

The PureDocs Plugin Architecture

Will Barton

March 13, 2007

Contents

1	General Overview	1
2	Concepts	2
2.1	Resources	2
2.2	Plugin Registration	2
2.3	Options	3
3	The Python Plugin API	3
3.1	The <code>puredocs.Resource</code> Class	3
3.2	Plugin Registration	4
3.3	Language Plugins	5
3.4	Output Plugins	5
3.5	Options	6
4	The C Plugin API	6
4.1	The Resource Type	6
4.2	Language Plugins	7
4.3	Output Plugins	7
4.4	Options	7
5	XSLT Stylesheets	7

Abstract

The PureDocs API documentation system is designed as a modular system to support languages and differing output formats. This document describes this plugin architecture and provides examples to assist in developing new plugins for both languages and output formats for the PureDocs API documentation system.

1 General Overview

Within PureDocs, there are two types of plugins, Language Plugins, and Output Plugins. Each type of plugin has a specific interface that is defined for it, and all plugins of that type are expected to conform to that interface. These interfaces are detailed below.

PureDocs also defines a tree-like structure in the form of Resources that plugins either input or output. A Resource object represents a node that will have properties, documentation, child resources, and importantly, a type that is specific to the language that resource belongs to. Resources are also described in greater detail below.

Language plugins provide PureDocs with a parser for a given language that will parse a source file of that language, and output a tree of Resource objects representing the content of that source file. The language plugin defines properties within resources, as well as resource types.

Output plugins take a resource tree generated by a language plugin, and generates the final output form of the resource tree. This includes parsing documentation that may be attached to a resource. The result of the output plugin's processing is the final output PureDocs provides. It is generally expected to be XML, but PureDocs makes no assumptions about the output format.

2 Concepts

PureDocs currently supports plugins written both in the Python language and the C language. While implementation of plugins in either language is different, the concepts involved are identical.

2.1 Resources

PureDocs provides a single type that is shared by both types of plugins, the Resource. A Resource is a tree of items within a source file that are to be included in the final output, in one form or another. The file itself is the top-level Resource, and all items within the file (functions, classes, etc) are children of that file Resource.

Language Plugins create Resource objects, give them a `type` (e.g. `function`), set `properties` on the resource (e.g. `parameters` for a `function`), set the `doc` string on the resource, and finally, add `child resources` to the resource.

Output Plugins take a tree of Resource objects that were created by a language plugin, and process that tree to create the final output format. This includes parsing the `doc` string in the plugin's preferred way.

2.2 Plugin Registration

PureDocs plugins register for certain "entry points" based on the type of plugin they are providing and the type of data they operate on. The declaration of these "entry points" differ between Python and C plugins, however they are conceptually identical.

Plugins register with either the `puredocs.lang` or `puredocs.out` entry points. The entry point registration maps a string (specific to the plugin type) to the plugin. What this mapping means is specific to the language the plugin is written in.

For language plugins, the string is the MIME-type of the file that the language plugin supports. For example, the reference Python language plugin maps the `python_lang` module to the MIME-type `text/x-python`. When PureDocs is given an input file

that matches the MIME-type the language plugin is mapped to, that language plugin is used.

For output plugins, the string is a short description of the output plugin's output type, that the user specifies with the `-f` or `--output-format` options to the **pure-docs** command. The Re-Structured Text XML output plugin uses the `rstxml` string.

2.3 Options

PureDocs also allows for any plugin to extend the command-line arguments the **pure-docs** tool takes. Plugins can declare their own options, using the provided PureDocs API. Any options that conflict with PureDocs options will be ignored, however.

3 The Python Plugin API

The Python Plugin API is provided entirely by the `puredocs` Python package.

3.1 The `puredocs.Resource` Class

`__init__` **typedocpropertiesresources** Creating a `Resource` is straight-forward.

The `__init__` method takes all the parameters described above.

PARAMETERS

type This is decided upon by the language plugin, and the only requirement is that a language plugin be consistent with itself. An example would be `function` for a resource representing a function.

doc The documentation string associated with the resource. It could be extracted from a comment around the item in the source file, or another convention used by the language.

properties This is a simple dictionary containing key/value pairs of property names and values for this `Resource`. An example would be a key of `parameters`, and a list of the parameters for a function resource.

resources This is a simple list of resources that are children of this resource. An example would be function or method resources that are children of a class resource.

An example of using the Python API for creating a `Resource` is creating a `Resource` for a Python language function. This example is loosely based on the Python language plugin.

Example 3.1 Creating a Resource

```
def visitFunction(self, node):
    func = Resource('function',
                    doc = node.doc,
                    properties = {'name':node.name,
                                  'decorators':node. ←
                                  decorators,
                                  'parameters':node. ←
                                  argnames ←
                                  })
```

In addition to Resource creation, Resource objects provide a Python sequence API for accessing child Resource objects. Please see the [Python sequence API documentation](#) for more information on how child Resource objects can be accessed.

Additionally, properties can be accessed as Resource object attributes by property name.

3.2 Plugin Registration

To support loading plugins, PureDocs makes use of the Python package [setuptools](#). setuptools allows for the loading of Python packages ("eggs") that register for certain "entry points". These are declared when the Python package is built and installed, and allows for independence of naming of Python packages, as well as the ability to integrate PureDocs plugins with other Python packages.

For Python plugins, an entry point is a mapping of a string to a python module, class, or function. PureDocs expects both entry points to map to a Python module or class that provides specific members that are detailed below in the Language and Output plugin descriptions.

Entry points are specified in a standard Python `setup.py` by importing the `setup` module, and using the `entry_points` dictionary argument to the `setup()`. The keys for the dictionary are the entry points, and the values are a list of the mappings detailed above.

Example 3.2 setup.py Entry Point Example

```
from setuptools import setup
setup(
    # ... Standard Arguments and Metadata...
    entry_points = {
        'puredocs.lang': ['text/x-python = puredocs. ←
                          plugins.python_lang', ],
    }
)
```

The [setuptools](#) webpage provides more information on setuptools and entry points, as well as more examples.

3.3 Language Plugins

The language plugin is responsible for parsing a provided file and returning a tree of `Resource` objects. The language plugin is responsible for defining the types of resource objects, deciding what gets included in the resource object tree, and attaching documentation strings to the resources from whatever convention may be used in the language.

Language plugins *must* provide the following members, whether the plugin is a module or a class.

parser A class that can parse a given file and return a `Resource` object tree. The nature of the parser class is described below.

options A class providing any command-line options the parser may wish to take to influence the way a file is parser. The nature of the options class is described below.

The language plugin's `parser` interface requirements are very simple. The interface for language parser classes follows.

THE LANGUAGE PARSER

root The `Resource` object forming the base of the `Resource` tree.

__init__ filename Parse the file with the given filename as this language, generating a tree of `Resource` objects accessible from this object's root variable.

Beyond that, there are no requirements of the language parser. `PureDocs` calls the `__init__` method, and then attempts to pass the `root` variable to the output plugin. The `root` variable could be populated by the `__init__`, or it could be dynamic.

3.4 Output Plugins

The output plugin is responsible for taking a `Resource` tree created by a language plugin and transforming it into the final output form that will be given to the user by `PureDocs`. The output plugin is also responsible for any processing that may be necessary of a `Resource`'s doc string.

Like the language plugin, the output plugin *must* provide the following members, whether the plugin is a module or a class.

writer A class that can process a `Resource` object tree and generate the final output format for the documentation. The nature of the writer class is described below.

options A class providing any command-line options the writer may wish to take to influence the way the output is written. The nature of the options class is described below.

The output writer class is required to implement the following methods:

THE OUTPUT WRITER

__init__resource Output a given resource and all of its sub-resources, processing any aspect of the resource that might be necessary, especially its documentation.

string Return a string representation of the final format of the output.

writedestination Write the final format of the output to the given destination.

If the destination is a directory, rather than a filename, generate a filename based on the name of the resource, if available, and place it in that directory.

3.5 Options

Command-line arguments can be accepted by any plugin, independent of PureDocs. These arguments are specified by the plugin's `options` object, described above. This should be a subclass of the `puredocs.Options` class, which is imported from the `twisted.python.usage` package.

The options that are defined in a plugin's `options` class are secondary to the standard PureDocs options, and should not conflict. An example of an options class follows.

For more information about Options classes, see [the Twisted Project's documentation](#).

4 The C Plugin API

The C Plugin API is provided by the `libpuredocs` library, and exposed via the `puredocs.h` header file.

4.1 The Resource Type

Resource * puredocs_createResource (char * type); Create and return a new Resource object with the given type. This is similar to the Python `__init__` method described above.

PARAMETERS

type This is decided upon by the language plugin, and the only requirement is that a language plugin be consistent with itself. An example would be `function` for a resource representing a function.

int puredocs_setDocString (Resource * resource, char * doc); Set the documentation string for the given Resource. The language plugin is responsible for stripping any extra characters (language-specific comment decorations, for example).

PARAMETERS

resource The Resource to add the doc string to.

doc The documentation string associated with the resource. It could be extracted from a comment around the item in the source file, or another convention used by the language.

int puredocs_setProperty (Resource * resource, char * property, char * value); Set the given `property` to the given `value` on the given Resource. If the Resource does not already have the `property`, it will be added.

Every Resource has properties, which are key/value pairs. An example would be a key of paramters, and a list of the parameters for a function resource.

PARAMETERS

resource The Resource to add the doc string to.

property The key for the `value` given.

value The value for the `property` given.

int puredocs_addChild (Resource * resource, Resource * child); Add the `child` Resource to the given Resource. An example of a child Resource would be a function within a module.

PARAMETERS

resource The Resource to add the doc string to.

child The Resource to add as the child.

4.2 Language Plugins

The C Language Plugin API is still being finalized.

4.3 Output Plugins

The C Output Plugin API is still being finalized.

4.4 Options

The C Options API is still being finalized.

5 XSLT Stylesheets

Because XML output is common to PureDocs, XSLT is a natural extension of PureDocs, and indeed, PureDocs ships with a collection of stylesheets for its default output plugin. Generally language plugin authors want to be aware of this, and at some point in the future, a rigid definition of the PureDocs stylesheets, as well as language-specific customizations, will be available. It would be in the best interest of language plugin authors to provide XSLT stylesheets for their language plugin's specific resource types, properties, etc.