once or iteratively until the complete content is written. This method will automatically be executed in a separate thread.

write(*content*)
Appends a piece of data. Call this function from within the run() method.
Parameters **content** – The next piece of content as byte array.

class shallot.**ThumbnailProvider**
Bases: *shallot._ApiProxy*

Abstract base class for a thumbnail. It can implement new ways of generating thumbnail images for files. Implement this class and register an instance with shallot.ThumbnailProvider.register.

get_thumbnail(*operation*, *node*, *contenttype*, *width*, *height*)
Returns the thumbnail. It must be a bytestring containing the image in a well known format (png, jpeg, svg, ...). *Override this method in custom subclasses.*

Parameters

- **operation** – The shallot.Operations.Operation operation object.
- **node** – The shallot.Filesystem.Node filesystem node to get a thumbnail for.
- **contenttype** – The content type of the original file.

- **width** – The requested thumbnail width in pixels.
- **height** – The requested thumbnail height in pixels.

static register(*thumbnailprovider*, *positionGroup=None*, *positionIndex=None*)
Registers a custom shallot.ThumbnailProvider instance for offering custom thumbnails.

Parameters

- **thumbnailprovider** – An instance of shallot.ThumbnailProvider.
- **positionGroup** – Controls execution order. See Position Indexes for details. Use one of the *INDEX_** values from shallot.ThumbnailProvider.
- **positionIndex** – Controls execution order. See Position Indexes for details.

class shallot.**_ApiProxy**(*native*)
Bases: object

Common superclass for many classes in this interface.

This class mostly has internal meaning and does not directly offer any services to the plugin developer. Read h_apiproxy.

_addmethod(*methodsetter*, *methodname*)

_native()

shallot.**_abstract**()

shallot.**_computeindex**(*defaultcategory*, *category*, *clss*, *index*)

shallot.**_isimplemented**(*fct*)

shallot.**_nonreal**()

shallot.**_optional**(*fct*)

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W