# 1. XBMC Addon Developers Guide

Author: Ashley Kitson

## 2. Table of Contents

# Revisions

| Version | Date | Item | Author |
|---|---|---|---|
| 0.1 | 06 Sept 2010 | Initial Draft | AK |
| 0.2 | 08 Sept 2010 | Second draft | AK |
| 0.3 | 09 Sept 2010 | Third draft<br>– includes first example program<br>- license added | AK |
| 0.4 | 09 Sept 2010 | Fourth draft<br>- includes second example | AK |
| 0.5 | 11 Sept 2010 | Fifth draft<br>- Completed language and settings.xml sections | AK |
| 0.6 | 15 Sep 2010 | Added final example | AK |
| 0.7 | 20 Sept 2010 | Type corrections etc | AK |

# Introduction

## *Purpose*

- to provide a guide for would be XBMC Python plugin addon script developers
- to provide the skeleton frameworks to enable script developers to get up and running quickly
- to provide links to authoritative sources of information for script developers

## *Caveats*

- This document does not try to teach you how to program in Python
- This document does not cover developing for the core XBMC system

## *Conventions*

Any commands that you need to type at a terminal are shown thus;

```
sudo apt-get install xbmc
```

Any program examples, i.e. code that you will type in is shown thus;

```
import urllib,urllib2,re,xbmcplugin,xbmcgui
#TV DASH - by You 2008.
def CATEGORIES():
        addDir('','',1,'')
        addDir( '','',1,'')
```

## *Prerequisites*

The first two of these are really nice to haves. If you are not a programmer, there is nothing to stop you giving it a go, it'll just take a bit longer is all.

- Some knowledge of application development, particularly in a GUI environment
- Knowledge of Python scripting
- Installed version of XBMC 10 (Dharma) minimum – you do not need the development source version

# First Steps

## *The development user*

This document assumes for the purposes of illustration, that the user that you installed XBMC with is 'xbmc' and that that user has all rights to the /home/xbmc directory. Please bear this in mind if you installed under a different user name on your box.

In addition you may want to set up another user on the XBMC server that you will use to login as the developer. This user should have sudo rights on a Linux based system. This is your development user.

## *Your development environment*

You need to consider how you want to work whilst programming. Three options are given in order of preference.

### Local working on your PC

Using a separate development environment is by far the easiest and preferred method of development. You will need to install XBMC on your local machine and run it there. In some senses this is not a bad idea as you can install a plain vanilla XBMC and concentrate on developing and testing your addon in isolation, moving it to your 'production' machine for final integration testing and thence on into the wild. In this case you can install XBMC[1] as your normal logged on user and you shouldn't need to worry about file permissions etc. You can run XBMC in a window to see the results of your work. (Go to System Settings - System Settings – Video Output and set Display Mode to Windowed.) Having a dual screen set-up can be very beneficial in this scenario.

### Local working on the XBMC server

This scenario is similar to working on a local machine except that you will need to install a window manager (Linux machines), run XBMC in a window and connect a keyboard and mouse to the machine (there are some nice wireless keyboards with touchpads about, Toshiba among others do one.) Otherwise the set-up is the same as for a local PC installation.

### Remote working to an XBMC server

In this scenario you are going to use your XBMC server to run your code on, but use another machine to do the coding from. You need to be able to access the server from the local machine.

Running XBMC under windows?
- share the xbmc user directory and set permissions for your development user to read and write to the directory and its subdirectories

Running XBMC under Linux?
- Perhaps the easiest way to do this is to set the file permissions on /home/xbmc to allow the group to read and write, enrol your development user name in the xbmc group. Set the sticky bits for the xbmc user and group so that all new files are owned by the xbmc user /group irrespective of who creates them;

```
sudo chmod -R u+s /home/xbmc/.xbmc
```

1 Fedora users can use a Dharma repository. See http://forums.fedoraforum.org/showthread.php?t=229121

```
sudo chmod -R g+rws /home/xbmc/.xbmc
sudo useradd -G xbmc your_dev_user_name
```

Then logon via ssh (or sftp) from your working machine. Alternatively, install Samba and set up the shares so that you can access your XBMC installation as any user. *Ref to Samba docs*

In either OS, set up permanent mappings from your working machine to the XBMC server so that your editors etc, can see the files

## *IDE's*

An integrated development environment is a good place to start when coding in any language. Check out the following link for a list of free and non free offerings. Look out for ones that support not only Python scripting, but also XML editing and in particular, local and remote debugging; as someone once said – Professional programmers use a debugger!

http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

See what other people are using at

http://stackoverflow.com/questions/81584

## *Editors*

If you don't/can't use an IDE, then check

http://wiki.python.org/moin/PythonEditors

for a list of editors that support Python in some way. And don't forget you can use any simple editor such as Nano or Gedit in Linux, ? In Windows, ? In Mac to edit python source code

## *Access to documentation*

It's a really good idea to have a web browser set up so that you have all the documentation you need at your finger tips. So go ahead and download another browser that is different to your normal one (e.g. I use Firefox for everyday stuff and Chrome as my Developers reference library,) or at least set up a group of tabs in your regular browser that you can fire up when you are developing. See the reference section at the end of this document for sites you will want to have at hand.

# Script Basics

## *Anatomy of an addon*

As of XBMC V10 (Dharma) all addon script packages have a similar structure.

Each addon (as they are now called – separated plugins, scripts etc are pre Dharma version), has its own directory. This directory is located in the .xbmc/addons directory.

### Directory Name

Your directory name should follow this convention[2];

<addonType>[.<mediaType>].<yourPluginName>

The name parts are case sensitive and must be in lower case. The dot character separates each name part – you can use further dot's to separate things in your plugin name if you wish, alternatively you may use a hyphen (-). No other non-alphanumeric characters should be used.

The addonType is one from the following list

| Addon Type Name | Description |
|---|---|
| repository | A repository definition file that allows users to add new repositories to the XBMC addon manager. Media type is not required. |
| plugin | A plugin script or module that adds to the functionality of XBMC. Plugins appear under the relevant media section of the main home menu. |
| script | A runnable program file that will appear in the Program section of the main home menu. |
| skin | An XBMC skin definition and its supporting script files. Media type is not required. |

The mediaType is not required in all cases. The following table describes the available mediaTypes for the available addon types. Your addon may provide more than one mediaType if you wish, whereby it will appear in more than one section. In most cases, however, a single media type will suffice, and it may be preferable to have multiple addons each providing a single media type rather than one addon that tries to do it all.

| Addon | Media Type Name | Description |
|---|---|---|
| plugin<br>script | audio | A music addon that will appear in the main menu music section |
| | video | A video addon that will appear in the main menu video section |
| | picture | A picture addon that will appear in the main menu picture section |
| | weather | A weather addon that will appear in the main menu weather section |
| script | module | A script plugin that will not appear under a category or within the Addons manager, but provides support for other addons |

2 The convention is not obligatory but represents best practice. See section on addon.xml for further information

The plugin name is up to you, but beware that others haven't used it before you.

So for instance, if you are creating an addon that integrates the Gpodder software with XBMC for audio podcasts you might name your directory;

plugin.audio.gpodder-xbmc[3]

If you are creating a screen scraper to present tv shows from MyGreatTv.com. It might be;

plugin.video.my-great-tv-com

A script to ping all your friends on twitter to tell them you are home might be called;

script.ping-twits-i-am-home

Having settled on your name create the directory under the .xbmc/addons directory.

## Directory structure

Your directory contains all the resources needed to operate your addon. The directory must be considered read-only and should not be used for storing intersession or transient data. Other mechanisms are available to do that (more later.)

The directory must contain a file called 'addon.xml'

The directory may optionally contain the following files;

- icon.png
- fanart.jpg
- changelog.txt

Other files may be required to run your addon, most notably the primary or start script if your addon is a piece of python code. It is also considered good practice to place various addon resources and support code modules as follows;

addon.xml
icon.png
fanart.jpg
changelog.txt
addon.py

/resources

    settings.xml

    /language

    /lib

    /data

    /media

i.e. everything that shouldn't be in the root of your directory is considered a resource and should be placed in that sub-directory or a sub-directory from /resources.

3 This Gpodder example will be used in this document.

## File contents

### *icon.png*

Unless your addon is never viewable by the end user, provide this item. It is displayed in various places within XBMC.

- It should convey what the addon does to the user by graphical means.
- It should be a 256x256 PNG.
- It should not have any shadows, gloss, or other overlays on it - XBMC's skinning system will handle that.
- It is suggested that a logo on a plain background (non-transparent) is best in many situations (e.g. for addons that retrieve media from an online service, use that service's logo [as long as you are free to do so]).

### *fanart.jpg*

This is a 'nice to have', and helps to keep XBMC graphically rich. fanart.png should be a 16:9 JPG image. Some simple guidelines:

- It is intended for the background, so should be simple and without text where reasonable.
- A 1280x720 JPG image. It should certainly be no larger than 1920x1080.
- Keep it as small as is reasonable with respect to file-size. Remember that hundreds of thousands of users are going to be downloading this. :-)

### *changelog.txt*

A text file that contains a description of each release change that you make to the system. This is displayed in the XBMC addon installation/update system. Recommended format is to have it sorted by version in descending order, with a simple description as to the major changes (new functionality, big fixes etc,) in each version. (In the author's opinion, too many addons skip this piece of information making it difficult for users to determine whether a particular problem that they may have been having has been fixed or not.)

### *resources/settings.xml*

An XML file that contains the current configuration for the addon. If your addon has configurable items that are set by the user, put them here. This file drives what the user sees when they click on Addon settings for your addon in the various places in XBMC that it can appear. You need do no coding to utilise this functionality

The format for the settings file is relatively straightforward as can be seen in the following example;

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<settings>
     <category label=”General”>
          <setting id="username" type="text" label="2000"
default="" />
          <setting id="password" type="text" option="hidden"
label="2001" enable="!eq(-1,)" default="" />
          <setting type="sep" />
          <setting id="debug" type="bool" label="2002"
default="false" />
     </category>
</settings>
```

You need to supply at least one category element.

The label attribute of both categories and settings can be a real piece of text or the id of a language string in your language files.

Setting type and additional attributes can be one in the following table

| **Type** | **Description** | **Value attribute (value="")** | **Notes** |
|---|---|---|---|
| text | Creates keyboard input | | Can have 'option' attribute set "true" or "false", will hide the text value if false. |
| file | Creates a file selector | | |
| folder | Creates a folder selector | | |
| enum | Creates a select box | Pipe separated list of values e.g. "1\|3\|5\|10\|All" | Using xbmcplugin.getSetting(pluginId,'mytagname') on an enum will return the index into the list, not the value itself. Also be aware that 1 digit indexes may be returned as '01', '02' etc |
| labelenum | | | Same as enum except the value is returned by getSetting() instead of the index |
| ipaddress | Creates IP dialog | | |
| integer | Creates numeric dialog | | |
| bool | Creates radio button | | Set default or value attribute to to "true" or "false" (note case).<br><br>Using xbmcplugin.getSetting(pluginId,'mytagname') on a bool will return 'true' or 'false'. Use appropriate conversion to turn into real boolean. |
| sep | Creates a separator line in the dialog | Ignored | |
| music | | | |
| video | | | |
| pictures | | | |
| programs | | | |
| local | | | |
| fileenum | Create a file selector based on path set in value attribute | Set value to root of directory you want to use for selection | Optional 'mask' attribute ? |

| | | | |
|---|---|---|---|
| action | Executes a script when selected | | Set the 'action' attribute to the name of your function to execute. e.g. `action="RunPlugin(plugin://video/Apple Movie Trailers Plugin/?update=newest)"`<br>Optionally set the 'option' attribute to “close” if you want to close the settings dialog when you click the setting. |

Settings can have additional attributes

source=”” "video", "music", "pictures", "programs", "files", "local" or blank. if source is blank it will use the type for shares if it is a valid share if not a valid share it will use, both local and network drives.

visible=”” “true”, “false”. Determines if the setting is displayed in the settings dialog (default = 'true')

enable=”” Allows you to determine whether this setting should be shown based on the value of another setting. 3 comparators are available,

- eq() Equal to
- gt() Greater than
- lt() Less than

You can AND the comparators using the + symbol and negate it using the ! symbol (e.g. !eq() ). Each comparator takes 2 operands,

- Relative position of setting to compare
- Value to compare against

Thus if we place settings in our file in the order
myFirst setting
mySecondSetting
myThirdSetting

for the third setting we might add the option
enable=”gt(-2,3) + lt(-2,10)”

### *resources/language*

put any language translation capability into this directory. A number of XBMC components expect language strings to be in this directory, so if you are using them, they must be here. For each language that you are supporting you will create a subdirectory named after the language that you are including. The list of languages supported by the core XBMC system is ;

| **Language (and language folder name)** | **Lang code used in addons.xml** |
|---|---|
| Bulgarian | bg |
| Catalan | ca |
| Chinese (Simple) | zh |

| Language (and language folder name) | Lang code used in addons.xml |
|---|---|
| Chinese (Traditional) | zh |
| Croatian | hr |
| Czech | cs |
| Danish | da |
| Dutch | nl |
| English | en |
| English (US) | en |
| Esperanto | eo |
| Finnish | fi |
| French | fr |
| German | de |
| Greek | el |
| Hebrew | iw |
| Hungarian | hu |
| Icelandic | is |
| Indonesian | in |
| Italian | it |
| Japanese | ja |
| Korean | ko |
| Maltese | mt |
| Norwegian | no |
| Polish | pl |
| Portuguese | pt |
| Portuguese (Brazil) | pt |
| Romanian | ro |
| Russian | ru |
| Serbian | sr |

| Language (and language folder name) | Lang code used in addons.xml |
|---|---|
| Serbian (Cyrillic) | sh |
| Slovak | sk |
| Slovenian | sl |
| Spanish | es |
| Spanish (Mexico) | es |
| Swedish | sv |
| Turkish | tr |
| Ukrainian | uk |

NB, where a language shares a lang code, XBMC will decide the language variant based on region and language settings that user has set in their preferences.

You are strongly urged to support multilingual capabilities with your addon. This allows the widest possible usage of the addon. The default language of XBMC is English (NB not English (US)), so if you are providing additional language support, you must at least provide English in addition to your other supported languages.

For example, lets say we are going to support English. English (US) and French. In the /resources/language directory you will create three sub directories;

/resources/language/English

/resources/language/English (US)

/resources/language/French

In each directory, create the file 'strings.xml'. The file format is;

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<strings>
     <string id="n">My text</string>
</strings>
```

The id is a number. You can use your own number sequence, but the convention is for addon id numbers to be in the range 30000-30999. So for our three languages we might create;

English/strings.xml

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<strings>
   <!-- Addon Name -->
    <string id="30000">My Addon</string>
    <!-- some other strings -->
    <string id="30100">Hello</string>
    <!-- Settings -->
    <string id="30200">Username</string>
    <string id="30210">Password</string>
    <string id="30220">Debugging (restart script to apply)</string>
```

```
</strings>
```

English (US)/strings.xml

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<strings>
   <!-- Addon Name -->
    <string id="30000">My Addon</string>
   <!-- some other strings -->
    <string id="30100">Hi</string>
    <!-- Settings -->
    <string id="30200">Username</string>
    <string id="30210">Password</string>
    <string id="30220">Debugging (restart script to apply)</string>
</strings>
```

French/strings.xml

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<strings>
   <!-- Addon Name -->
    <string id="30000">Mon Addon</string>
   <!-- some other strings -->
    <string id="30100">Bonjour</string>
    <!-- Settings -->
    <string id="30200">Pseudo</string>
    <string id="30210">Mot de passe</string>
    <string id="30220">Débogage (redémarrer le script à
appliquer)</string>
</string
```

Google Translate and Yahoo's Babelfish are good tools for basic translation, particularly of short strings. Do however get a native speaker to check over your translations if at all possible, because both tools may not use the most appropriate idiomatic syntax. (For instance Google translates 'Username' as 'Nom d'utilisateur'. A more appropriate idiomatic translation would be 'Pseudo'.) The worst that can happen is that you provide a somewhat amusing translation to a native speaker.

The worked example section details how to use translations in your script.

### *resources/lib*

put any module definitions or third party software libraries into this directory

### *resources/data*

Store any other static data structures your application requires here. Examples might be XLT/XSD files or static XML files that contain lookup tables etc.

### *resources/media*

store any static media (picture, audio, video etc.) files in this directory

Remember, the above is a recommended outline for your addon – if you need fewer or more directories to organise your work, just change it. The only absolutely required file is;

### *addon.xml*

addon.xml tells XBMC;

- what the addon provides
- what the addon relies on to work
- what script to run when it is fired up (if it is meant to be started.)

The basic format of the file is

```
<?xml version="1.0" encoding="UTF-8"?>
<addon
  id=""
  version=""
  name=""
  provider-name=""
  >
  <requires> </requires>
  <extension point="see notes"></extension>
  <extension point="xbmc.addon.metadata">
    <summary></summary>
    <description></description>
    <platform></platform>
  </extension>
</addon>
```

The attributes of the <addon> opening tag are required

- id - The id attribute is the unique identifier used for this addon. It must be unique, and must use only lowercase characters, periods, underscores, dashes and numbers. **This identifier is also used as the name of the folder that contains the addon**. See comments on directory naming above.
- version - The version attribute is used by XBMC to determine whether updates are available. You should use something like 1.0.3 for this.
- name - The name attribute is the name of the addon as it appears in the UI. This should be in English where it makes sense for it to be so, and is not translatable.
- provider - The provider-name attribute is used as the author field. This could be a team of authors or a single author.

The <requires> tag is optional but where it exists it has the following format;

```
<requires>
     <import addon="" version=""/>
</requires>
```

i.e a number of <import> tags make up the <requires> tag. Each <import> tag has two attributes;

- addon the name of the library that the addon is dependent on
- version the minimal version number of the required library element.

Example;

```
<requires>
     <import addon="xbmc.gui" version="2.11"/>
     <import addon="script.module.simplejson" version="2.0.10"/>
```

```
</requires>
```

In this example, the addon requires the XBMC GUI library which comes with XBMC and a third party library (still within XBMC control,) that is delivered in another addon package called 'script.module.simplesjon'.

NB, if your module relies on third party modules, they must be installed prior to installing your module, by the user. Assuming the third party module is available on an existing repository, XBMC will install this automatically when the user installs your addon. Libraries outside of the xbmc domain must be loaded by your code and do not form part of the <requires> statement as XBMC doesn't know what to do with them.

The <extension> elements are required. You will need at least two of them; the first is always named with a point attribute of 'xbmc.addon.metadata'. This must be supplied by all addons. It tells the addon manager important information about your addon that it uses to display to the user or in managing the addon itself. The second <extension> tells XBMC where to locate this plugin within its hierarchy via the point attribute. You must have at least one of these extensions, and it is possible to have more than one if for instance your addon extends say, Music and Video

An <extension point="xbmc.addon.metadata"> tag has the following sub tags;

- <summary [lang=""]> One or more summary elements provide a short summary of what the addon does. This should be a single sentence. It may be translated into multiple languages, whereby each has a lang="ch" attribute. No lang attribute indicates English.
- <description [lang=""]> One or more description elements provide a more detailed summary of what the addon does. Again, these can be translated.
- <disclaimer [lang=""]> One or more disclaimer elements that indicate what (if any) things the user should know about the addon. There is no need to have a disclaimer if you don't want one, though if something requires settings, or only works in a particular country then you may want to state this here.
- <platform> A platform tag which specifies which platforms (operating systems, hardware) this addon runs on. Many addons will run on all platforms, so "all" is an option, as are "linux", "osx", "windx" and "wingl". If the platform tag is missing, it is assumed the addon runs on all platforms.

Depending on what part of XBMC you are extending will depend on what attributes are contained in the second <extension tag>. Your addon may specify additional extension points if required.

The most common extension point that will be used by plugin addon developers is "xbmc.python.pluginsource" . The following addon.xml example demonstrates the setup for our Gpodder plugin addon.

```
<?xml version="1.0" encoding="UTF-8"?>
<addon
  id="plugin.audio.gpodder-xbmc"
  version="0.0.1"
  name="Gpodder XBMC Integrator"
  provider-name="Ashley Kitson"
  >
  <requires>
     <import addon="xbmc.gui" version="2.11"/>
     <import addon="gpodder.api"/>
  </requires>
```

```
  <extension point="xbmc.python.pluginsource" library="gpodder-
xbmc.py">
     <provides>audio video</provides>
  </extension>
  <extension point="xbmc.addon.metadata">
    <summary>XBMC Integration to Gpodder</summary>
     <summary lang="fr">Intégration de XBMC Gpodder</summary>
    <description>Provides the ability to control Gpodder
podcatching application from within XBMC including automating
podcast downloads, managing subscriptions and
episodes</description>
     <description lang="fr">Fournit la capacité de contrôler
gpodder baladodiffusion application à partir de XBMC, y compris
l'automatisation téléchargements podcast, la gestion des
abonnements et des épisodes</description>
    <platform>linux</platform>
  </extension>
</addon>
```

The <extension point="xbmc.python.pluginsource"> tag has an extra attribute;

- library This is the name of the python script (startup script) that will be run when the plugin is activated. This file must exist in the root of your addon directory.

The extension has an addition sub element;

- <provides> is a whitespace separated list of image, video, audio, executable. This determines in what area (or context) of the XBMC system your addon will make itself visible in;
  image => Pictures
  video => Video
  audio => Music
  executable => Programs
  <blank> => No visible presence

  At the present time there is no inbuilt method provided by the XBMC API libraries to determine what context we are currently running in. See Appendix D for a hack to determine the context that you can include in your scripts.

See http://wiki.xbmc.org/index.php?title=Plugin_Sources for additional information on this extension type.

## *How to make the addon visible to XBMC*

Currently XBMC scans for addons only on startup. This may be changed in the future. So for now, assuming you have put the files in the correct place (i.e. within .xbmc/addons directory as described in previous sections,) simply restart XBMC. Check that you can see the addon in the expected place. If not, its time to debug! See section on debugging below.

## *Packaging your script for release via XBMC Addons repository*

Once your addon is complete you are ready to release it in the wild, or at least via the inbuilt repository for XBMC. XBMC require that you host your addon in a repository for preference and either Git or SVN is compatible. Failing that, you need to create a ZIP file (i.e. winzip or similar created with a Linux/Mac tool,) and place it on a public server that XBMC can access. See

http://wiki.xbmc.org/index.php?title=Official_Addons_Repository

for more details. XBMC have a few provisos for adding to the official XBMC repository (all of

them reasonable.)

Remember - letting your addon loose carries a certain amount of responsibility, primarily to keep it bug free. For scraper based authors this can be problematic as websites do simple things like change their layout, css or other html elements on a page.  If you want to enhance it as time goes on, that of course is up to you,  but can I suggest that if you do get to a point that you don't want to maintain the code any more, you find someone who will take it on, or remove it from distribution.

Alternatively, you can create your own public facing repository and simply submit your repository details to XBMC, (using a <extension point="xbmc.addon.repository"> addon configuration.)  A third alternative is to distribute the addon as a zip file from your own site, as XBMC has the ability to install from zip file.

# Worked examples

The following worked examples are based in the Music section of XBMC

## *Static Listing*

This is the simplest of addons and is used to introduce the basic constructs required to develop a 'xbmc.python.pluginsource' extension addon.

### Goal

- To introduce the basic program construct for any xbmc.python.pluginsource addon
- To demonstrate a working addon

### Code it!

Go to Appendix A and cut and paste the contents of the files there into the places specified.

This is the contents of our basic addon startup script.  If you've been following the examples so far, you should by now have an addons.xml file that points to your startup script.

```
"""
XBMC Addon Developer's Guide
Example 1 - The basic plugin structure
        Demonstrates creating a static list

NB This is done using functions - you could use classes

Author: Ashley Kitson
"""

# Step 1 - load in xbmc core support and setup the environment
import xbmcplugin
import xbmcgui
import sys

# magic; id of this plugin's instance - cast to integer
thisPlugin = int(sys.argv[1])

# Step 2 - create the support functions (or classes)

def createListing():
    """
    Creates a listing that XBMC can display as a directory listing

    @return list
    """
    listing = []
```

```
28      listing.append('The first item')
29      listing.append('The second item')
30      listing.append('The third item')
31      listing.append('The fourth item')
32      return listing
33
34
35  def sendToXbmc(listing):
36      """
37      Sends a listing to XBMC for display as a directory listing
38      Plugins always result in a listing
39
40      @param list listing
41      @return void
42      """
43      #access global plugin id
44      global thisPlugin
45
46      # send each item to xbmc
47      for item in listing:
48          listItem = xbmcgui.ListItem(item)
49          xbmcplugin.addDirectoryItem(thisPlugin,'',listItem)
50
51      # tell xbmc we have finished creating the directory listing
52      xbmcplugin.endOfDirectory(thisPlugin)
53
54  # Step 3 - run the program
55  sendToXbmc(createListing())
```

## Breaking it down

A plugin type addon is required to do only one thing;  display a list of links that XBMC can interpret as something to play or call additional functionality with.  The example program illustrates the basic program structure;

- Create your list of things to display
- Send them to XBMC to display

I have split these two bits up for clarity.  Purists would say that you could create and display at the same time and so you can, but it is no bad idea for maintainability to split functionality into small blobs if possible.  As your programs get bigger, more complex and probably slower to run (i.e. if you are screen scraping and need to fetch data from another web site,) you may need to revert to creating and displaying at the same time so that the user isn't left with a blank screen whilst your program does something useful.

This example creates the list.  The list it provides doesn't do anything yet so don't be despondent. Example 2 expands on it to do something active.

L11 – We need to import some supporting libraries.  In addition to the Python core 'sys' lib, we will invariably need to add the XBMC GUI (xbmcgui) and Addon (xbmcaddon) libraries.

XBMC also provides a plugin helper library (xbmcplugin) in addition to its own core library (xbmc). These are all documented via the Pydocs script, see Current XBMC Python library API references

L17 – This is a little bit of scriptfoo magic that you won't find documented other than in examples. This line grabs the id of the currently running plugin instance (i.e. your one). As it is possible that XBMC may be running more than one instance of a plugin, we need to be able to identify which one we are dealing with. If you grab and set this early in your script, you can then access it when required. A lot of XBMC functionality requires this id. In your function definitions (or indeed your class definitions when you get to writing them,) you can always access it by telling the function that it is to use the global instance of your variable. This is what the statement at line 44 does. It makes a variable which is in its parent scope available within the local function scope.

L21 – This defines a function whose job it is to create the list of things we want to display. This is where the real work goes on. I am using the native Python list object (similar to arrays in PHP and the like,) to store my list information. As you can see, a very simple list.

L35 – This defines a function that will send the list items to the XBMC console for display as a standard directory listing.

L47 – We loop through our list of items, creating an xbmcgui.ListItem() from each one and then send it to XBMC via xbmcplugin.addDirectoryItem(). Both the ListItem() and addDirectoryItem() methods have additional parameters that we can use to do something useful with the list we produce, but I've left that out for clarity at the moment.

L52 – We need to tell XBMC that we have finished sending items for display. This line does that and as it is the effective last line in the script, control now passes back to XBMC and our work is done.

L55 – This runs the main program. It calls the sendToXbmc() function, passing in the result of a call to the createListing(). We could have written it on two lines thus;

```
list = createListing()
sendToXbmc(list)
```

but the sort of shorthand in the main listing is acceptable in simple cases. I could have written Lines 48 and 49 as single line;

```
        xbmcplugin.addDirectoryItem(thisPlugin,'',xbmcgui.ListItem(
item))
```

but as you can see, it wraps the line, making readability less easy. Another alternative would be to spread it over a number of lines thus;

```
        xbmcplugin.addDirectoryItem(
            thisPlugin,
            '',
            xbmcgui.ListItem(item)
        )
```

This is also easily readable. Whatever style you choose, try and keep it readable, for your own sanity and others that may follow you.

A short plea on behalf of code comments; If you want to help your fellow XBMC devs and indeed yourself when it comes to maintainability of your complex code down the line, please liberally comment your code. I see so much code (not just in the XBMC forum threads and in the XBMC addon Python codebase, but in my professional PHP career as well,) that isn't commented, that it makes me want to weep when I have to read it or maintain it. Just get into the habit of commenting. More is definitely better here!

## *Dynamic listing – Non scraper based*

Builds on previous example to show how we can dynamically generate the directory listing for display. This example uses data that is on your machine and being controlled by another application.

For this example to work, you will need to install Gpodder, the podcasting software. Gpodder is available for the Linux and Windows platforms. You can get it at http://http://gpodder.org/ or via your package manager in Fedora and Ubuntu (at least.)

Install the software on your development machine and start it up. Add a new subscription, ensure it plays, (if not – fix the problem). Now add some more subscriptions and download some episodes. Now open up the Preferences dialog and hit the Advanced button. Make a note of the location of **download_dir**, you'll need it later.

Now, XBMC can play the contents of Gpodders downloads all by itself. If you want to prove this, In your Music section in XBMC, add a new source and specify the download_dir you noted earlier.

Browse to your new source in XBMC and click on a file. If the listing shows a '.m3u' extension this will play the only episode if there is only one, or it will display a sub directory listing the available episodes to play. OK, you say, so what? Well, also in your listing you will see the actual folders where the downloaded files are kept, i.e. you appear to get a double listing. Bit confusing isn't it? This example shows you how you can clean this all up via an addon[4].

### Goal

- To create a simple listing using dynamically generated information from an external source
- To show how to interface to python libraries outside of xbmc domain
- To introduce modules to the development process

### Code it!

You need to add some additional files for this example. Go to Appendix B and cut and paste the contents of the files there into the places specified. You will need the download_dir value you collected earlier.

***addondev2.py***

```
"""
XBMC Addon Developer's Guide
Example 2 - Moving on
        Demonstrates creating a dynamic list
        Demonstrates using your own modules and classes

NB This is done using functions - you could use classes

Author: Ashley Kitson
 """
 #
 # Step 1 - load in core support and setup the environment
```

4 If you are interested you can automate GPodder doing the podcatching for you on Linux. See http://forum.xbmc.org/showthread.php?t=80476 and read the bit about using cron. The code you are seeing in this document is forming the basis for a complete Gpodder control integrated into XBMc that the author is currently writing as his first Python/XBMC project. Watch the forum for announcements.!

```
13 #
14 import sys
15 import xbmcplugin
16
17 #addon id - name of addon directory
18 _id='plugin.audio.addon-dev-ex2'
19 #resources directory
20 _resdir = "special://home/addons/" + _id + "/resources"
21 #add our library to python search path
22 sys.path.append( _resdir + "/lib/")
23
24 #import our worker classes from our module
25 import gpodder as worker
26
27 # magic; id of this plugin's instance - cast to integer
28 _thisPlugin = int(sys.argv[1])
29
30 #
31 # Step 2 - instantiate the support classes
32 #
33 creator = worker.creator(_thisPlugin, _id)
34 sender = worker.sender(_thisPlugin)
35
36 #
37 # Step 3 - run the program
38 #
39 sender.send(creator.get())
40 xbmcplugin.endOfDirectory(_thisPlugin)
```

***resources/lib/gpodder.py***

```
"""
 4 XBMC Addon Developer's Guide
 5 Example 2 - Demonstrates creating a dynamic list from Gpodder directory
 6         This module provides the classes that will
 7         create and display the contents
 8
 9 Author: Ashley Kitson
10 """
11 #make xbmc and system modules available
12 import xbmc
13 import xbmcplugin
14 import xbmcgui
15 import dircache
16 import fnmatch
17
18 #define global constants for settings xml tags
```

```
19 __GPOPATH_TAG__ = 'gpoPath'
20
21 #define classes
22
23 class creator:
24    """
25    Responsible for creating the list of items that will get displayed
26    """
27    #
28    # PRIVATE Methods
29    #
30
31    # current instance of plugin identifer
32    _pluginId = 0
33    # plugin name
34    _pluginName = ''
35
36    def __init__(self, pluginId, pluginName):
37       """
38       constructor
39       @parm int pluginId - Current instance of plugin identifer
40       @param string pluginName - Name of plugin calling us
41       """
42       self._pluginId = pluginId
43       self._pluginName = pluginName
44
45    def _createList(self):
46       """
47       Create the dynamic list
48       @access private
49       @returns list
50       """
51       #get the user setting for the gpodder directory
52       dir = xbmcplugin.getSetting(self._pluginId,__GPOPATH_TAG__)
53       #get contents of gpodder directory
54       dirContent = dircache.listdir(dir)
55       #parse contents for all .m3u files
56       dirContent = fnmatch.filter(dirContent, '*.m3u')
57
58       #create listing
59       listing = []
60       for file in dirContent:
61          uri = xbmc.translatePath(dir + '/' + file)
62          label = file.replace('.m3u','')
63          listing.append([label,uri])
64
```

```
65        return listing
66
67
68    #
69    # PUBLIC API
70    #
71
72    def get(self):
73        """
74        Refresh and retrieve the current list for display
75        @access public
76        @returns list
77        @usage     c=example2.creator()
78                list = c.get()
79        """
80        return self._createList()
81
82 class sender:
83    """
84    Responsible for sending output to XBMC
85    """
86    # current instance of plugin identifer
87    _pluginId = 0
88
89    def __init__(self, pluginId):
90        """
91        constructor
92        @parm int pluginId - current instance of plugin identifer
93        """
94        self._pluginId = pluginId
95
96
97    def send(self,listing):
98        """
99        Send output to XBMC
100       @param list listing - the list of items to display
101       @return void
102       """
103       #create listing items
104       # item[0] = list label
105       # item[1] = item uri
106       for item in listing:
107          listItem = xbmcgui.ListItem(item[0])
108          xbmcplugin.addDirectoryItem(
                        self._pluginId,item[1],listItem
                    )
```

## Breaking it down

This addon expands on what we have already learn't.  Some simple things first;

The addon uses a settings file (resources/settings.xml), the format of which was explained earlier in this document.  We need this because the location of the Gpodder downloaded files may vary from one machine to another.  This example doesn't show you how you can edit the value on line.  That comes in a later example!  In the meantime if you need to change the value in it, you can simply do it in your editor.

The addon makes use of a module to drive its core functionality from.  The primary reason is that the code in the gpodder.py module file is potentially reusable in other addons.  The secondary reason is to split up functionality into smaller files that are manageable.  The third reason is that it allows us to show how you pull in your own classes from separated modules in your addon directory hierarchy.

### addondev2.py

L14 – Pull in external modules that we need

L17 to 22 – We need to tell Python where to find our module's files.  These lines set up the path and add it to the python search path.  The _id variable is also used later on.

L25 – having told Python where we can find our library modules, now we can load them.  I've renamed the reference to the gpodder module to be 'worker'.  It isn't necessary, but for semantic reading reasons it is sometimes nice to do this.

L33 & 34 – We now instantiate the classes that we are going to use.  These effectively replace the functions from the last example.  The variable values being passed in are required by the classes to operate.

L39 – Run the program.  Like the previous example, this uses shorthand to run one method with the contents of another.

L40 – Unlike example 1, we explicitly tell XBMC that we have finished work.  This separation of tasks, makes it easier to add functionality via the main program script without having to worry about a module exiting the directory listing.

### resources/lib/gpodder.py

L19 – Python doesn't really support constants in the same way that say, PHP does.  The convention is to use __TAG__ for a variable thatyou want to treat as a constant.  This constant matches the name of the setting that we want to read from the settings file.

L23 – Here we define our class that is going to be responsible for creating our content listing.  It's constructor at L36 simply stores the passed in values for later use.

L45 – Here we define the method that actaully does the work of creating the list.  It has a leading underscore in its name.  Again Python doesn't really support the concept of private or protected methods, it's merely a convention.

L52 – get the setting from the current user's setting file for the gpoPath setting.  This xbmc method takes all the pain out of getting the current user (profile's) addon variables.

L54 – Get the contents of the gpodder directory using the Python dircache module

L56 – Use the Python fmatch library to strip out all teh directory contents except those ending in '.m3u'
We could have concatenated the previous three lines together;

```
    dirContent = fnmatch.filter(
```

```
            dircache.listdir(
            xbmcplugin.getSetting(self._pluginId,__GPOPATH_TAG__)
            ), '*.m3u')
```

L60 – 63 – XBMC really needs two bits of information to operate on in a directory listing;

- the label to display to the user – this is what we did in Example 1
- the URI on which to act. This can be a url (http://), a file location (/home/user/bal/blah.mp3) or any other URI format that it supports (What does it support?)

We know that XBMC knows how to deal with .m3u (playlist files), so all we need to do is give the listing the location of the m3u file.

This loop constructs a list of lists, in each sublist there are two items, the label and the uri

L61 – Important one this – This is the construct that we need to use to ensure that the file path we are using is translated into something that teh platform on which XBMC us running, can understand

L62 – to get he label, we simply remove '.m3u' from the file name

L72 – A 'public' wrapper around around the function that actually does the work.

L82 – definition of the sender class that will output the listing to XBMC

L106 – 108 – The only difference between this and the Example 1 process is that we pass in an extra parameter to xbmcplugin.addDirectoryItem(). The second parameter is the URI of the thing that we want to display when the user clicks on the listing item.

### Homework

The above example doesn't show the podcast image file when each item is browsed on. Have a look at how Gpodder stores this information in the download directories and pick up the file location for adding to the listing.

## *Dynamic listing – A better RSS Reader?*

One of the more frustrating bits of XBMC is that you can't read the RSS news feeds. There are a number of ways of doing it, but how about opening up a browser with the news item? Read on.

### Goal

- To demonstrate utilising another addon as a module library for resuability
- To demonstrate calling a plugin from a directory listing
- To demonstrate starting anothe rporgram from within xbmc

To make this example work, you will first need to install the Rss Editor addon available from team XBMC (default addon library).

### Code it!

The scripts for this example are contained in Appendix C. Cut and paste them into script.rss-chippyash directory under ./xbmc/addons. The full listings also demonstrate the commenting required to comply with XBMC requirements to add licensing for your addon if you want to distribute it. Rather than reproduce the listing again here, I will refer directly to the source code and pull out relevant lines for discussion.

## Breaking it down

### addon.xml

The only real thing of interest here is the requirement

```
    <requires>
        <import addon="script.rss.editor" version="1.5.9"/>
    </requires>
```

which tells the system that we are dependent on the RSS Editor addon

### rss-example.py

Lines 35 – 42

```
#name of the rwparris rss editor addon
__RSSEDITOR_NAME__ = "script.rss.editor";

#setup library path for RSS Editor
_path = xbmcaddon.Addon(__RSSEDITOR_NAME__).getAddonInfo('path')
sys.path.append (xbmc.translatePath( os.path.join( _path, 'resources', 'lib' ) ))
#set up language bootstrapping for XML parser
__language__ = xbmcaddon.Addon(__RSSEDITOR_NAME__).getLocalizedString
```

This is where we set up the system to include the RSS Editor code into our script search path

the __language__ global variable setting is required as the XMLParser class that we will use in our code from the module requires it to be set.

Lines 66 – 79

```
#see if we have a URL parameter
params = creator.getparams()
try:
  url = params["url"]
except:
  url = None

if url == None:
    #do listing
    sender.send(creator.get())
    xbmcplugin.endOfDirectory(_thisPlugin)
else:
    #display news item
    sender.displayNews(url)
```

Here, we get any parameters that are passed on the command line and test to see if we have one called 'url'.  If we do then we call sender.displayNews(url) otherwise we just do as normal, create a listing and display it.

### resources/lib/rss_chippyash.py

L33

```
from xmlParser import XMLParser
```

This is importing the XMLParser class from RSS Editor so we can reuse it.

L45

```
__BROWSER__ = 'google-chrome'
```

This sets the command to fire up our browser. On my system I have Google Chrome installed. You may want to change this to another browser.

L47 – This defines the rssFeed class. This is nothing particularly special, it simply takes a feed URL and interrogates it to retrieve title and article link information.

L101

```
    feeds = XMLParser().getCurrentRssFeeds()
```

This is where we use the RSS Editor's XMLParser class to retrieve and interpret the XBMC Rss feed file.

L106

```
    self.feeds.append(rssFeed(feed['url']))
```

here we just create an rssFeed class for each feed we have.

L108 Our _createListAll method has been modified to create a listing list in our normal format for use by the sender class, by using the information in our feeds data structure.

L141 – The getparams() method is a straight lift from the Shoutcast addon and demonstartes how to get parameters from the command line.

L196

```
    url = "plugin://" + self._pluginName + "?url=" + item[1]
```

We only need to slightly modify our send() method to create a special url that XBMC will interpret as 'run the plugin called X with a parameter list of Y. Using this method you can then call back tou your own addon and do the extra functionality required.

L199 The definition of displayNews() method. A simple one liner which tells the system to open a new process (our browser in this case,) with a parameter – the url of the item we want to read.

Now, purists are going to leap up and down and say stuff like, how can you control a browser with only a remote control and no keyboard. Of course they are absolutely right and it is one of the reasons that a browser hasn't been incorporated into XBMC to date. But hopefully this example gives you some useful insight into a number of techniques that you might use in your own work

## Homework

Using what you've learnt so far,

1/ change the example so that it is a Program script, rather than an addon. What effect does that have on its operation?

2/ Instead of opening a browser, have a think about opening an XBMC dialog instead that displays the news item. Take a look at http://wiki.xbmc.org/?title=HOW-TO_write_plugins_for_XBMC by Voinage and http://wiki.xbmc.org/?title=HOW-TO_write_Python_Scripts_for_XBMC for more information on scraping and GUI related stuff.

# Differences required for other extensions of XBMC

## *Scripts*

<extension point="xbmc.python.script"> addons appear under the Program main menu section of XBMC.  They need to handle all user interaction and are typically constructed to either run in a window constructed by the programmer, or to simply run and display a termination status to the user.  They can have settings like other addons which the user can access via the context menu when highlighting the program name.

For further up to date information see http://wiki.xbmc.org/index.php?title=Script_Sources

## *Repository*

The repository addon only requires two files – addon.xml and icon.png.  It allows XBMC to include your SVN or GIT repository in the list of repositories available for users to download from.  See http://wiki.xbmc.org/index.php?title=Addon_Repositories#Repository_Addon for up to date details on the format of the  <extension point="xbmc.addon.repository">

# Debugging your script

A number of things can go wrong when developing your script. XML validation errors caused by unfinished tags, use of the wrong string delimiter character (e.g. using “” instead of "", often as the result of copying from a word processed source instead of a plain text source).

## *Debug Strategies*

### XBMC debug log

Your first port of call is to look at the XBMC logs. Switch debugging on in the XBMC system settings and then look at the log file it produces. The log is created in the .xbmc/temp directory.

Linux users can use the

```
tail -f xbmc.log
```

command to watch the log

You can view the log using normal editors and tools available on your platform.

### Script debugging

At the current time it is not possible to debug through the XBMC Python code as the Python modules are injected directly from C++ and there are no Python libraries in the normal sense. This is apparently being worked on[5].

You can put print statements in your code and they will appear in the debug log. As that spews out quite a lot of stuff, using something like;

```
print '########################################'
print 'what I actually want to see'
```

can help you locate it easily in the log.

Some helpful setup instructions for installing a Python debugger can be found at;

http://wiki.xbmc.org/index.php?title=HOW-TO_debug_Python_Scripts

http://wiki.xbmc.org/index.php?title=HOW-TO_debug_Python_Scripts_with_Eclipse

The NetBeans Python addon comes with a debugger built in.

5 See http://forum.xbmc.org/showthread.php?t=80984

## The XBMC libraries

XBMC has 4 libraries for Python developers.  This may change in time as the system is restructured to encompass the addon concept instead of the legacy separation of different components.

### xbmc

This library provides access to the core functionality of XBMC

### xbmcgui

This library is primarily concerned with supporting the graphical user interface

### xbmcplugin

This library supports plugin scripts.

### xbmxaddon

This library supports a single class which allows you to get and set information and settings for your addon.

Please see the References section to find ut how you can generate the latest API reference documents for the XBMC library.

# References

## *Python*

Python.org:

http://www.python.org/ - main Python site

http://wiki.python.org/moin/ - Python documentation wiki

http://www.python.org/doc/ - Main Python documentation

## *XBMC*

Additional resources (some of which this document is based on) can be found at;

Addons for XBMC – **prime source – read it!**
http://wiki.xbmc.org/index.php?title=Addons_for_XBMC

Additional information on language support

http://wiki.xbmc.org/index.php?title=Information_on_Language_Support

### Current XBMC Python library API references

You can view the up to date API reference documents by following these steps;

1. Download the Nuka repository xml file at
   http://xbmc-addons.googlecode.com/svn/packages/repository.googlecode.xbmc-addons.zip
2. In XBMC home screen, go to System – addons – Install from Zip file, Browse to where the zip file is located and click OK (or simply enter on the file name)
3. From XBMC Home screen go to System – Addons and select the Googlecode-xbmc-addons entry. From there, navigate to Program Addons and select and install Pydocs printer.
4. From XBMC Home screen go to Programs and select the Pydocs printer. It will ask for a location to put the generated files – select a suitable location on your file system and hit go. This will generate 4 (at time of writing) html files that you can then browse to in your browser. These are the API references.

## *XML*

XBMC makes extensive use of XML for storing configurations and user data. You can find reference material on XML and XSD

Definition for XML Schemas

http://www.w3.org/2001/XMLSchema.dtd

xbmc addons.xml XSD definition

http://trac.xbmc.org/browser/trunk/addons/xbmc.python/pluginsource.xsd

## Credits

Thanks to the following people who have helped either thought guidance, advice or perseverance

Jonathan Marshall – Team-XBMC Developer.  (http://forum.xbmc.org/member.php?u=3254)

rwparris2 – Team-XBMC Python Coder (http://forum.xbmc.org/member.php?u=29904)

Amet – Team-XBMC Member (http://forum.xbmc.org/member.php?u=51533)

# Appendix A – Code for working example 1

Create the following directory and files under .xbmc/addons

plugin.audio.addon-dev-ex1

addon.xml

addondev1.py

into addon.xml, paste the following

```
<?xml version="1.0" encoding="UTF-8"?>
<addon
  id="plugin.audio.addon-dev-ex1"
  version="0.0.1"
  name="XBMC Addon Developers Guide – Example 1"
  provider-name="Ashley Kitson"
  >
  <requires>
     <import addon="xbmc.gui"/>
     <import addon="xbmc.plugin"/>
  </requires>
  <extension point="xbmc.python.pluginsource"
library="addondev1.py">
     <provides>audio</provides>
  </extension>
  <extension point="xbmc.addon.metadata">
    <summary>XBMC Addon Developers Guide</summary>
     <summary lang="fr">XBMC Developers Guide Addon</summary>
    <description>Demonstrates basic code loop for an XBMC plugin
type addon</description>
     <description lang="fr">Démontre boucle code de base pour un addon
type de plugin XBMC</description>
    <platform>all</platform>
  </extension>
</addon>
```

into addondev1.py, paste the following

```
"""
XBMC Addon Developer's Guide
Example 1 - The basic plugin structure
            Demonstrates creating a static list

NB This is done using functions - you could use classes

Author: Ashley Kitson
"""

# Step 1 - load in xbmc core support and setup the environment
import xbmcplugin
```

```
import xbmcgui
import sys

# magic; id of this plugin - cast to integer
thisPlugin = int(sys.argv[1])

# Step 2 - create the support functions (or classes)

def createListing():
    """
    Creates a listing that XBMC can display as a directory listing

    @return list
    """
    listing = []
    listing.append('The first item')
    listing.append('The second item')
    listing.append('The third item')
    listing.append('The fourth item')
    return listing


def sendToXbmc(listing):
    """
    Sends a listing to XBMC for display as a directory listing
    Plugins always result in a listing

    @param list listing
    @return void
    """
    #access global plugin id
    global thisPlugin

    # send each item to xbmc
    for item in listing:
        listItem = xbmcgui.ListItem(item)
        xbmcplugin.addDirectoryItem(thisPlugin,'',listItem)

    # tell xbmc we have finished creating the directory listing
    xbmcplugin.endOfDirectory(thisPlugin)

# Step 3 - run the program
sendToXbmc(createListing())
```

Restart XBMC and then select your addon from the Music section to run it

## Appendix B – Code for working example 2

Create the following directory and files under .xbmc/addons

plugin.audio.addon-dev-ex2

addon.xml

addondev2.py

/resources

settings.xml

/lib

gpodder.py

into addon.xml, paste the following

```
<?xml version="1.0" encoding="UTF-8"?>
<addon
  id="plugin.audio.addon-dev-ex2"
  version="0.0.1"
  name="XBMC Addon Developers Guide – Example 1"
  provider-name="Ashley Kitson"
  >
  <requires>
     <import addon="xbmc.gui"/>
     <import addon="xbmc.plugin"/>
  </requires>
  <extension point="xbmc.python.pluginsource"
library="addondev2.py">
     <provides>audio video</provides>
  </extension>
  <extension point="xbmc.addon.metadata">
    <summary>XBMC Addon Developers Guide</summary>
     <summary lang="fr">XBMC Developers Guide Addon</summary>
    <description>Demonstrates creating dynamic listings for an XBMC
plugin type addon</description>
     <description lang="fr">Montre la création d'annonces dynamiques pour
un addon type de plugin XBMC</description>
    <platform>all</platform>
  </extension>
</addon>
```

into addondev2.py, cut and paste the following

```
"""
XBMC Addon Developer's Guide
Example 2 - Moving on
            Demonstrates creating a dynamic list
            Demonstrates using your own modules and classes
```

```
NB This is done using functions - you could use classes

Author: Ashley Kitson
"""
#
# Step 1 - load in core support and setup the environment
#
import sys
import xbmcplugin

#addon id - name of addon directory
_id='plugin.audio.addon-dev-ex2'
#resources directory
_resdir = "special://home/addons/" + _id + "/resources"
#add our library to python search path
sys.path.append( _resdir + "/lib/")

#import our worker classes from our module
import gpodder as worker

# magic; id of this plugin's instance - cast to integer
_thisPlugin = int(sys.argv[1])

#
# Step 2 - instantiate the support classes
#
creator = worker.creator(_thisPlugin, _id)
sender = worker.sender(_thisPlugin)

#
# Step 3 - run the program
#
sender.send(creator.get())
xbmcplugin.endOfDirectory(_thisPlugin)
```

in resources/settings.xml, paste the following

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : settings.xml
    Created on : September 9, 2010, 1:45 PM
    Author     : Ashley Kitson
    Description: Settings for gpodder example
                 (XBMC Addons Developer Guide example 2)
-->
```

```
<settings>
     <setting id="gpoPath" type="text" label="Path to Gpodder
downloads" default="" />
</settings>
```

Now add your download_dir value to the default setting for gpoPath setting and save the file

into resources/lib/gpodder.py, cut and paste the following

```
"""
XBMC Addon Developer's Guide
Example 2 - Demonstrates creating a dynamic list from Gpodder
directory
            This module provides the classes that will
            create and display the contents

Author: Ashley Kitson
"""
#make xbmc and system modules available
import xbmc
import xbmcplugin
import xbmcgui
import dircache
import fnmatch

#define global constants for settings xml tags
__GPOPATH_TAG__ = 'gpoPath'

#define classes

class creator:
    """
    Responsible for creating the list of items that will get
displayed
    """
    #
    # PRIVATE Methods
    #

    # current instance of plugin identifer
    _pluginId = 0
    # plugin name
    _pluginName = ''

    def __init__(self, pluginId, pluginName):
        """
        constructor
        @parm int pluginId - Current instance of plugin identifer
        @param string pluginName - Name of plugin calling us
```

```
        """
        self._pluginId = pluginId
        self._pluginName = pluginName

    def _createList(self):
        """
        Create the dynamic list
        @access private
        @returns list
        """
        #get the user setting for the gpodder directory
        dir = xbmcplugin.getSetting(self._pluginId,__GPOPATH_TAG__)
        #get contents of gpodder directory
        dirContent = dircache.listdir(dir)
        #parse contents for all .m3u files
        dirContent = fnmatch.filter(dirContent, '*.m3u')

        #create listing
        listing = []
        for file in dirContent:
            uri = xbmc.translatePath(dir + '/' + file)
            label = file.replace('.m3u','')
            listing.append([label,uri])

        return listing



    #
    # PUBLIC API
    #

    def get(self):
        """
        Refresh and retrieve the current list for display
        @access public
        @returns list
        @usage      c=example2.creator()
                    list = c.get()
        """
        return self._createList()

class sender:
    """
    Responsible for sending output to XBMC
    """
    # current instance of plugin identifer
```

```
    _pluginId = 0

    def __init__(self, pluginId):
        """
        constructor
        @parm int pluginId - current instance of plugin identifer
        """
        self._pluginId = pluginId


    def send(self,listing):
        """
        Send output to XBMC
        @param list listing - the list of items to display
        @return void
        """
        #create listing items
        # item[0] = list label
        # item[1] = item uri
        for item in listing:
            listItem = xbmcgui.ListItem(item[0])
            xbmcplugin.addDirectoryItem(self._pluginId,item[1],list
Item)
```

# Appendix C – Code for working example 3

Create ./xbmc/addons/ script.rss-chippyash directory

Into addon.xml, cut and paste the following

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : addon.xml
    Package    : A better RSS reader?
    Author     : Ashley Kitson
    Copyright  : 2010, Ashley Kitson, UK
    License    : Gnu General Public License - see LICENSE.TXT
    Description: XBMC Addon settings for XBMC RSS Reader
-->
<addon id="script.rss-chippyash" version="0.0.1" name="A better RSS
Reader?" provider-name="Ashley Kitson">
  <extension point="xbmc.python.pluginsource" library="rss-
example.py">
      <provides>audio</provides>
      <requires>
          <import addon="script.rss.editor" version="1.5.9"/>
      </requires>
  </extension>
  <extension point="xbmc.addon.metadata">
     <summary>A better RSS Reader?</summary>
     <description>Example dynamic listing program for the Addon
Developer's Guide</description>
     <platform>all</platform>
  </extension>
</addon>
```

into rss-example.py, cut and paste the following

```
"""
    Document   : rss-example.py
    Package    : A better RSS Reader?
    Author     : Ashley Kitson
    Copyright  : 2010, Ashley Kitson, UK
    License    : Gnu General Public License - see LICENSE.TXT
    Description: Main program script for package
"""
"""
This file is part of "A better RSS Reader?"

    "A better RSS Reader?" is free software: you can redistribute
    it and/or modify it under the terms of the GNU General Public
License as
    published by the Free Software Foundation, either version 3 of
the License,
```

```
    or (at your option) any later version.

    "A better RSS Reader?" is distributed in the hope that it will
    be useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public
License
    along with "A better RSS Reader?".
    If not, see <http://www.gnu.org/licenses/>.

"""
#
# Step 1 - load in core support and setup the environment
#
import os
import sys
import xbmcplugin
import xbmcaddon

#name of the rwparris rss editor addon
__RSSEDITOR_NAME__ = "script.rss.editor";

#setup library path for RSS Editor
_path = xbmcaddon.Addon(__RSSEDITOR_NAME__).getAddonInfo('path')
sys.path.append (xbmc.translatePath( os.path.join( _path,
'resources', 'lib' ) ))
#set up language bootstrapping for XML parser
__language__ =
xbmcaddon.Addon(__RSSEDITOR_NAME__).getLocalizedString

# magic; id of this plugin's instance - cast to integer
_thisPlugin = int(sys.argv[1])

#addon id - name of addon directory
_id='script.rss-chippyash'

#set our library path
sys.path.append (xbmc.translatePath( os.path.join( os.getcwd(),
'resources', 'lib' ) ))

#import our worker classes from our module
import rss_chippyash as worker

#
# Step 2 - instantiate the support classes
```

```
#
creator = worker.creator(_id)
sender = worker.sender(_thisPlugin, _id)

#
# Step 3 - run the program
#

#see if we have a URL parameter
params = creator.getparams()
try:
  url = params["url"]
except:
  url = None

if url == None:
    #do listing
    sender.send(creator.get())
    xbmcplugin.endOfDirectory(_thisPlugin)
else:
    #display news item
    sender.displayNews(url)
```

and into resources/lib/rss_chippyash.py, cut and paste the following

```
"""
    Document   : rss_chippyash.py
    Package    : A better RSS Reader?
    Author     : Ashley Kitson
    Copyright  : 2010, Ashley Kitson, UK
    License    : Gnu General Public License - see LICENSE.TXT
    Description: Worker class library
"""
"""
This file is part of "A better RSS Reader?"

    "A better RSS Reader?" is free software: you can redistribute
    it and/or modify it under the terms of the GNU General Public
License as
    published by the Free Software Foundation, either version 3 of
the License,
    or (at your option) any later version.

    "A better RSS Reader?" is distributed in the hope that it will
    be useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
```

```
    You should have received a copy of the GNU General Public
License
    along with "A better RSS Reader?".
    If not, see <http://www.gnu.org/licenses/>.

"""
#make xbmc and system modules available
import subprocess
import sys
import xbmcplugin
import xbmcaddon
import xbmcgui
from xmlParser import XMLParser
import urllib2
from xml.dom import minidom

#define global constants for settings xml tags
__GPOPATH_TAG__ = 'gpoPath'
__GPOUPDT_TAG__ = 'gpoUpdate'

#Addon Title for dialogs
__TITLE__ = 'A Better RSS Reader?'

#Browser command
__BROWSER__ = 'google-chrome'

class rssFeed:
    """
    A very simple RSS feed class
    """
    feedLinks = []
    feedTitles = []
    feedDescs = []
    feedCount = 0

    def __init__(self, feedUrl):
        """
        gather the news items in an RSS feed
        """
        fileRequest = urllib2.Request(feedUrl)
        fileOpener = urllib2.build_opener()
        fileFeed = fileOpener.open(fileRequest).read()
        fileXml = minidom.parseString(fileFeed)
        itemNode = fileXml.getElementsByTagName("item")
        for item in itemNode:
```

```
            self.feedTitles.append(item.childNodes[1].firstChild.da
ta)
            self.feedDescs.append(item.childNodes[2].firstChild.dat
a)
            self.feedLinks.append(item.childNodes[3].firstChild.dat
a)
            self.feedCount += 1


class creator:
    """
    Responsible for creating the list of items that will get
displayed
    """
    #
    # PRIVATE Methods
    #

    # current instance of plugin identifer
    #_pluginId = 0
    # plugin name
    _pluginName = ''
    #this addon class
    _thisaddon = None
    #rss feed list
    _feedList = []
    #rss feed content
    feeds = []

    def __init__(self, pluginName):
        """
        constructor
        @parm int pluginId - Current instance of plugin identifer
        @param string pluginName - Name of plugin calling us
        """
        #self._pluginId = pluginId
        self._pluginName = pluginName
        self._thisaddon = xbmcaddon.Addon(pluginName)
        #get current feeds
        feeds = XMLParser().getCurrentRssFeeds()
        for setNum in sorted(feeds.keys()):
            for feed in feeds[setNum]['feedslist']:
                self._feedList.append(feed['url'])
                #get the feed contents
                self.feeds.append(rssFeed(feed['url']))

    def _createListAll(self):
        """
```

```
        Create the dynamic list of all content
        @param list dirContent - list of __PLAYLIST__ files in
gpodder directory
        @param string dir - gpodder directory location
        @access private
        @return list
        """

        #create listing
        listing = []
        for feed in self.feeds:
            c = range(0,feed.feedCount-1)
            for x in c:
                listing.append([feed.feedLinks[x],feed.feedTitles[x
],''])

        return listing

    #
    # PUBLIC API
    #

    def get(self):
        """
        Refresh and retrieve the current list for display
        @access public
        @returns list
        @usage      c=example2.creator()
                    list = c.get()
        """
        return self._createListAll()



    def getparams(self):
        """
        Pick up parameters sent in via command line
        @return dict list of parameters
        @thanks Team XBM  - I lifted this straight out of the
shoutcast addon
        """
        param=[]
        paramstring=sys.argv[2]
        if len(paramstring)>=2:
            params=sys.argv[2]
            cleanedparams=params.replace('?','')
            if (params[len(params)-1]=='/'):
                params=params[0:len(params)-2]
```

```
            pairsofparams=cleanedparams.split('&')
            param={}
            for i in range(len(pairsofparams)):
                splitparams={}
                splitparams=pairsofparams[i].split('=')
                if (len(splitparams))==2:
                    param[splitparams[0]]=splitparams[1]
        return param



class sender:
    """
    Responsible for sending output to XBMC
    """
    # current instance of plugin identifer
    _pluginId = 0
    # plugin name
    _pluginName = ''

    def __init__(self, pluginId, pluginName):
        """
        constructor
        @parm int pluginId - current instance of plugin identifer
        """
        self._pluginId = pluginId
        self._pluginName = pluginName
        pass


    def send(self,listing):
        """
        Send output to XBMC
        @param list listing - the list of items to display
        @return void
        """
        #create listing items
        # item[0] = list label
        # item[1] = item uri
        # item[2] = image uri
        for item in listing:
            listItem = xbmcgui.ListItem(item[0])
            url = "plugin://" + self._pluginName + "?url=" +
item[1]
            xbmcplugin.addDirectoryItem(self._pluginId,url,listItem
)
```

```
    def displayNews(self, url):
        """
        Display the news item in a browser
        @param string url - Url to display
        """
        subprocess.Popen([__BROWSER__, url])
```

# Appendix D – Code Snippets

## *Get Current Context*

```
class context:
    """
    Hack to get current context that addon is running in

    @usage      context = context().getContext()
    @author chippyash
    @thanks amet - xbmc.org
    @link http://wiki.xbmc.org/index.php?title=Window_IDs
    """
    # Set up window Ids for the various contexts
    _ctxt_audio = (10005,10500,10501,10502)
    _ctxt_video = (10006,10024,10025,10028)
    _ctxt_image = (10002)
    _ctxt_executable = (10001,10020)
    # current  window id
    _currId = 0;

    def __init__(self):
        self._currId = xbmcgui.getCurrentWindowId();

    def getContext(self):
        """
        Returns the current system context
        @return string
('audio','video','image','executable','unknown')
        """
        if self._currId in self._ctxt_audio:
            return 'audio'
        elif self._currId in self._ctxt_video:
            return 'video'
        elif self._currId in self._ctxt_image:
            return 'image'
        elif self._currId in self._ctxt_executable:
            return 'executable'
        else:
            return 'unknown'
```

This allows you to put multiple values in the <extension><provides> tag in addons.xml schema and determine at run time what section of XBMC you are in so that your addon an provide different responses dependent on the context.  At some point the XBMC Python API may incorporate this into it's provision.