Python - Tips and Tricks for working with arcpy mapping

jack Horton
Learn how to use the arcpy.mapping module to:

- Alter the contents of map documents such as title, layers, and layouts
- Repair your MXD files when your data is moved
- Work live on your current map document or work on MXD files
- Process multiple MXD files

... and see that ArcGIS Pro has very similar functionality in its arcpy.mp module
We import `arcpy.mapping`, so it is now enabled, and we have a variable named `am` to quickly reference it.
Our am variable can get to lots of things...
... including a map document
MapDocument (arcpy.mapping)

**Summary**
Provides access to map document (.mxd) properties and methods. A reference to this object is essential for most map scripting operations.

**Discussion**
For a more complete discussion refer to the MapDocument Class help.

**Syntax**
MapDocument (mxd_path)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxd_path</td>
<td>A string that includes the full path and file name of an existing map document (.mxd) or a string that contains the keyword CURRENT.</td>
<td>String</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapDocument</td>
<td>The MapDocument object provides access to map document properties and methods. A reference to this object is essential for most map scripting operations.</td>
</tr>
</tbody>
</table>
The `MapDocument` method takes an `mxd_path` and it returns a `MapDocument` Object.

**Syntax**

```
MapDocument (mxd_path)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mxd_path</code></td>
<td>A string that includes the full path and file name of an existing map document (.mxd) or a string that contains the keyword CURRENT.</td>
</tr>
</tbody>
</table>

**Data Type**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MapDocument</code></td>
<td>The <code>MapDocument</code> object provides access to map document properties and methods. A reference to this object is essential for most map scripting operations.</td>
</tr>
</tbody>
</table>
### Syntax

**MapDocument (mxd_path)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxd_path</td>
<td>A string that includes the full path and file name of an existing map document (.mxd) or a string that contains the keyword CURRENT.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapDocument</td>
<td>The MapDocument object provides properties and methods. A reference most map scripting operations.</td>
</tr>
</tbody>
</table>

You can provide the pathname to an .mxd file, or you can use “CURRENT” to work live with the current map document.
The help shows examples of each technique.
We can use our am variable in place of arcpy.mapping to save time.
Our new variable mxd can get to lots of things about the map document.
For example, the title property is the map document title.
Title is a Map Document Property
It is settable in the Map Document Properties Dialog, and it is used in the Title map element.
Set the title property in Python, and it changes in ArcMap!
Set the title property in ArcMap, and it changes in Python!

```python
>>> import arcpy.mapping as am
>>> mxd = am.MapDocument("CURRENT")
>>> mxd.title = "Invasive Plant Map"
>>> mxd.title
u'San Juan National Forest Invasive Plant Map'
```
The map title is Dynamic Text, displaying the Title property of the Document.
There are lots of other useful map document properties, such as dateSaved.
You can read `dateSaved`, but you cannot modify it, because it is a Read Only property.
The help documentation for MapDocument tells you which properties are Read Only and which are Read Write.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateSaved (Read Only)</td>
<td></td>
<td>Returns a Python datetime object that reports the date the last time the map document was saved.</td>
</tr>
<tr>
<td>description (Read and Write)</td>
<td></td>
<td>Provides the ability to either get or set the map document’s description information.</td>
</tr>
<tr>
<td>filePath (Read Only)</td>
<td></td>
<td>Returns a string value that reports the fully qualified map document path and file name.</td>
</tr>
<tr>
<td>hyperlinkBase (Read and Write)</td>
<td></td>
<td>Provides the ability to either get or set the base path or URL used for field-based hyperlinks to documents or URLs.</td>
</tr>
</tbody>
</table>
Another thing you can do with a map document is list its layout elements.

Syntax:

```
ListLayoutElements (map_document, {element_type}, {wildcard})
```

- **map_document**: A variable that references a `MapDocument` object.
- **element_type**: A string that represents the element type that will be used to filter the returned list of elements.
  - DATAFRAME_ELEMENT — Dataframe element
  - GRAPHIC_ELEMENT — Graphic element
  - LEGEND_ELEMENT — Legend element
  - MAPSURROUND_ELEMENT — Mapsurround element
  - PICTURE_ELEMENT — Picture element
  - TEXT_ELEMENT — Text element
- **wildcard**: A combination of asterisks (*) and characters can be used to help limit the results.
  (The default value is None)

**Data Type**

- **Object**: A Python list of page layout elements. The types of objects that can be returned are: `DataFrame`, `GraphicElement`, `LegendElement`, `MapsurroundElement`, `PictureElement`, and `TextElement`. 

**Data Type**

- **String**: A string used for filtering element types.
Let's experiment with one of the samples

This script will search all text elements, including elements in a group, that have a text value of old string and replace that value with new string.

```python
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT"
    if elm.text == "Old String":
        elm.text = "New String"

mxd.save()
del mxd
```

ListLayoutElements example 2:
The following script will find a picture element using a wildcard and then change the picture's data source.

```python
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd,"PICTURE_ELEMENT"
    if elm.name == "CityLogo"
        elm.sourceImage = r"C:\Project\Data\Photo.bmp"

mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```
This script will search all text elements, including elements in a group, that have a text value of old string and replace that value with new string.

```python
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT ELEMENT"):
    if elm.text == "Old String":
        elm.text = "New String"
mxd.save()
del mxd
```

ListLayoutElements example 2:
The following script will find a picture element using a wildcard and then change the picture's data source.

```python
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd,"PICTURE_ELEMENT"):
    if elm.name == "CityLogo":
        elm.sourceImage = r"C:\Project\Data\Photo.bmp"
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT"):
    if elm.text == "Old String":
        elm.text = "New String"

mxd.save()
del mxd

... and paste to Notepad or your favorite Python editor. We need to modify this sample before we use it.
For example, our mxd variable is already set

```python
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT"):
    if elm.text == "Old String":
        elm.text = "New String"

mxd.save()
del mxd
```
for elm in arcpy.mapping.ListLayoutElements(mxd, 'TEXT_ELEMENT'):
    if elm.text == "Old String":
        elm.text = "New String"

mxd.save()
del mxd

... and lets list all the elements, not just TEXT_ELEMENTs
... And if we are going to list all the elements, most of them will not have text properties

for elm in arcpy.mapping.ListLayoutElements(mxd):
    if elm.text == "Old String":
        elm.text = "New String"

mxd.save()
delete mxd
for elm in arcpy.mapping.ListLayoutElements(mxd):
    mxd.save()
del mxd
Every element has a name and a type, so let's print them out.

```python
for elm in arcpy.mapping.ListLayoutElements(mxd):
    print(elm.name + " - " + elm.type)
mxd.save()
del mxd
```
And we do not need to save the map document at this time.
Deleting the `mxd` variable does not delete the map document. It just releases Python’s hold on it and clears the lock. We don’t want to do that either.
for elm in arcpy.mapping.ListLayoutElements(mxd):
    print(elm.name + " - " + elm.type)
The text wraps around in the window. We pasted a for loop, so we need to hit enter twice. That way the Python window knows we are done adding lines inside the loop.
Lets widen the Python window so we can see better.
We have five elements on the layout:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>North Arrow</td>
<td>MAPSURROUND_ELEMENT - TEXT_ELEMENT</td>
</tr>
<tr>
<td>Alternating Scale Bar</td>
<td>MAPSURROUND_ELEMENT</td>
</tr>
<tr>
<td>overview</td>
<td>DATAFRAME_ELEMENT</td>
</tr>
<tr>
<td>San Juan</td>
<td>DATAFRAME_ELEMENT</td>
</tr>
</tbody>
</table>

The name property of the text element does not have anything in it.
Let's shift the North Arrow 1.7 page inches to the left.
As you can see, only the North Arrow shifted ...
... and it shifted 1.7 page inches to the left
Let's list the names of just the dataframes.

```python
>>> for elm in arcpy.mapping.ListLayoutElements(mxd):
...     if elm.name == "North Arrow":
...         elm.elementPositionX = elm.elementPositionX - 1.7
...     for elm in arcpy.mapping.ListLayoutElements(mxd):
...         if elm.type == "DATAFRAME_ELEMENT":
...             print("Dataframe: " + elm.name)
```

Dataframe: overview
Dataframe: San Juan

Let's list the names of just the dataframes.
Now let's list the layers in each dataframe, using these variables:

- `elm`: Each dataframe
- `lay`: Each layer
- `mxd`: The map document
- `am`: `arcpy.mapping`

```python
>>> for elm in arcpy.mapping.ListLayoutElements(mxd):
...     if elm.type == "DATAFRAME_ELEMENT":
...         print("Dataframe: " + elm.name)
...     for lay in am.ListLayers(mxd, "", elm):
...         print(" Layer: " + lay.name)
```
There are many kinds of layers and they have many different properties. Not all layers support all properties.
The supports method lets you test ahead of time and avoid errors.
For example, a basemap layer does not have a workspace path. Let's modify our script to print out the workspace path of each layer, but only if it has one!

```python
for elm in arcpy.mapping.ListLayoutElements(mxd):
    if elm.type == "DATAFRAME_ELEMENT":
        print("Dataframe: " + elm.name)
        for lay in am.ListLayers(mxd, "", elm):
            print("   Layer: " + lay.name)

for elm in arcpy.mapping.ListLayoutElements(mxd):
    if elm.type == "DATAFRAME_ELEMENT":
        print("Dataframe: " + elm.name)
        for lay in am.ListLayers(mxd, "", elm):
            if lay.supports("WORKSPACEPATH"):
                print("   Layer: " + lay.name)
                print("     Path: " + lay.workspacePath)
```
For our example, layers have different workspaces, but they are all in the D:\temp folder.
Let's point the layers to a different folder.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_workspace_path</td>
<td>A string that represents the workspace path or connection file you want to find. If an empty string is passed, then all workspace paths will be replaced with the replace_workspace_path parameter depending on the value of the validate parameter.</td>
<td>String</td>
</tr>
<tr>
<td>replace_workspace_path</td>
<td>A string that represents the workspace path or connection file you want to replace.</td>
<td>String</td>
</tr>
<tr>
<td>validate</td>
<td>If set to True, the workspace will only be updated if the replace_workspace_path value is a valid workspace. If it is not valid, the workspace will not be replaced. If set to False, the method will set the workspace to match the replace_workspace_path, regardless of a valid match. In this case, if a match does not exist, then the layer's data source would be broken. (The default value is True)</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the [Updating and fixing data sources with arcpy.mapping help topic](#).
import arcpy.mapping as am
mxd = am.MapDocument("CURRENT")
for elm in arcpy.mapping.ListLayoutElements(mxd):
    if elm.type == "DATAFRAME_ELEMENT":
        print("Dataframe: " + elm.name)
        for lay in am.ListLayers(mxd, "", elm):
            if lay.supports("WORKSPACEPATH"):
                print("Layer: " + lay.name)
                print("Path: " + lay.workspacePath)

    oldpath = lay.workspacePath
    newpath = oldpath.replace("D:\temp", "D:\temp\testing")
    lay.findAndReplaceWorkspacePath(oldpath, newpath)
    print("Path: " + lay.workspacePath)
    print("")

Notes:
1. The \ is needed by Python because \ is a special character.
2. replace is a Python function to replace the contents of strings
The paths have all been changed.

Layer: index
Path: D:\temp\Database
Path: D:\temp\testing\Database

Layer: Forest boundary
Path: D:\temp\Database
Path: D:\temp\testing\Database

Layer: States
Path: D:\temp\Database\San_Juan.gdb
Path: D:\temp\testing\Database\San_Juan.gdb
Now let's use `ListFiles()` to fix all the .mxd files in a folder.
Optionally, use `da.Walk` to fix all the `.mxd` files in an entire directory tree.
We need to run it outside of ArcMap. Use any IDE.
import arcpy
arcpy.env.workspace = "D:\\temp\\mxds to fix"
MXDFiles = arcpy.ListFiles("*.mxd")
for MXDFile in MXDFiles:
x = am.MapDocument(MXDFile)
print('
')
for elm in arcpy.mapping.ListLayoutElements(x):
    if elm.type == "DATAFRAME_ELEMENT":
        print("DataFrame: "+elm.name)
        for lay in am.ListLayers(x, "", elm):
            if lay.supports("WORKSPACEPATH"):

Add these lines get each .mxd file
Indent your original script inside the for loop.
And save each mxd as it is processed

```python
arcpy.env.workspace = "D:\temp\mxds to fix"
MXDFiles = arcpy.ListFiles("*.mxd")
for MXDFile in MXDFiles:
    mxd = am.MapDocument(MXDFile)
    print(""
    print("---------- Processing " + MXDFile + " ----------")
    for elm in arcpy.mapping.ListLayoutElements(mxd):
        if elm.type == "DATAFRAME_ELEMENT":
            print ("Dataframe: " + elm.name)
            for lay in am.ListLayers(mxd, "", elm):
                if lay.supports("WORKSPACEPATH"):
                    print(" Layer: " + lay.name)
                    print(" Path: " + lay.workspacePath)
                    oldpath = lay.workspacePath
                    newpath = oldpath.replace("D:\temp", "D:\temp\testing")
                    lay.findAndReplaceWorkspacePath (oldpath, newpath)
                    print (" Path: " + lay.workspacePath)
    mxd.save()
    print("Done")
```
Review

- Use “CURRENT” as the MapDocument to work live in ArcMap
  - Test live in ArcMap and save to a script file
- Use the MXD file name as the MapDocument to work on MXD files
- Read and sometimes write document properties (Title, dates, etc.)
- Work with elements in a layout
- Work with map layers
- Set data sources on many MXD files at once using ListFiles or da.Walk
ArcPy.Mapping in Pro

Similar functionality, but Pro is different

- Python 3.5 instead of 2.7
- arcpy.mp replaces arcpy.mapping
- .aprx file replaces .mxd
- Multiple layouts
- Map, MapFrame, and Camera objects replace the Data Frame
Migrating from arcpy.mapping to ArcGIS Pro

- Python 3
- arcpy.mapping is now arcpy.mp
- The ArcGIS Pro project file (.aprx)
- Many of the list functions have moved
- Export functions have moved
- Layer management functions have moved
- Layer files have changed
- New Map, MapFrame, and Camera objects replace the role of the data frame
- A new Layout object
- The application always refreshes when using CURRENT
- Updating data sources has changed

Arcpy.mapping scripts authored with ArcGIS Desktop will need to be modified before they will run in ArcGIS Pro. The changes are straightforward and logical and can be accomplished mostly with find and replace operations. The sections below highlight many of the significant changes to the arcpy.mp API as well as new features that were added.