PhotoObject()

The object container designated for photos/images.

PhotoAlbumObject()

collection of PhotoObjects()

EpisodeObject()

missing index attribute information for episode

index

An integer identifying the episode number within the season

rating

The framework docs say this is an integer when it is really a float

TrackObject()

missing additional instructions necessary for support of all clients.

For the Plex desktop client (Plex Home Theater, PHT) to handle audio files, it can require a “hint” about the file-type in the form of an extra argument declaring the file extension. For example, ext=’mp3’ for mp3 files or ext=’ts’ for an HLS m3u8 file.

Examples:

```
MediaObject(
    parts = [PartObject(key = Callback(PlayAudio, url=url, ext='mp3'))],
    )
```

or

```
MediaObject(
    parts = [PartObject(key = HTTPLiveStreamURL(Callback(PlayAudio, url=url,
ext='mp3')))],
    )
```

Other clients do not necessarily need this but it including it will improve PHT support without detriment to other clients.

NextPageObject()

not included in Framework documentation

NextPageObject(key, title=”More”, summary=None, thumb=”more.png”**)**
a variation of a DirectoryObject(). Allows for automated paging (Next Page) of long lists

and infinite scrolling for other clients (Click for More or automatically pulled when hitting end of list).

InputDirectoryObject()

missing prompt requirements for Roku

The Roku input screen is intuitive and chooses a Search screen or input screen based on the words you use in the prompt= text. When using the InputDirectoryObject(), to ensure it brings up the right type of entry for Roku users, make sure the prompt= text starts with the word Search for a search input, otherwise it will give you a text input screen.

Eg. That gives you a search screen:
oc.add(InputDirectoryObject(key=Callback(TheseShows, var='x'), title='Search Shows', summary="Look for shows", thumb=R(ICON), prompt="Search for shows you would like to find"))

Eg. That gives you an input screen:
oc.add(InputDirectoryObject(key=Callback(TheseShows, var='x'), title='Search Shows', summary="Look for shows", thumb=R(ICON), prompt="Enter urls for shows"))

Plex/Web
InputDirectoryObject only uses the very first instance in Plex/Web. If you have multiple InputDirectoryObjects (like in DailyMotion's Search directory) it will use only the first one and it will use the topmost "search" field to do this input. This is a known limitation of Plex/Web.

MediaObject()

missing info for audio_language

When creating a PartObject, pass in
streams=[AudioStreamObject(language_code=Locale.Language.English)]

Container File Types

missing FLV and need info on Live Streams

Container.FLV

"mpegts"
For Live Streams, it can be set manually. If set manually, it should be quoted as

Eg. container = "mpegts"

If you use the HTTPLiveStreamURL method then the following attributes and values will be set automatically

protocol = 'hls'
container = 'mpegts'
video_codec = VideoCodec.H264
audio_codec = AudioCodec.AAC
The video_resolution, audio_channels, and optimized_for_streaming attributes can also be added or set.

Full listing of containers and VideoCodecs:

VideoCodecs:

H263 = 'h263'
H264 = 'h264'
VP6 = 'vp6'
WVC1 = 'wvc1'
DIVX = 'divx'
DIV4 = 'div4'
XVID = 'xvid'
THEORA = 'theora'

Containers:

MKV = 'mkv'
MP4 = 'mp4'
MOV = 'mov'
AVI = 'avi'
MP3 = 'mp3'
OGG = 'ogg'
FLAC = 'flac'
FLV = 'flv'

Datetime

missing info for strings to milliseconds and the ability to pull or parse date parts

Datetime.MillisecondsFromString(str)
Converts the given string into milliseconds
returns: int (?)

Datetime.Now() and Datetime.ParseDate()
You can add .day, .month or .year to the end of each of these commands to return that particular section of the date value in the framework numeric format. The .ParseDate() option will also recognize and convert multiple variable formats including partial month values like Nov, Apr, etc.

Hash

missing Hash methods

Hash.SHA1(str**)**
Hash.MD5(str**)**

Locale.Geolocation

missing reference to source of 2 letter country codes and XX for undetermined

returns the 2 letter country code (http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2...) or XX if it's unable to determine where a user is located.
Note: You may see a lot of XX returns here, it's not super up-to-date in terms of looking up IP addresses at the moment.

@route

missing attributes types and PlexSync

Passing attributes types with @route:

By default @route() sets all passed attributes to type str, so any attributes that are passed that need to be of other types must be set in the route
Ex: @route(PREFIX + "/getvideolist", page=int, data_list=list)

PlexSync

PlexSync will save online content for offline viewing. The intent is that it be used for content providers which already allow for downloading their media. If you wish to enable it for your plugin, you must add "allow_sync=True" to the @route decorator for any Menu which returns a list of "sync-able" media.

Adding @route to all functions

If you do not add a route to each of your functions in your __init__.py, you will get an error in your logs "Generating a callback path for a function with no route." And each route must have a unique name. It is best practice to declare a global prefix variable for

your handler and then use that prefix to add a route in each function to limit syntax errors.

```
PREFIX = '/video/yourpluginname'

@handler(PREFIX, TITLE, art=ART, thumb=ICON)
def MainMenu():
    your main menu code here
@route(PREFIX + '/nextfunction')
def NextFunction(title):
    your next function code here
```

Special handling for images

because loading images can be considerably slower than loading text-based metadata, the plugin framework allows for asynchronous loading of image files (thumb, art). This drastically reduces the likelihood of time-outs while loading menus with numerous items each having a unique thumb (or background art). Without the asynchronous loading, the client app cannot render the ObjectContainer until all metadata including images is fully downloaded. With the asynchronous loading, the app will display the text-based metadata immediately, then images will be displayed as they complete downloading. The way to take advantage of this asynchronous load mechanism is to assign a callback as the value for the 'thumb' (or 'art') attribute of the object in question. The callback can be a method elsewhere in the channel code, for example if determining the final url for an appropriate image takes one or more extra HTTP requests. Or, it can be the plugin framework's built-in method which allows for a graceful fallback if the image url fails to load. ie:

```
thumb = Resource.ContentsOfURLWithFallback(url=url, fallback=fallback)
```

By default, the "fallback" parameter has the value of the channel icon but another image can be assigned. The "url" paramater takes either a single URL or a list of URLs as an argument. If a list is provided, then the framework will fallback through each url in the given order ending with the "fallback" image (if one or more urls fail to load). Convention suggests that providing a list of more than three URLs is impractical.

Check for URL Service

not included in Framework Documentation (Not sure where it would be added)

When you have URLs in your __init__.py for multiple websites, you may need to check them for an existing Plex URL service to ensure a directory pull will not return an error. You can use the following code to check a URL:

```
URLService.ServiceIdentifierForURL(url_to_check)
```

If the above code returns a value, then a URL service exists, if the value is None, there

is no Plex URL service for the that url.

Call an URL Service for help

The use of both of the following methods is generally discouraged but can be very useful in certain cases. In effect, they query the list of available URL Services for the requested object(s) and return them rather than passing control from the URL Service to the player app.

URLService.MetadataObjectForURL(url)

URLService.MediaObjectsForURL(url)

Exceptions:

not included in Framework Documentation (Not sure where it would be added)

There are a number of exceptions that can be raised when channel content is inaccessible to the client for some reason. Generally should be raised from within a *try/except* block, like so “raise Ex.Blah”. When used in an URL Service, the ideal time to raise this sort of exception is either in the *MetadataObjectForURL()* function or in the *PlayVideo()* callback function for the MediaObjects. Raising exceptions in the *NormalizeURL()* or *MediaObjectsForURL()* functions should be avoided as it can prevent proper loading of the parent ObjectContainer. The available exceptions are as follows:

- MediaNotAvailable
- MediaExpired
- LiveMediaNotStarted
- MediaNotAuthorized
- MediaGeoblocked
- StreamLimitExceeded

Regular Expressions (regex):

not included in Framework documentation (Not sure where it would be added)

The plugin framework includes support for manipulating regular expressions with the built-in *Regex()* function. It functions similarly to python’s re.compile() and will convert the given string into a regex pattern. The pattern should be assigned to a global variable (at or near the top of the file) which can be referenced later on in the code.

```
RE_MY_PATTERN = Regex(‘string to use as (match) pattern’)
```

It’s possible to force the given string to be treated as a raw string rather than messing with backslash-escaping, by pre-pending it with ‘r’.

```
RE_MY_PATTERN = Regex(r‘\s make escaping complicated for proper (match)’)
```

When referencing the pattern, standard regex functions such as .search(), .match(), and .findall() are usable and return regex match objects, groups, or lists as expected based on the python re equivalent.

```
match_object = RE_MY_PATTERN.search(string_to_be_searched)
```

content_of_1st_match = match_object.group(1)

Information about the Server and Client being used:

Platform

returns details about the server and the machine running it.

Platform.OS

Reports the current server's operating system.

Platform.OSVersion

Reports the current server's operating system version.

Platform.ServerVersion

Reports the current server's version string.

Client

returns details about the client app being used to access the channel.

Client.Platform

Reports the platform of the client currently accessing the plug-in.

Client.Product

Reports the client product currently accessing the plug-in.

Client.Version

Reports the version of the client currently accessing the plug-in.

The Info.plist file

missing many optional keys in Framework, contains out-of-date references (See Below), and separate plist for Services.

The Framework Documentation only refers to the Info.plist and does not separately address creating a ServiceInfo.plist for Services. For a full list of required and optional keys and their value options, please see the following templates:

Info.plist template -
https://forums.plexapp.com/index.php/topic/62610-channel-development-templates/?p=365280
ServiceInfo.plist template -
https://forums.plexapp.com/index.php/topic/62610-channel-development-templates/?p=393464

Add Country list for plist Region Restrictions:

missing region restrictions

<key>PlexPluginRegions</key>

```
<array>
        <string>US</string>
</array>
```

To find the proper two letter code for the country use the format as listed here: http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

PlexFrameworkFlags

missing Framework Flags in plist documentation

an array of strings identifying framework options for the plug-in.

UseRealRTMP

a boolean setting that enables using librtmp instead of the hosted Plex player. Defaults to false(?).
Deprecated. All RTMP videos will be transcoded using librtmp.

```
<key>PlexFrameworkFlags</key>
<array>
        <string>UseRealRTMP</string>
</array>
```

Advanced plist settings:

These settings are not used or necessary for the vast majority of channels. Any channel which uses them is unlikely to be made available via the “official” Channel Directory.

Elevated Code Policy

<key>PlexPluginCodePolicy</key>
<string>Elevated</string>
Running a plugin with the “elevated” code policy allows it to do a wide variety of things which regular channels are forbidden. For example; importing modules which are on the restricted python blacklist, querying PMS directly via the HTTP API, accessing the file system.

Daemon Mode

<key>PlexPluginMode</key>
<string>Daemon</string>
Allows a plugin to continue running and executing code in the background even when it is not in use by any client app.

Currently Unsupported

Channel Subtitles:

There is not currently any support for subtitles in Plex channels. It has been mentioned that it may be implemented at some point but, AFAIK there is no ETA.

Python Tips

Sorting Data

For sorting data returned in your object containers you may use:
oc.objects.sort(key = lambda obj: obj.title, reverse=True)

Dealing with Backslashes

Tip for Python coding

To delete a backslash in a string you have to use:
str.replace('\\','')
(the backslash escapes the character following it)

If you want to use regex and want it to pick up the data just as the regex appears including the backslashes, use r in your regex variable
Ex: RE_TAB = Regex(r'yui3-tabview-content\\">(.+?)<\\/div')

To let the Plex Media Server transcode everything set `optimized_for_streaming` to `False` (example).

Old or Out-of-Date Framework Documentation Information

From the Configuration Files/Info.plist section

-- The example Info.plist is out of date as well as the optional keys listed below -- ShopGirl

PlexMediaContainer

An array of strings identifying the containers returned by the plug-in. The array should contain constants from Container.

Note This property is optional.

PlexAudioCodec

An array of strings identifying the containers returned by the plug-in. The array should contain constants from AudioCodec.

Note This property is optional.

PlexVideoCodec

An array of strings identifying the containers returned by the plug-in. The array should contain constants from VideoCodec.

Note This property is optional.

From URL Generating Functions

-- this is old and no longer applicable -- Gerk

RTMPVideoURL(url, clip=None, clips=None, width=None, height=None, live=False)

Returns a URL that starts WebKit playback of the given RTMP URL using Plex's hosted RTMP player.