Workflow Library

Introduction

This is an AppleScript library for creating workflows with Alfred 2. This library provides an object-oriented class with functions for working with plist settings files, reading and writing data to files, generating Alfred feedback results, requesting remote data, and more.

It was originally created by David Ferguson using PHP, and was rewritten by me in AppleScript to provide the same functionality to all my fellow AppleScript lovers.

So you may be asking yourself:

- ***why on Earth would I use AppleScript when I already have PHP, Python, Ruby, Bash, etc.?*** – yes, it's true, Alfred can be scripted using all those languages, but ask yourself this: **are you able to control MacOS and its Apps using those languages?** I'm afraid not, and this is where AppleScript comes to help.
- ***but isn't it simpler to use my PHP / Python / etc. skills and combine them with AppleScript inside Alfred?*** Actually no, it isn't simpler – I've tried it, and it becomes really messy, not to mention that Alfred's workflow system doesn't allow that much mixing.

Known Limitations

Now, because AppleScript is a bit limited in terms of capabilities, some functionality isn't available right now, but I will try to improve this library further.

- **no JSON support yet** – AppleScript doesn't know anything about JSON, but I'm already planning a JSON parser for AppleScript
- **bigger file size** – since AppleScript requires extra coding for text manipulation and object handling, the file size is a bit large compared to the PHP equivalent, and it will probably increase as I add new features to it
- **strict syntax for plist records** – it's known that AppleScript's records are a bit clumsy since they lack so many features, that's why when saving a list of records as a PList settings file you should adhere to the following strict record notation:

```
{
  {theKey:"someKeyName", theValue: "textValue"},
  {theKey:"mynum", theValue: 2},
  {theKey: "booltest", theValue: false},
  {theKey:"na", theValue: missing value}
}
```

Initialization

```
set workflowFolder to do shell script "pwd"
set wf to load script POSIX file (workflowFolder & "/workflow.scpt")
set wf to wf's new_workflow()
```

or by specifying a bundle name:

```
...
set wf to wf's new_workflow_with_bundle("com.mycompany.mybundlename")
```

Explanations:

- the first line determines the Alfred workflow's bundle path because this is where the "workflow.scpt" library should be placed in order to work
- the second line loads the library from the bundle's path assuming that you already placed it there
- the last line creates a new script object (the equivalent of a class in other languages) with all the required functionality
- since AppleScript doesn't support optional parameters, there are 2 constructors: `new_workflow()` with no parameters, which creates a new class that automatically fetches the bundle name from Alfred, and `new_workflow_with_bundle(<name>)`, which takes 1 parameter with the desired bundle name if none was specified in Alfred.

Methods

1. get_bundle()

Takes no parameter and returns the value of the bundle id for the current workflow. If no value is available, then `missing value` is returned.

Example:

```
wf's get_bundle()
```

output:

```
com.qlassiqa.iTunesRatings
```

2. get_data()

Takes no parameter and returns the value of the path to the storage directory for your workflow if it is available. Returns missing value if the value isn't available.

Example:

```
wf's get_data()
```

output:

```
/Users/qlassiqa/Library/Application Support/Alfred 2/Workflow
Data/com.qlassiqa.iTunesRatings/
```

3. get_cache()

Takes no parameter and returns the value of the path to the cache directory for your workflow if it is available. Returns missing value if the value isn't available.

Example:

```
wf's get_cache()
```

output:

```
/Users/qlassiq/Library/Caches/com.runningwithcrayons.Alfred-2/Workflow
Data/com.qlassiqa.iTunesRatings/
```

4. get_path()

Takes no parameter and returns the value of the path to the current folder for your workflow if it is available. Returns missing value if the value isn't available.

Example:

```
wf's get_path()
```

output:

```
/Users/qlassiqa/Dropbox/Public/Alfred2/Alfred.alfredpreferences/workflows/user.workflow.
3BA9A8FC-75DB-494F-926A-CE19221E1211/
```

5. get_home()

Takes no parameter and returns the value of the home path for the current user. Returns missing value if the value isn't available.

Example:

```
wf's get_path()
```

output:

```
/Users/qlassiqa
```

6. set_value(key, value, plistfile)

Save values to a specified plist. If the plist file doesn't exist, it will be created in the workflow's data folder, and if the plistfile parameter is `missing value` or an empty string, a default "settings.plist" file will be created.

If the first parameter is a record list then the second parameter becomes the plist file to save to. Or you could just ignore this and use the `set_values` method that takes only 2 parameters for this scenario (presented next).

If the first parameter is text, then it is assumed that the first parameter is the key, the second parameter is the value, and the third parameter is the plist file to save the data to.

Example:

```
# add a username key with a text value to a default "settings.plist" file
1. wf's set_value("username", "mike", "")

# add a key with a boolean value
2. wf's set_value("default", true, "")

# add a key with a number value to a specific plist file
3. wf's set_value("age", 23, "settings.plist")

# add a key with a real number value
4. wf's set_value("weight", 65.3, "settings.plist")

# doesn't add anything since missing value was passed
5. wf's set_value("none", missing value, "settings.plist")

# add a list with mixed values
6. wf's set_value("my first list", {1, 2, 3, "bob"}, "settings.plist")

# add a list with values and a sublist
7. wf's set_value("my second list", {1, 2, 3, {"bob", "anne"}}, "settings.plist")

# replace previously created key
8. wf's set_value("username", "john", "settings.plist")
```

output:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
    <key>age</key>
    <integer>23</integer>
    <key>default</key>
    <true/>
    <key>my first list</key>
    <array>
        <integer>1</integer>
        <integer>2</integer>
        <integer>3</integer>
        <string>bob</string>
    </array>
    <key>my second list</key>
```

```
    <array>
        <integer>1</integer>
        <integer>2</integer>
        <integer>3</integer>
        <array>
            <string>bob</string>
            <string>anne</string>
        </array>
    </array>
    <key>username</key>
    <string>john</string>
    <key>weight</key>
    <real>65.299999999999997</real>
</dict>
</plist>
```

7. set_values(listofrecords, plistfile)

Save a list of records to a specified plist. If the plist file doesn't exist, it will be created in the workflow's data folder, and if the plistfile parameter is `missing value` or an empty string, a default "settings.plist" file will be created.

Each record must adhere to the following notation:

`{theKey: "somekeyname", theValue: 13}`

Example:

```
set theList to {{theKey:"favcolor", theValue:"red"}, {theKey:"hobbies", theValue:
{"sports", "music"}}}
wf's set_values(theList, "settings.plist")
```

output:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
    <key>favcolor</key>
    <string>red</string>
    <key>hobbies</key>
    <array>
        <string>sports</string>
        <string>music</string>
    </array>
</dict>
</plist>
```

8. get_value(key, plistfile)

Read a value from the specified plist. Note that if the plist file cannot be located, the script will automatically create an empty plist file in the data folder of the workflow. Also, if the plistfile parameter is `missing value` or an empty string, a default "settings.plist" file will be created.

Example:

```
# get a simple value from the default "settings.plist" file
1. wf's get_value("username", "")

# get a simple list (see example plist on method [5])
2. wf's get_value("my first list", "settings.plist")

# get a list containing a sublist (see example plist on method [5])
3. wf's get_value("my second list", "settings.plist")
```

output:

```
1. mike
2. {1,2,3,"bob"}
3. {1,2,3,{"bob","anne"}}
```

9. request(url)

Read data from a remote file/url, essentially a shortcut for curl

Example:

```
# get a json file from twitter
1. wf's request("http://mysite.com/search.json?q=json&rpp=5")

# get the contents of a website
2. wf's request("http://www.adobe.com/")
```

output:

```
1. {
    "name": "JSON.sh",
    "version": "0.1.4",
    "description": "JSON parser written in bash",
    "homepage": "http://github.com/dominictarr/JSON.sh",
    "repository": {
        "type": "git",
        "url": "https://github.com/dominictarr/JSON.sh.git"
    },
    "bin": {
        "JSON.sh": "./JSON.sh"
    },
    "dependencies": {},
    "devDependencies": {},
    "author": "Dominic Tarr",
    "scripts": { "test": "./all-tests.sh" }
}


2. the raw html contents of Adobe's website
```

10. mdfind(query)

Allows searching the local hard drive using mdfind, and returns a list of all found paths.

Example:

```
wf's mdfind("php.ini")
```

output:

```
{
    "/private/etc/php.ini.default",
    "/usr/local/php/lib/php.ini",
    ...
}
```

11. write_file(textorlist, cachefile)

Accepts data and a string file name to store data to local file. Each call to this method will overwrite the file if it already exists.

Note: due to AppleScript's lack of JSON support, this method can write to file only a piece of text, a value that can be converted to text, or a list that doesn't contain sublists or records.

Example:

```
1. wf's write_file("testing" & return & "string", "test.dat")
2. wf's write_file(12.5, "test.dat")
```

output:

```
1. testing
string

2. 12.5
```

12. read_file(cachefile)

Returns data from a local cache file, or missing value if the file doesn't exist. Note that if the file exists but is empty, it will be automatically deleted to clean up the workflow folder.

Example:

```
wf's write_file(12.5, "test.dat")
wf's read_file("test.dat")
```

output:

```
"12.5"
```

13. get_result

Creates a new result item that is cached within the class object. This set of results is available via the get*results() functions, or, can be formatted and returned as XML via the to*xml() function.

Note: this method uses the labeled parameter syntax in AppleScript (see example), and takes the following 'camelCase' parameters:

- **theUid**: the uid attribute is a value that is used to help Alfred learn about your results. You know that Alfred learns based on the items you use the most. That same mechanism can be used in feedback results. Give your results a unique identifier and Alfred will learn which ones you use the most and prioritize them by moving them up in the result list
- **theArg**: the arg attribute is the value that is passed to the next portion of the workflow when the result item is selected in the Alfred results list. So if you pressed enter on a result, the arg value would be passed to a shell script, applescript, or any of the other Action items
- **theTitle**: the title element is the value that is shown in large text as the title for the result item. This is the main text/title shown in the results list
- **theSubtitle**: the subtitle element is the value shown under the title in the results list. When performing normal searches within Alfred, this is the area where you would normally see the file path
- **theAutocomplete**: the autocomplete attribute is only used when the valid attribute has been set to `false`. When attempting to action an item that has the valid attribute set to 'no' and an autocomplete value is specified, the autocomplete value is inserted into the Alfred window. When using this attribute, the arg attribute is ignored
- **theType**: the type attribute allows you to specify what type of result you are generating. Currently, the only value available for this attribute is "file". This will allow you to specify that the feedback item is a file and allows you to use Result Actions on the feedback item
- **theIcon**: the icon element allows you to specify the icon to use for your result item. This can be a file located in your workflow directory, an icon of a file type on your local machine, or the icon of a specific file on your system. To use the icons of a specific file type you use this syntax `"filetype:public.folder"`. To use the icons of another folder/file you use this syntax `"fileicon:/Applications"`. To use an icon inside a subfolder located within the workflow you use this syntax `"subfolder/icon.png"`.

Example: (the following code was separated on different lines for easier reading, but should be written on a single line or separated using the "¬" AppleScript reserverd character.

```
1. get_result of wf without isValid given
    theUid:"alfred",
    theArg:"alfredapp",
    theTitle:"Alfred",
    theAutocomplete:"Alfred",
    theSubtitle:"/Applications/Alfred.app",
    theIcon:"fileicon:/Applications/Alfred 2.app",
    theType:"Alfredapp"
```

```
2. get_result of wf with isValid given
    theUid:"r9996",
    theArg:5,
    theTitle:"Alfred",
    theSubtitle:"",
    theAutocomplete:missing value,
    theIcon:"icon.png",
    theType:missing value
```

output:

```
1. {theUid:"alfred", theArg:"alfredapp", theTitle:"Alfred",
theSubtitle:"/Applications/Alfred.app", theIcon:"fileicon:/Applications/Alfred 2.app",
isValid:false, theAutocomplete:"Alfred", theType:"Alfredapp"}

2. {theUid:"r9996", theArg:5, theTitle:"Alfred", theSubtitle:"", theIcon:"icon.png",
isValid:true, theAutocomplete:"", theType:missing value}
```

Note: any of the above parameters can accept empty strings or missing values.

14. get_results()

Returns a list of available result items from the class' internal cache.

Example:

```
get_result of wf without isValid given theUid:"alfred", theArg:"alfredapp",
theTitle:"Alfred", theAutocomplete:"Alfred", theSubtitle:"/Applications/Alfred.app",
theIcon:"fileicon:/Applications/Alfred 2.app", theType:"Alfredapp"

get_result of wf with isValid given theUid:"r9996", theArg:5, theTitle:"Alfred",
theSubtitle:"", theAutocomplete:missing value, theIcon:"icon.png", theType:missing
value

wf's get_results()
```

output:

```
{
    {theUid:"alfred", theArg:"alfredapp", theTitle:"Alfred",
theSubtitle:"/Applications/Alfred.app", theIcon:"fileicon:/Applications/Alfred 2.app",
isValid:false, theAutocomplete:"Alfred", theType:"Alfredapp"},
    {theUid:"r9996", theArg:5, theTitle:"Alfred", theSubtitle:"", theIcon:"icon.png",
isValid:true, theAutocomplete:"", theType:missing value}
}
```

15. to_xml(listofrecords)

Convert a list of records into XML format. Passing an empty string or `missing value` as the parameter will make the method use the class' internal cache results as the list of records (this is built using the `get_result` method).

Example:

```
get_result of wf without isValid given theUid:"alfred", theArg:"alfredapp",
theTitle:"Alfred", theAutocomplete:"Alfred", theSubtitle:"/Applications/Alfred.app",
theIcon:"fileicon:/Applications/Alfred 2.app", theType:"Alfredapp"

get_result of wf with isValid given theUid:"r9996", theArg:5, theTitle:"Alfred",
theSubtitle:"", theAutocomplete:missing value, theIcon:"icon.png", theType:missing
value

wf's to_xml("")
```

output:

```
<?xml version="1.0"?>
<items>
    <item uid="alfred" arg="alfredapp" valid="no" autocomplete="Alfred"
type="Alfredapp">
        <title>Alfred</title>
        <subtitle>/Applications/Alfred.app</subtitle>
        <icon type="fileicon">/Applications/Alfred 2.app</icon>
    </item>
    <item uid="r9996" arg="5">
        <title>Alfred</title>
        <subtitle></subtitle>
        <icon>icon.png</icon>
    </item>
</items>
```