Referencer Plugin Writing Guide V1.1.6

abstract:

Referencer is a tool to help you manage your collection of documents, and to use them to generate a bibliography. This guide describes the plugin API of Referencer.

publisher:
Referencer Development Team

authorgroup:
John Spray

revhistory:

releaseinfo:

legalnotice:

To report a bug or make a suggestion regarding the Referencer application or this manual, please visit the Referencer home page.

Introduction

The Referencer plugin API provides a relatively painless way of adding functionality. Plugins can either add user interface entries to menus and toolbars (action plugins), or provide hooks for downloading metadata for documents (metadata plugins).

At the time of writing, the plugin interface is new and not widely tested. Although it is not gratuitously broken there may be issues. You can ask at the mailing list.

This document is written for Referencer v1.1.6.

Installing Plugins

Plugins are python scripts. They don't have to be executable, but they do have to have a filename ending .py. At startup, Referencer searches several locations for plugins: ./plugins, $PREFIX/lib/referencer, and ~/.referencer/plugins.

Common API

Plugin info dictionary

All plugins must contain a dictionary at file scope capped `referencer_plugin_info`. This provides a number of fields describing the plugin. All plugins must provide the following fields, further optional fields are listed in following sections.

Table 1. Plugins Plugin Name Description author Author's name, or comma separated list. eg. "Bob Dole" version A version string for the plugin. eg. "1.3.2" longname The plugin's purpose. eg. "Format author initials"

The plugin info structure might be defined like this:

```
referencer_plugin_info  =
  {
  "author":        "John  Spray",
  "version":      "1.1.2",
"longname":  _("Generate  keys  from  metadata")
}
```

Utility functions

Import the module `referencer` to access the following functions:

referencer.download(short desc, long desc, url)

Web download convenience function. Returns a string containing the contents of the downloaded file, or an empty string on errors. Example:

referencer.download("Downloading tripe", "Downloading page from slashdot.org", "http://www.slashdot.org")

As well as being convenient, this function inherits the user's gnome-vfs proxy settings so is in general preferable where more complex http functionality is not required. In any error case, the function returns an empty string.

referencer._(text)

Translation function. You probably want to do "from referencer import _" in order to support localisation of user-visible strings. Any user-visible string should be expressed as _("Some text")

referencer.pref_get(key)

Load a persistent configuration string. If the key is not found an empty string is returned. To avoid conflicting with other plugins, each plugin should use key names prefixed with the name of the plugin. These configuration items are stored in the GConf database along with Referencer's native configuration.

referencer.pref_set(key, value)

Set a persistent configuration string.

document class

Referencer exposes individual documents with the document class. This supports a limited number of getter/setter methods:

get_field(key)

Retrieve a (case-insensitive) field such as "author". Builtin fields are doi, title, volume, number, journal, author, year and pages. Other arbitrarily named fields may or may not exist for a document. Getting a non-existent field returns an empty string.

set_field(key, value)

Set a field.

get_key()

Get the key of a document. This is the short ID the user would use to reference a document in a latex paper.

set_key(value)

Set the key.

get_filename()

Get the URI (eg "file:///home/me/A_File.pdf") of the file associated with the document.

set_filename(value)

Set a document's file URI to value.

parse_bibtex(value)

Parse value as plain-text BibTex and set the document's fields accordingly.

Configuration dialog

Plugins can provide a configuration user interface invoked from the preferences dialog. The configuration button in the preferences dialog is enabled if the `referencer_config()` plugin includes a function.

Action Plugins

Three things are required to insert actions into the Referencer UI: a description of the action, a string describing location of UI elements, and a function implementing the action.

An action is defined as a dictionary with the following fields:

Table 2. Actions Field Description name Author's name, or comma separated list. eg. "Bob Dole" label A version string for the plugin. eg. "1.3.2" tooltip Long description icon Filename or stock eg. "foo.png", eg. "_stock:gtk-edit". callback Action Function sensitivity (optional) Sensitivity policy function accelerator (optional) Shortcut key, eg "<control><shift>q"

An action definition might look like this:

```
action = { "name":"_plugin_genkey_genkey", "label":_("Generate Key"), "tooltip":_("Generate keys for the selected documents from their metadata"), "icon":"_stock:gtk-edit", "callback":"do_genkey", "sensitivity":"sensitivity_genkey", "accelerator":"<control>g" }
```

The plugin should also create a list called `referencer_plugin_actions` containing all actions:

```
referencer_plugin_actions = [action]
```

To place the actions in the user interface, the field "ui" must be added to the `referencer_plugin_info` dictionary. The ui value is a piece of GtkUIManager XML. This specifies UI elements as children of existing structures such as menus and toolbars. The parent structure is defined as `src/referencer_ui.h` in the Referencer source code. Here's an example of creating menu items in the **Document** menu and in the toolbar:

```
<ui> <menubar name='MenuBar'> <menu action='DocMenu'> <placeholder name='PluginDocMenuActions'> <menuitem action='_plugin_genkey_genkey'/> </placeholder> </menu> </menubar> <toolbar name='ToolBar'> <placeholder name='PluginToolBarActions'> <toolitem action='_plugin_genkey_genkey'/> </placeholder> </toolbar> <popup name='DocPopup'> <placeholder name='PluginDocPopupActions'> <menuitem action='_plugin_genkey_genkey'/> </placeholder> </popup> </ui>
```

The functions referenced as "callback" and "sensitivity" in the action dictionary both have the prototype `myfunction(library, documents)` where documents is a list of `referencer.document` and library is a unused. For example: def sensitivity_genkey (library, documents): pass def do_genkey (library, documents): pass

Some plugin actions may wish to display arbitrary UI such as dialogs: this can be done using PyGTK. A detailed explanation of PyGTK would be outside the scope of this document: there are many tutorials on writing PyGTK applications. Note that GTK is already initialised by Referencer, so a plugin must not do any GTK initialisation or finalisation. For example, the following code would stand entirely alone:

import gobject import gtk dialog = gtk.Dialog (buttons=(gtk.STOCK_CANCEL, gtk.RESPONSE_REJECT, gtk.STOCK_OK, gtk.RESPONSE_ACCEPT)) label = gtk.Label ("Hello World") dialog.vbox.pack_start (label) dialog.show_all () response = dialog.run () dialog.hide ()

For an example of an action plugin, have a look at `plugins/genkey.py` in the Referencer source code tree.

Metadata Plugins

Metadata plugins provide a function to fill out a document's metadata fields based on a document identifier.

To describe which identifier formats are supported, the plugin should create a list of strings called `referencer_plugin_capabilities`. At time of writing, the possible capabilities are "doi", "pubmed" and "arxiv".

To do the lookup, a function `resolve_metadata`(doc, method) should be created. doc is the `referencer.document` whose fields should be filled out, and method is one of the capability strings listed in `referencer_plugin_capabilities`.

For an example of a metadata plugin, have a look at `plugins/pubmed.py` in the Referencer source code.

About Referencer

Referencer and this Plugin Writing guide were written by John Spray (jcspray@icculus.org). Some contributions to this guide came from Mario Blättermann (mariobl@gnome.org. To find more information about Referencer and the most recent documentation, please visit the project page.