ArcPy.Mapping and ArcPy.Time Reference

This is an offline reference of the following ESRI documentation:

http://resources.arcgis.com/en/help/main/10.2/index.html#/alphabetical_list_of_arcpy_mapping_classes/00s30000000t000000/
http://resources.arcgis.com/en/help/main/10.2/index.html#/alphabetical_list_of_arcpy_mapping_functions/00s300000010000000/
http://resources.arcgis.com/en/help/main/10.2/index.html#/alphabetical_list_of_arcpy_mapping_constants/00s30000002p000000/
http://resources.arcgis.com/en/help/main/10.2/index.html#/what_is_the_time_module/01vm00000003000000/

ArcPy.Mapping Sections:

ArcPy.Mapping Classes

ArcPy.Mapping Functions

ArcPy.Mapping Constants

ArcPy.Time Sections:

ArcPy.Time Classes and Functions

ArcPy.Mapping Classes

DataDrivenPages

Methods

- exportToPDF(out_pdf, {page_range_type}, {page_range_string}, {multiple_files}, {resolution}, {image_quality}, {colorspace}, {compress_vectors}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality}, {show_selection_symbology})
- getPageIDFromName(page_name)
- printPages({printer_name}, {page_range_type}, {page_range_string}, {out_print_file}, {show_selection_symbology})
- refresh()

Properties

- currentPageID—Read/Write; Long
- dataFrame—Read-only; DataFrame object
- indexLayer—Read-only; Layer object
- pageCount—Read-only; Long
- pageNameField—Read-only; GP Field object
- pageRow—Read-only; GP Row object
- selectedPages—Read-only; Python list of index numbers

DataFrame

Methods

- panToExtent(extent)
- zoomToSelectedFeatures()

Properties

- credits—Read/Write; String
- description—Read/Write; String
- displayUnits—Read/Write; String
- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- extent—Read/Write; GP Extent object
- geographicTransformations—Read/Write; Python list of strings
- mapUnits—Read-only; String
- name—Read/Write; String
- referenceScale—Read/Write; Double
- rotation—Read/Write; Double
- scale—Read/Write; Double
- spatialReference—Read/Write; GP Spatial Reference object
- time—Read-only; DataFrameTime object
- type—Read-only; String

DataFrameTime

Methods

- resetTimeExtent()

Properties

- currentTime—Read/Write; Python datetime object
- endTime—Read/Write; Python datetime object
- startTime—Read/Write; Python datetime object
- timeStepInterval—Read-only; Python timedelta object
- timeWindow—Read/Write; Double
- timeWindowUnits—Read/Write; String

GraduatedColorsSymbology

Methods

- reclassify()

Properties

- classBreakDescriptions—Read/Write; Python list of strings
- classBreakLabels—Read/Write; Python list of strings
- classBreakValues—Read/Write; Python list of strings
- normalization—Read/Write; String
- numClasses—Read/Write; Long
- valueField—Read/Write; String

GraduatedSymbolsSymbology

Methods

- reclassify()

Properties

- classBreakDescriptions—Read/Write; Python list of strings
- classBreakLabels—Read/Write; Python list of strings
- classBreakValues—Read/Write; Python list of strings
- normalization—Read/Write; String
- numClasses—Read/Write; Long
- valueField—Read/Write; String

GraphicElement

Methods

- clone({suffix})
- delete()

Properties

- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- isGroup— Read-only; Boolean
- name—Read/Write; String
- type—Read-only; String

LabelClass

Properties

- className—Read/Write; String
- expression—Read/Write; String
- showClassLabels—Read/Write; Boolean
- SQLQuery—Read/Write; String

Layer

Methods

- findAndReplaceWorkspacePath(find_workspace_path, replace_workspace_path, {validate})
- getExtent({symbolized_extent})
- getSelectedExtent({symbolized_extent})
- replaceDataSource(workspace_path, workspace_type, dataset_name, {validate})
- save()
- saveACopy(file_name, {version})
- supports(layer_property)

Properties

- brightness—Read/Write; Long
- contrast—Read/Write; Long
- credits—Read/Write; String
- datasetName—Read-only; String
- dataSource—Read-only; String
- definitionQuery—Read/Write; String
- description—Read/Write; String
- isBroken—Read-only; Boolean
- isFeatureLayer—Read-only; Boolean
- isGroupLayer—Read-only; Boolean
- isNetworkAnalystLayer—Read-only; Boolean
- isRasterizingLayer—Read-only; Boolean
- isRasterLayer—Read-only; Boolean
- isServiceLayer—Read-only; Boolean
- labelClasses—Read/Write; List of LabelClass objects
- longName—Read-only; String
- maxScale—Read/Write; Double
- minScale—Read/Write; Double
- name—Read/Write; String
- serviceProperties—Read-only; Dictionary of property sets
- showLabels—Read/Write; Boolean
- symbology—Read-only; Layer symbology object
- symbologyType—Read-only; String
- time—Read-only; Layer time object
- transparency—Read/Write; Long
- visible—Read/Write; Boolean
- workspacePath—Read-only; String

LayerTime

Properties

- daylightSavings—Read-only; Boolean
- displayDataCumulatively—Read-only; Boolean
- endTime—Read-only; Python datetime object
- endTimeField—Read-only; String
- isTimeEnabled—Read-only; Boolean
- startTime—Read-only; Python datetime object
- startTimeField—Read-only; String
- timeFormat—Read-only; String
- timeOffset—Read-only; Python timedelta object
- timeStepInterval—Read-only; Python timedelta object
- timeZone—Read-only; String

Next Page

LegendElement

Methods

- adjustColumnCount(column_count)
- listLegendItemLayers()
- removeItem(legend_item_layer, {index})
- updateItem(legend_item_layer, {legend_item_style_item}, {preserve_item_sizes}, {use_visible_extent}, {show_feature_count}, {use_ddp_extent}, {index})

Properties

- autoAdd—Read/Write; Boolean
- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- isOverflowing—Read-only; Boolean
- items—Read-only; Python list of strings
- name—Read/Write; String
- parentDataFrameName—Read-only; String
- title—Read/Write; String
- type—Read-only; String

MapDocument

Methods

- deleteThumbnail()
- findAndReplaceWorkspacePaths(find_workspace_path, replace_workspace_path, {validate})
- makeThumbnail()
- replaceWorkspaces(old_workspace_path, old_workspace_type, new_workspace_path, new_workspace_type, {validate})
- save()
- saveACopy(file_name, {version})

Properties

- activeDataFrame—Read-only; DataFrame object
- activeView—Read/Write; String
- author—Read/Write; String
- credits—Read-only; String
- dataDrivenPages—Read-only; DataDrivenPages object
- dateExported—Read-only; Python datetime object
- datePrinted—Read-only; Python datetime object
- dateSaved—Read-only; Python datetime object
- description—Read/Write; String
- filePath—Read-only; String
- hyperlinkBase—Read/Write; String
- isDDPEnabled—Read-only; Boolean
- pageSize—Read-only; Python named tuple
- relativePaths—Read/Write; Boolean
- summary—Read/Write; String
- tags—Read/Write; String
- title—Read/Write; String

MapsurroundElement

Properties

- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- name—Read/Write; String
- parentDataFrameName—Read-only; String
- type—Read-only; String

PDFDocument

Methods

- appendPages(pdf_path, {input_pdf_password})
- attachFile(file_path, {description})
- deletePages(page_range)
- insertPages(pdf_path, {before_page_number}, {input_pdf_password})
- saveAndClose()
- updateDocProperties({pdf_title}, {pdf_author}, {pdf_subject}, {pdf_keywords}, {pdf_open_view}, {pdf_layout})
- updateDocSecurity({new_master_password}, {new_user_password}, {encryption}, {permissions})

Properties:

- pageCount—Read-only; Long

PictureElement

Properties

- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- name—Read/Write; String
- sourceImage—Read/Write; String
- type—Read-only; String

RasterClassifiedSymbology

Methods

- reclassify()

Properties

- classBreakDescriptions—Read/Write; Python list of strings
- classBreakLabels—Read/Write; Python list of strings
- classBreakValues—Read/Write; Python list of strings
- excludedValues—Read/Write; String
- normalization—Read/Write; String
- numClasses—Read/Write; Long
- valueField—Read/Write; String

StyleItem

Properties

- itemName—Read-only; String
- itemCategory—Read-only; String
- styleFolderName—Read-only; String

TableView

Methods

- findAndReplaceWorkspacePath(find_workspace_path, replace_workspace_path, {validate})
- replaceDataSource(workspace_path, workspace_type, dataset_name, {validate})

Properties

- datasetName—Read-only; String
- dataSource—Read-only; String
- definitionQuery—Read/Write; String
- isBroken—Read-only; Boolean
- name—Read/Write; String
- workspacePath—Read-only; String

TextElement

Methods

- clone({suffix})
- delete()

Properties

- elementHeight—Read/Write; Double
- elementPositionX—Read/Write; Double
- elementPositionY—Read/Write; Double
- elementWidth—Read/Write; String
- fontSize—Read/Write; Double
- name—Read/Write; String
- text—Read/Write; String
- type—Read-only; String

UniqueValuesSymbology

Methods

- addAllValues()

Properties

- classDescriptions—Read/Write; Python list of strings
- classLabels—Read/Write; Python list of strings
- classValues—Read/Write; Python list of strings
- showOtherValues—Read/Write; Boolean
- valueField—Read/Write; String

Previous Page

DataDrivenPages (arcpy.mapping)

Top

Summary

Provides access to methods and properties for managing the individual pages within a map document that has Data Driven Pages enabled.

Discussion

Map series can be created without any scripting at all by using the ***Data Driven Pages*** toolbar from within ArcMap. The reverse is also true: map series can be completely scripted using arcpy.mapping without using the Data Driven Pages user interface in ArcMap, but there are good reasons for combining both techniques. The ArcMap Data Driven Pages toolbar may not provide enough options for creating the "perfect" map series, but the inherent behavior of a Data Driven Pages-enabled map document can save many lines of code because the page extents, scales, dynamic text, and so forth, are all managed automatically within the map document so that code does not need to be written.

An example of this would be a scenario where a text element's string information needs to be formatted using custom logic or needs to be constructed from multiple fields. It would be unnecessary to have to do everything in arcpy.mapping just because of such a simple, custom requirement. In this case, the map document can still be authored with Data Driven Pages enabled, and arcpy.mapping can handle the custom text element string requirements. A code sample below addresses this scenario.

Data Driven Pages must first be enabled and authored within a map document (`.mxd`) using the Data Driven Pages toolbar in ArcMap before it can be referenced with arcpy.mapping.

The **`DataDrivenPages`** class only has a single **`exportToPDF`** method but that does not mean other export files can't be created. See the first code sample below.

For more information about Data Driven Pages, see the following topics:

- Building map books with ArcGIS
- Creating Data Driven Pages

Properties

Property	Explanation	Data Type
currentPageID (Read and Write)	The **`currentPageID`** property represents the active or current page for a map document (`.mxd`) that has Data Driven Pages enabled. This value is displayed in the Data Driven Pages toolbar when Show Page is selected; it represents the x of x of y.	Long
dataFrame (Read Only)	Returns a reference to the data frame the index layer resides within a Data Driven Pages enabled map document.	DataFrame
indexLayer (Read Only)	Returns a reference to the index layer in a Data Driven Pages enabled map document.	Layer
pageCount (Read Only)	The **`pageCount`** property returns the total page count for a map document (`.mxd`) that has Data Driven Pages enabled. This value is displayed in the Data Driven Pages toolbar when Show Page is selected; it represents the y of x of y.	Long
pageNameField (Read Only)	Returns a field object that represents the field used in the index feature class when setting up Data Driven Pages.	Field
pageRow (Read Only)	Use **`pageRow`** to return the index layer's row object for the active or current page. The index layer fields can then be read and/or modified as necessary.	Row
selectedPages (Read Only)	Returns a Python list of index numbers that represent selected index layer features in a Data Driven Pages enabled map document.	List

Method Overview

Method	Explanation
exportToPDF (out_pdf, {page_range_type}, {page_range_string}, {multiple_files}, {resolution}, {image_quality}, {colorspace}, {compress_vectors}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality}, {show_selection_symbology})	Exports a specified set of pages to a multipage PDF document for a map document (`.mxd`) that has Data Driven Pages enabled
getPageIDFromName (page_name)	Returns a Data Driven Pages index value based on the name of the page
printPages ({printer_name}, {page_range_type}, {page_range_string}, {out_print_file}, {show_selection_symbology})	Prints specific pages from a Data Driven Pages-enabled map document (`.mxd`) to a specified printer
refresh ()	Refreshes an existing Data Driven Pages series

Methods

exportToPDF (out_pdf, {page_range_type}, {page_range_string}, {multiple_files}, {resolution}, {image_quality}, {colorspace}, {compress_vectors}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality}, {show_selection_symbology})

Parameter	Explanation	Data Type
out_pdf	A string that represents the path and file name for the output export file.	String
page_range_type	The string value that designates how the pages will be printed, similar to the **Pages** tab within the ArcMap ***Export Map*** dialog box for PDF documents. • ALL —All pages are exported. • CURRENT —The active page is exported. • RANGE —Only pages listed in the page_range_string parameter will be exported. • SELECTED —Selected index layer features/pages are exported. (The default value is ALL)	String
page_range_string	A string that identifies the pages to be printed if the RANGE option in the **`page_range_type`** parameter is used (for example, `1, 3, 5-12`).	String
multiple_files	An option to control how the output PDF is created. By default, all pages are exported into a single, multipage document. You can also specify that individual, single-page PDF documents be exported using two different options. • PDF_MULTIPLE_FILES_PAGE_NAME —Export single-page documents using the page name for the output file name. • PDF_MULTIPLE_FILES_PAGE_INDEX —Export single-page documents using the page index value for the output file name. • PDF_SINGLE_FILE —Export a multipage document. (The default value is PDF_SINGLE_FILE)	String
resolution	An integer that defines the resolution of the export file in dots per inch (dpi). (The default value is 300)	Integer
image_quality	A string that defines output image quality. • BEST —An output image quality resample ratio of 1 • BETTER —An output image quality resample ratio of 2 • NORMAL —An output image quality resample ratio of 3 • FASTER —An output image quality resample ratio of 4 • FASTEST —An output image quality resample ratio of 5 (The default value is BEST)	String
colorspace	A string that defines the color space of the export file. • CMYK —Cyan, magenta, yellow, and black color model • RGB —Red, green, and blue color model (The default value is RGB)	String
compress_vectors	A Boolean that controls compression of vector and text portions of the output file. Image compression is defined separately. (The default value is True)	Boolean
image_compression	A string that defines the compression scheme used to compress image or raster data in the output file. • ADAPTIVE —Automatically selects the best compression type for each image on the page. JPEG will be used for large images with many unique colors. DEFLATE will be used for all other images. • JPEG —A lossy data compression. • DEFLATE —A lossless data compression	String

	• LZW —Lempel-Ziv-Welch, a lossless data compression • NONE —Compression not applied • RLE —Run-length encoded compression (The default value is ADAPTIVE)	
picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture markers/fills. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String
convert_markers	A Boolean that controls the conversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to `True` disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean
embed_fonts	A Boolean that controls the embedding of fonts in an export file. Font embedding allows text and character markers to be displayed correctly when the document is viewed on a computer that does not have the necessary fonts installed. (The default value is True)	Boolean
layers_attributes	A string that controls inclusion of PDF layer and PDF object data (attributes) in the export file. • LAYERS_ONLY —Export PDF layers only. • LAYERS_AND_ATTRIBUTES —Export PDF layers and feature attributes. • NONE —No setting is applied. (The default value is LAYERS_ONLY)	String
georef_info	A Boolean that enables exporting of coordinate system information for each data frame into the output PDF file. (The default value is True)	Boolean
jpeg_compression_quality	A number that controls compression quality value when **`image_compression`** is set to ADAPTIVE or JPEG. The valid range is 1 to 100. A **`jpeg_compression_quality`** of 100 provides the best quality images but creates large export files. The recommended range is between 70 and 90.	Integer
	(The default value is 80)	
show_selection_symbology	A Boolean that controls whether the selection symbology should be displayed in the output. (The default value is False)	Boolean

Data Driven Pages are exported to a multipage PDF document. The map document must have Data Driven Pages enabled. PDF files are designed to be consistently viewable and printable across different platforms. They are commonly used for distributing documents on the Web and are becoming a standard interchange format for content delivery. ArcMap PDFs are editable in many graphics applications and retain annotation, labeling, and attribute data for map layers from the ArcMap table of contents. PDF exports from ArcMap support embedding of fonts and thus can display symbology correctly even if the user does not have Esri fonts installed. PDF exports from ArcMap can define colors in CMYK or RGB values.

Refer to the Exporting your map topic in ArcGIS for Desktop Help for more detailed discussions on exporting maps.

getPageIDFromName (page_name)

Parameter	Explanation	Data Type
page_name	A value in the index layer that corresponds to the **`Name`** field that was used to set up Data Driven Pages	String

Many of the Data Driven Pages properties and methods use an internal index value rather than the literal names of the pages used to create the index layer. The index values are automatically generated based on the **`Name`** and **`Sort`** fields. It may not be obvious which index value represents a specific page.

The **`getPageIDFromName`** method provides a mechanism for this translation.

```
pageID = mxd.dataDrivenPages.getPageIDFromName("HarborView")
mxd.dataDrivenPages.currentPageID = pageID
```

printPages ({printer_name}, {page_range_type}, {page_range_string}, {out_print_file}, {show_selection_symbology})

Parameter	Explanation	Data Type
printer_name	A string that represents the name of a printer on the local computer. (The default value is None)	String
page_range_type	The string value that designates how the pages will be printed, similar to the **Pages** tab within the ArcMap ***Export Map*** dialog box for PDF documents. • ALL —All pages are exported. • CURRENT —The active page is exported. • RANGE —Only pages listed in the page_range_string parameter will be exported. • SELECTED —Selected index layer features/pages are exported. (The default value is ALL)	String
page_range_string	A string that identifies the pages to be printed if the RANGE option in the **page_range_type** parameter is used (for example, `1, 3, 5-12`).	String
out_print_file	A path that includes the name of an output print file. The format created is dependent on the printer. If you're using a PostScript printer, the format will be PostScript, and it is recommended that a `.ps` extension be provided. If you're using a Windows printer, use a `.prn` extension. (The default value is None)	String
show_selection_symbology	A Boolean that controls whether the selection symbology should be displayed in the output. (The default value is False)	Boolean

The ListPrinterNames() function is an easy way to get the string for the **printer_name** parameter.

Note:

Driver based printing is not supported on ArcGIS for Server. For Data Driven Pages printing tasks as geoprocessing services, use the `exportToPDF` function in the`DataDrivenPages` class. For information on general printing in web applications see Printing in web applications.

The following script prints a specific set of Data Driven Pages to a local printer:

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\ParcelAtlas.mxd")
mxd.dataDrivenPages.printPages(r"\\olyfile\OLYCanon", "RANGE", "1,3,5-12")
```

refresh ()

You will want to use the **refresh** method if one of the following occurs: (1) features are added to or deleted from your index layer; (2) edits are made to the **Sort** or **Name** field values; (3) the data frame extent is changed due to zooming, panning, or change to map scale; or (4) edits are made to any field being used by Data Driven Pages for an index layer feature that is driving the current geographic extent. Data Driven Pages will retain the original settings in these cases until the **refresh** method is executed.

Code Sample

DataDrivenPages example 1

The following script exports each page of a Data Driven Pages series into an individual PNG file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\ParcelAtlas.mxd")
for pageNum in range(1, mxd.dataDrivenPages.pageCount + 1):
    mxd.dataDrivenPages.currentPageID = pageNum
    print "Exporting page {0} of {1}".format(str(mxd.dataDrivenPages.currentPageID), str(mxd.dataDrivenPages.pageCount))
    arcpy.mapping.ExportToPNG(mxd, r"C:\Project\OutPut\ParcelAtlas_Page" + str(pageNum) + ".png")
del mxd
```

DataDrivenPages example 2

The following script will print only a set of map pages using a list of page names and also modifies text element map title information using customized logic that can only be accomplished within the scripting environment (in other words, the title string is custom built based on an attribute value). The script loops through each named page and sets the currentPageID accordingly. It then extracts the value from a field in the index layer called `TRS`. It next parses the values, strips away leading zeros, reconstructs the text element title string, and sends the results to a printer.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\ParcelAtlas.mxd")
pageNameList = ["MPB", "PJB", "AFB", "ABB"]
for pageName in pageNameList:
    pageID = mxd.dataDrivenPages.getPageIDFromName(pageName)
    mxd.dataDrivenPages.currentPageID = pageID
    fieldValue = mxd.dataDrivenPages.pageRow.TSR  #example values from
a field called TSR are "080102", "031400"
    TRSTitle = arcpy.mapping.ListLayoutElements(MXD, "TEXT_ELEMENT",
"TRSTitle")[0]
    township, range, section = fieldValue[:2].strip("0"),
fieldValue[2:-2].strip("0"), fieldValue[-2:].strip("0")
    if section != "":
        TRSTitle.text = "Section {0} T.{1}N. R.{2}W.
W.M.".format(section, township, range)
    else:
        TRSTitle.text = "T.{0}N. R.{1}W. W.M.".format(township, range)
    mxd.dataDrivenPages.printPages(r"\\olyfile\SUITE_303", "CURRENT")
del mxd
```

DataDrivenPages example 3

The following script will export only the selected index pages (pages 1-10) out to individual PDF files. The resulting PDFs will have the index number appended to the output file name.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\ParcelAtlas.mxd")
ddp = mxd.dataDrivenPages
indexLayer = ddp.indexLayer
arcpy.SelectLayerByAttribute_management(indexLayer, "NEW_SELECTION",
                                        '"PageNumber" >= 1 AND
"PageNumber" <= 10')
for indexPage in ddp.selectedPages:
  ddp.currentPageID = indexPage
  ddp.exportToPDF(r"C:\Project\Output\Page" + str(indexPage) + ".pdf",
"CURRENT")

del mxd
```

DataFrame (arcpy.mapping)

Top

Summary

The **DataFrame** object provides access to many of the data frame properties found within a map document (`.mxd`). A reference to the **DataFrame** object is often used as an argument for many functions to filter layers or tables within a specific data frame.

Discussion

The **DataFrame** object provides access to important data frame properties. The ListDataFrames function returns a Python list of **DataFrame** objects. It is necessary to then iterate through each item in the list or to specify an index number to reference a specific **DataFrame** object. The object works with both map units and page units depending upon the property being used. For example, it can be used to set map extent, scale, and rotation, as well as items like spatial reference. The **DataFrame** object can also be positioned and/or sized on the layout using page units. The **DataFrame** object also provides access to informational items like credits and description.

A reference to a data frame can also be very useful when trying to reference other objects as well. For example, the ListLayers function provides an optional parameter called **data_frame** so that layers can be searched within a single data frame rather than the entire map document. The DataFrame object is also used as an optional parameter to distinguish printing or export a data frame versus a page layout. For this reason it is important to uniquely name each data frame so a specific data frame can be easily referenced.

The data frame **extent** coordinates are based on the extent of the data frame in Layout View, not in Data View. This is because the shape of the data frame in Data View may have a different aspect ratio than the data frame in Layout View. The **elementPositionX** and **elementPositionY** parameters are based on the element's anchor position which is set via the **Size and Position Tab** on the elements properties dialog box in ArcMap.
Page units can only be changed in the ArcMap via **Customize** > **ArcMap Options** > **Layout View Tab**.

Properties

Property	Explanation	Data Type
credits (Read and Write)	Provides the ability to either get or set the data frame credits information.	String
description (Read and Write)	Provides the ability to either get or set the data frame description information.	String
displayUnits (Read and Write)	Provides the ability to either get or set the data frame distance units.	String
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The X location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The Y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
extent (Read and Write)	Provides the ability to get or set the data frame's map extent using map coordinates (i.e., map units). A copy of the **Extent** object must be made before modifying its properties. The modified copy is then used to set the new extent properties. Note: If you try to set the extent by simply referencing the **Extent** object, changes won't be saved. For example, `df.extent.xMin = some value` won't work. If the aspect ratio of the extent does not match the shape of the data frame, the final extent will be adjusted to fit the new extent within the shape of the data frame. In other words, if you set explicit X, Y coordinates, you may not get the same values returned if you attempt to read them later. Note: The properties of the **Extent** object are by default read-only in the help system. A special exception was made for the arcpy.mapping scripting environment to enable changing extents during a map automation process.	Extent

	`` df = arcpy.mapping.ListDataFrames(mxd)[0] newExtent = df.extent newExtent.XMin, newExtent.YMin = -180.0, -90.0 newExtent.XMax, newExtent.YMax = 180.0, 90.0 df.extent = newExtent ```	
geographicTransformations (Read and Write)	Provides the ability to either get or set the data frame's geographic transformation(s). The property will return the name(s) of the transformation(s) in a list. Only existing, predefined transformation names (or their corresponding code value) can be used to set a geographic transformation. A complete list of transformations and code values can be found on the ArcGIS Resource Center. The **geographicTransformations** property cannot be used to create custom transformations. Only predefined methods can be referenced. There is always one geographic transformation loaded into a map document by default: NAD_1927_To_NAD_1983_NADCON. This will be overwritten when setting a new list. Geographic transformations can also be cleared by setting an empty list. The following examples will set two transformation methods using name strings. The first from NAD27 to NAD 83 and the second from NAD83 to HARN. The second example does the same thing but uses transformation codes instead. ``` df.geographicTransformations = [u'NAD_1927_To_NAD_1983_NADCON', u'NAD_1983_To_HARN_New_Jersey'] df.geographicTransformations = [1241,1554] ```	String
mapUnits (Read Only)	Returns a string value that reports the current data frame map units. These are based on the data frame's current coordinate system.	String
name	Provides the ability to either get or set the data frame's name as it appears in the table of contents in a map	String

(Read and Write)	document and also the actual name of the element within the layout.	
referenceScale (Read and Write)	Provides the ability to either get or set the data frame's reference scale. This is the scale in which all symbol and text sizes used in the data frame will be made relative.	Double
rotation (Read and Write)	Provides the ability to either get or set the data frame's rotation value. It represents the number of degrees by which the data frame will be rotated, measured counterclockwise from the north. To rotate clockwise, use a negative value.	Double
scale (Read and Write)	Provides the ability to either get or set the current scale of the active data frame. A numerical (double) value must be used.	Double
spatialReference (Read and Write)	Provides access to the **SpatialReference** of the data frame. The spatial reference contains information about the coordinate system and units.	SpatialReference
time (Read Only)	Returns the DataFrameTime object that provides access to controlling the display for time-enabled layers.	DataFrameTime
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Dataframe element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Mapsurround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Method Overview

Method	Explanation
panToExtent (extent)	Pans and centers the data frame extent using a new Extent object without changing the data frame's scale
zoomToSelectedFeatures ()	Changes the data frame extent to match the extent of the currently selected features for all layers in a data frame

Methods

panToExtent (extent)

Parameter	Explanation	Data Type
extent	A geoprocessing Extent object	Extent

This method is perfect for situations where data frame scale does not change. Rather than setting the extent and then having to reset the scale each time, **panToExtent** maintains the scale and simply centers the current data frame on the new extent.

The following example will pan the extent of a data frame by passing in the extent of selected features for a specific layer.

The Layer class **getSelectedExtent** method is used for this purpose.

```
df.panToExtent(lyr.getSelectedExtent())
```

zoomToSelectedFeatures ()

This performs the same ArcMap operation as **Selection** > **Zoom To Selected Features**. One difference is that if no features are selected, it will zoom to the full extent of all layers. If you want to zoom to the extent of selected features for a specific layer, try using the Layer.getSelectedExtent method.

The following example will zoom to the extent of all selected features:

```
df.zoomToSelectedFeatures()
```

The following example will zoom to the extent of selected features for a specific layer. This example uses a method on the Layer object.

```
df.extent = lyr.getSelectedExtent()
```

Code Sample

DataFrame example 1

The following script will set each data frame's **rotation** property to `0` and **scale** property to `1:24000` before exporting each data frame to an individual output TIFF file using the data frame **name** property as the name of the output file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    df.rotation = 0
    df.scale = 24000
    outFile = r"C:\Project\Output\\" + df.name + ".tif"
    arcpy.mapping.ExportToTIFF(mxd, outFile, df)
del mxd
```

DataFrame example 2

The following script will set the data frame's **extent** to the extent of the selected features in a layer. A `10%` buffer is added to the extent by multiplying the **scale**. This example uses the **getSelectedExtent** method on the Layer object. This script is run from the interactive window within ArcMap using the CURRENT keyword rather than providing a path to a map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
accidentLayer = arcpy.mapping.ListLayers(mxd, "accidents", df)[0]
arcpy.SelectLayerByAttribute_management(accidentLayer,
"NEW_SELECTION", "\"Accidents\" > 5")
df.extent = accidentLayer.getSelectedExtent(False)
df.scale = df.scale * 1.1
arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Output\FrequentAccidents.pdf", df)
arcpy.RefreshActiveView()
del mxd
```

DataFrame example 3

The following script will modify the position and size of a specific data frame on the layout.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
df.elementPositionX, df.elementPositionY = 1, 1
df.elementHeight, df.elementWidth = 5, 6.5
mxd.save()
del mxd
```

DataFrame example 4

The following script converts a data frame's extent object into a polygon feature so that it can be used with the **`SelectLayerByLocation_management`** function.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
lyr = arcpy.mapping.ListLayers(mxd, "Accidents", df)[0]
#The DataFrame extent object is converted into a polygon feature so it
can be used with the SelectLayerByLocation function.
dfAsFeature = arcpy.Polygon(arcpy.Array([df.extent.lowerLeft,
df.extent.lowerRight, df.extent.upperRight, df.extent.upperLeft]),
                            df.spatialReference)
arcpy.SelectLayerByLocation_management(lyr, "INTERSECT", dfAsFeature,
"", "NEW_SELECTION")
mxd.save()
del mxd
```

DataFrameTime (arcpy.mapping)

Top

Summary

The **DataFrameTime** object provides access to time management operations for time-enabled layers in a data frame.

Discussion

The **DataFrameTime** object provides capabilities for two basic scenarios:

- The first scenario is for map documents that have already been published with time-enabled layers where settings were established via the **Time Slider Options** dialog box and saved with the map document. With this scenario, those existing properties can be used for automating output. Examples 1 and 2 below are examples for this first scenario.
- The second scenario deals with map documents that don't have any time-enabled layers but that time-enabled layers are added by a script and therefore the time slider options need to be set so that the desired output can be automated. Example 3 below represents this second scenario.

The **timeStepInterval** uses the EsriTimeDelta class. The EsriTimeDelta class is an alternative to the core python **datetime.timedelta** and uses internal Esri time units for intervals that can't be handled by the core python **timedelta** object (such as months, weeks, etc.) The **timeStepInterval** can be used to loop through dates and times. If you want to use a different time interval within a loop, create a new python **timedelta** object or an EsriTimeDelta object (see example 3 below). Be aware that if the **timeWindowUnits** are modified and saved, it will affect the **timeStepInterval**.

Legacy:

Prior to the 10.1 release, the **timeStepInterval** property from the **DataFrameTime** class returned a python `datetime.timedelta` object.

To embed a time stamp within an image when exporting a data frame, use time text.

There are numerous help articles concerning time in ArcGIS. Here are a few that are most related to the methods and properties on the **DataFrameTime** object:

- Using time information from the data source
- Using the Time Slider window
- About modifying time slider properties

Properties

Property	Explanation	Data Type
currentTime (Read and Write)	The current date and time for a data frame with time-enabled layers. The same property can be found in the **Time Slider Options**dialog box in ArcMap.	DateTime
endTime (Read and Write)	The end date and time for a data frame with time-enabled layers. The same property can be found in the **Time Slider Options**dialog box in ArcMap.	DateTime
startTime (Read and Write)	The start date and time for a data frame with time-enabled layers. The same property can be found in the **Time Slider Options**dialog box in ArcMap.	DateTime
timeStepInterval (Read Only)	Returns the time step interval that has been set via the **Time Slider Options** dialog box in ArcMap. This value is a EsriTimeDeltaobject and is used to iterate over a period of time (e.g., 2 days, 1 month, and so on).	EsriTimeDelta
timeWindow (Read and Write)	The time window that controls how many units (e.g., days) of time-enabled data appear prior to and including the current time. For example, if **timeWindow** is set to 1 and **timeWindowUnits** is set to weeks, then the time window will be 2 weeks.	Double
timeWindowUnits (Read and Write)	The units that are used with the **TimeStepInterval** and the **TimeWindow** properties. The valid units are: • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS • WEEKS • MONTHS • YEARS • DECADES • CENTURIES	String

Method Overview

Method	Explanation
resetTimeExtent ()	Resets the time extent to the time window extent of all time-enabled layers in a data frame

Methods

resetTimeExtent ()

This performs the same operation as clicking the **Min Time** and **Max Time** buttons on the ***Time Slider Options*** dialog box in ArcMap.

Code Sample

DataFrameTime example 1

The following script uses time settings (start time, end time, and time interval) published in an existing map document to export a series of images. Each output image is given a unique name using the parsed date information from the time stamp.

```
import arcpy
import os

mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Traffic Analysis")[0]
df.time.currentTime = df.time.startTime

while df.time.currentTime <= df.time.endTime:
    # An example str(newTime) would be: "2008-12-29 02:19:59"
    # The following line splits the string at the space and takes the
first
    # item in the resulting string.
    fileName = str(df.time.currentTime).split(" ")[0] + ".png"
    arcpy.mapping.ExportToPNG(mxd, os.path.join(r"C:\Project\Output",
fileName), df)
    df.time.currentTime = df.time.currentTime +
df.time.timeStepInterval
del mxd
```

DataFrameTime example 2

The following script is identical to example 1 above but uses a modified start time, end time, and interval that are different from the existing settings that were published in a time-enabled data frame within a map document. The Python datetime module is used to create times and time deltas. Alternatively, the EsriTimeDelta class could also be used to create time deltas.

```
import arcpy
import datetime
import os

mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Traffic Analysis")[0]
df.time.currentTime = datetime.datetime(2008, 10, 1)
endTime = datetime.datetime(2008, 10, 31)
interval = datetime.timedelta(days=7)

while df.time.currentTime <= endTime:
    # An example str(newTime) would be: "2008-01-29 02:19:59"
    # The following line splits the string at the space and takes the
first
    # item in the resulting string.
    fileName = str(df.time.currentTime).split(" ")[0] + ".png"
    arcpy.mapping.ExportToPNG(mxd, os.path.join(r"C:\Project\Output",
fileName), df)
    df.time.currentTime = df.time.currentTime + interval
del mxd
```

DataFrameTime example 3

The following script represents a scenario in which a new time-enabled layer file is added to a new data frame that is not time aware; therefore, the time slider properties are not set within the map document. This script adds a time-enabled layer to the data frame using the AddLayer function and then sets the appropriate time settings similar to the scripts above. In this example, the time interval is set using the EsriTimeDelta class. Alternatively, the Python datetime module could be used to create the time delta.

```
import arcpy
import datetime
import os

mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
timeLayer = arcpy.mapping.Layer(r"C:\Project\Data\Accidents.lyr")
arcpy.mapping.AddLayer(df, timeLayer, "AUTO_ARRANGE")
df.time.resetTimeExtent()
df.time.timeWindowUnits = "DAYS"
df.time.timeWindow = 7
df.time.currentTime = datetime.datetime(2008, 10, 1)
endTime = datetime.datetime(2008, 10, 31)
interval = arcpy.time.EsriTimeDelta(1, 'weeks')

while df.time.currentTime <= endTime:
    # An example str(newTime) would be: "2008-01-29 02:19:59"
    # The following line splits the string at the space and takes the
first
    # item in the resulting string.
    fileName = str(df.time.currentTime).split(" ")[0] + ".png"
    arcpy.mapping.ExportToPNG(mxd, os.path.join(r"C:\Project\Output",
fileName))
    df.time.currentTime = df.time.currentTime + interval
del mxd, timeLayer
```

GraduatedColorsSymbology (arcpy.mapping)

Top

Summary

The **GraduatedColorsSymbology** class provides access to different properties that allow you to change the appearance of a layer's graduated colors symbology.

Discussion

The **GraduatedColorsSymbology** class provides access to a limited number of properties and methods that allow you to automate layer symbology in a map document (`.mxd`) or layer file (`.lyr`). Basic operations such as changing the number of classes, modifying class break values and labels, or changing the field that the symbology is based on are some of the properties that can be modified. For access to the complete set of layer symbology properties and settings, for example, changing individual symbols for individual classes, it is necessary to author these changes in the ArcMap user interface and save those changes to a layer file. These custom changes can then be applied to existing layers using the UpdateLayer function.

A layer can use any number of symbologies but not all of them can be modified. Not all layers will use the **GraduatedColorsSymbology** class, so it is important to first test if the layer is using this symbology class before attempting to make changes to its properties. The **symbologyType** property on the Layer class was designed for this purpose. First test if the **symbologyType** for the layer is graduated colors (`if lyr.symbologyType == "GRADUATED_COLORS":`), then create a variable reference to the **GraduatedColorsSymbology** class for that layer (`lyrSymbolClass = lyr.symbology`).
The **symbologyType** on the Layer object is a read-only property. In other words, you can't change a graduated colors symbology to a graduated symbols or a unique value symbology. You can only change the properties for a specific symbology class on a layer. The only way to change the symbology type is by publishing the desired result to a layer file and using the UpdateLayer function.
The classification method cannot be changed. In cases where you want to use different classification methods, you would need to preauthor layer files and use those to update a layer, then modify the properties that can be changed. The only exception to this rule is when you set **classBreakValues**. Similar to the ArcMap user interface, explicitly setting **classBreakValues** will automatically set the classification method to `manual`. Also similar to the ArcMap user interface, once the classification method is set to `manual`, you can't change the **numClasses** parameter.

Unlike the ArcMap user interface, you can set a minimum value when setting the **classBreakValues** parameter. The first value in the **classBreakValues** list is the minimum value. All other values are the class break values as they appear in the ArcMap user interface. For this reason, the **classBreakValues** list will always have one more value than the **classBreakLabels** and **classBreakDescriptions** lists.

The **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions** lists must always be sorted from lowest value to highest value. This is also the case for layers that were authored with reversed sorting.

Setting one parameter will often modify other parameters automatically. For example, if you set **numClasses**, **normalization**, or the **valueField** parameters, the **classBreakValues**, **classBreakLabels**, and**classBreakDescriptions** properties will automatically be adjusted based on the current classification method. For this reason, the order in which properties are modified is important.

The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). For example, let's say you want to update the color properties of the symbology of a layer in a map document with the symbology stored in a layer file. However, the layer in the map document and the layer file have different data sources. The minimum and maximum values and class breaks in the layer file may be different than the layer in the map document. Updating the symbology of the layer in the map document with the symbology stored in the layer file may produce unintended results (for example, the class break values will match the layer file's data source statistics as opposed to the map document layer's data source statistics. However, if you follow UpdateLayer with the **reclassify()** method, the end result is like using the color properties from the symbology in the layer file, but other characteristics are based on the map document layer's underlying source data.

If you are making these symbology modifications via the Python window and you are referencing a map document using `CURRENT`, you may not immediately see the changes in the ArcMap application. To refresh the map document, try using the RefreshActiveView and RefreshTOC functions.

Properties

Property	Explanation	Data Type
classBreakDescriptions (Read and Write)	A sorted list of strings that represent the descriptions for each class break value that can optionally appear in a map document's legend. These values are only accessible in the ArcMap user interface by right-clicking a symbol displayed within the **Symbology**tab in the **Layer Properties** dialog box and selecting **Edit Description**. The number of descriptions in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties, so it is best practice to set these values last.	List
classBreakLabels (Read and Write)	A sorted list of strings that represent the labels for each class break that appear in the map document's table of contents and legend items. The number of labels in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties so it is best practice to set these values last.	List
classBreakValues (Read and Write)	A sorted list of doubles that includes the minimum and maximum values that represent the class breaks. When setting**classBreakValues**, it will automatically set the **numClasses** property and will also set the classification method to `manual` as well as update other properties like **classBreakLabels**. Unlike the ArcMap user interface, you have the ability to set a minimum value. The first value in the sorted list represents the minimum value and the remaining values are the class breaks that appear in the user interface; therefore, there will always be one more item in the **classBreakValues** list than in the**classBreakLabels** and **classBreakDescriptions** lists. Changing this value will automatically adjust other symbology properties based on the new information.	List
normalization (Read and Write)	A string that represents a valid dataset field name used for normalization. Changing this value will automatically adjust other symbology properties based on the new information. The normalization field can be removed by setting the value to `None` (for example, `lyr.symbology.normalization = None`).	String
numClasses (Read and Write)	A long integer that represents the number of classes to be used with the current classification method. Changing this value will overwrite other symbol properties like **classBreakValues** and **classBreakLabels**. This value cannot be set if the classification method is `manual`; therefore, **numClasses** should not be called after the **classBreakValues** property because it will automatically set the classification method to `manual`. Changing this value will automatically adjust other symbology properties based on the new information.	
valueField (Read and Write)	A string that represents a valid dataset field name used for the layer's classification symbology. Changing this value will automatically adjust other symbology properties based on the new information.	String

Method Overview

Method	Explanation
reclassify ()	Resets the layer's symbology to the layer's data source information and statistics.

Methods

reclassify ()

The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). This method will automatically update the symbology properties based on the layer's actual data source information and statistics and not the information that is persisted in a layer file. The method needs to be used cautiously because it has the potential to overwrite other symbology properties.
The **reclassify** method will regenerate **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions**. It will not affect**numClasses** or **normalization**. The **reclassify** method has no affect on a `manual` classification.

Code Sample

GraduatedColorsSymbology example 1

The following script modifies the symbology for all graduated colors layers in a map document. It iterates through each layer, changes the value field and the number of classes, and saves the map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for lyr in arcpy.mapping.ListLayers(mxd):
  if lyr.symbologyType == "GRADUATED_COLORS":
    lyr.symbology.valueField = "POP2007"
    lyr.symbology.numClasses = 5
mxd.save()
del mxd
```

GraduatedColorsSymbology example 2

The following script modifies the symbology for a layer in a map document. It first updates the layer's symbology using a layer file on disk with the UpdateLayer function. The layer file contains a custom color ramp which is applied to the layer. Next, it verifies that the layer has graduated color symbology. Finally, the script modifies a number of the properties on the **GraduatedColorsSymbology** class and exports the result to PDF.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Census")[0]
lyr = arcpy.mapping.ListLayers(mxd, "StatePopulation", df)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\LYRs\Population.lyr")
arcpy.mapping.UpdateLayer(df, lyr, lyrFile, True)
if lyr.symbologyType == "GRADUATED_COLORS":
  lyr.symbology.valueField = "POP2000"
  lyr.symbology.classBreakValues = [250000, 999999, 4999999, 9999999,
35000000]
  lyr.symbology.classBreakLabels = ["250,000 to 999,999", "1,000,000
to 4,999,999",
                                    "5,000,000 to 9,999,999",
"10,000,000 to 35,000,000"]
arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Output\StatePopulation.pdf")
del mxd, lyrFile
```

GraduatedSymbolsSymbology (arcpy.mapping)

Top

Summary

The **GraduatedSymbolsSymbology** class provides access to different properties that allow you to change the appearance of a layer's graduated symbols symbology.

Discussion

The **GraduatedSymbolsSymbology** class provides access to a limited number of properties and methods that allow you to automate layer symbology in a map document (`.mxd`) or layer file (`.lyr`). Basic operations such as changing the number of classes, modifying class break values and labels, or changing the field that the symbology is based on are some of the properties that can be modified. For access to the complete set of layer symbology properties and settings, for example, changing individual symbols for individual classes, it is necessary to author these changes in the ArcMap user interface and then save those changes to a layer file. These custom settings can then be applied to existing layers using the UpdateLayer function.

A layer can use any number of symbologies, but not all of them can be modified. Not all layers will use the **GraduatedSymbolsSymbology** class, so it is important to first test if the layer is using this symbology class before attempting to make changes to its properties. The **symbologyType** property on the Layer class was designed for this purpose. First test if the **symbologyType** for the layer is graduated symbols (`if lyr.symbologyType == "GRADUATED_SYMBOLS":`), then create a variable reference to the **GraduatedSymbolsSymbology** class for that layer (`lyrSymbolClass = lyr.symbology`).

The **symbologyType** on the Layer object is a read-only property. In other words, you can't change a graduated symbols symbology to a graduated colors or a unique value symbology. You can only change the properties for a specific symbology class on a layer. The only way to change the symbology type is by publishing the desired result to a layer file and using the UpdateLayer function. The classification method cannot be changed. In cases where you want to use different classification methods, you would need to preauthor layer files and use those to update a layer, then modify the properties than can be changed. The only exception to this rule is when you set **classBreakValues**. Similar to the ArcMap user interface, explicitly setting **classBreakValues** will automatically set the classification method to `manual`. Also similar to the ArcMap user interface, once the classification method is set to `manual`, you can't change the **numClasses** parameter.

Unlike the ArcMap user interface, you can set a minimum value when setting the **classBreakValues** parameter. The first value in the **classBreakValues** list is the minimum value. All other values are the class break values as they appear in the ArcMap user interface. For this reason, the **classBreakValues** list will always have one more value than the **classBreakLabels** and **classBreakDescriptions** lists. The **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions** lists must always be sorted from lowest value to highest value. This is also the case for layers that were authored with reversed sorting.

Setting one parameter will often modify other parameters automatically. For example, if you set **numClasses**, **normalization**, or the **valueField** parameters, the **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions** properties will automatically be adjusted based on the current classification method. For this reason, the order in which properties are modified is important. The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). For example, let's say you want to update the color properties of the symbology of a layer in a map document with the symbology stored in a layer file. However, the layer in the map document and the layer file have different data sources. The minimum and maximum values and class breaks in the layer file may be different than the layer in the map document. Updating the symbology of the layer in the map document with the symbology stored in the layer file may produce unintended results (for example, the class break values will match the layer file's data source statistics as opposed to the map document layer's data source statistics. However, if you follow UpdateLayer with the **reclassify()** method, the end result is like using the color properties from the symbology in the layer file, but other characteristics are based on the map document layer's underlying source data.

If you are making these symbology modifications via the Python window and you are referencing a map document using `CURRENT`, you may not immediately see the changes in the ArcMap application. To refresh the map document, try using the RefreshActiveView and RefreshTOC functions.

Properties

Property	Explanation	Data Type
classBreakDescriptions (Read and Write)	A sorted list of strings that represent the descriptions for each class break value that can optionally appear in a map document's legend. These values are only accessible in the ArcMap user interface by right-clicking a symbol displayed within the **Symbology**tab in the **Layer Properties** dialog box and selecting **Edit Description**. The number of descriptions in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties, so it is best practice to set these values last.	List
classBreakLabels (Read and Write)	A sorted list of strings that represent the labels for each class break that appear in the map document's table of contents and legend items. The number of labels in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties so it is best practice to set these values last.	List
classBreakValues (Read and Write)	A sorted list of doubles that includes the minimum and maximum values that represent the class breaks. When setting**classBreakValues**, it will automatically set the **numClasses** property and will also set the classification method to `manual` as well as update other properties like **classBreakLabels**. Unlike the ArcMap user interface, you have the ability to set a minimum value. The first value in the sorted list represents the minimum value and the remaining values are the class breaks that appear in the user interface; therefore, there will always be one more item in the **classBreakValues** list than in the**classBreakLabels** and **classBreakDescriptions** lists. Changing this value will automatically adjust other symbology properties based on the new information.	List
normalization (Read and Write)	A string that represents a valid dataset field name used for normalization. Changing this value will automatically adjust other symbology properties based on the new information. The normalization field can be removed by setting the value to `None` (for example, `lyr.symbology.normalization = None`).	String
numClasses (Read and Write)	A long integer that represents the number of classes to be used with the current classification method. Changing this value will overwrite other symbol properties like **classBreakValues** and **classBreakLabels**. This value cannot be set if the classification method is `manual`; therefore, **numClasses** should not be called after the **classBreakValues** property because it will automatically set the classification method to `manual`.	Long
	Changing this value will automatically adjust other symbology properties based on the new information.	
valueField (Read and Write)	A string that represents a valid dataset field name used for the layer's classification symbology. Changing this value will automatically adjust other symbology properties based on the new information.	String

Method Overview

Method	Explanation
reclassify ()	Resets the layer's symbology to the layer's data source information and statistics.

Methods

reclassify ()

The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). This method will automatically update the symbology properties based on the layer's actual data source information and statistics and not the information that is persisted in a layer file. The method needs to be used cautiously because it has the potential to overwrite other symbology properties.
The **reclassify** method will regenerate **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions**. It will not affect**numClasses** or **normalization**. The **reclassify** method has no affect on a `manual` classification.

Code Sample

GraduatedSymbolsSymbology example 1

The following script modifies the symbology for all layers in a map document. It iterates through each layer, changes the value field, changes the number of classes, and saves the map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for lyr in arcpy.mapping.ListLayers(mxd):
  if lyr.symbologyType == "GRADUATED_SYMBOLS":
    lyr.symbology.valueField = "POP2007"
    lyr.symbology.numClasses = 5
del mxd
```

GraduatedSymbolsSymbology example 2

The following script modifies the symbology for a layer in a map document. It first updates the layer's symbology using a layer file on disk with the UpdateLayer function. Next, it verifies that the layer has graduated symbols symbology. Finally, the script modifies a number of the properties on the **GraduatedSymbolsSymbology** class and exports the result to PDF.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Census")[0]
lyr = arcpy.mapping.ListLayers(mxd, "StatePopulation", df)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\LYRs\Population.lyr")
arcpy.mapping.UpdateLayer(df, lyr, lyrFile, True)
if lyr.symbologyType == "GRADUATED_SYMBOLS":
  lyr.symbology.valueField = "POP2000"
  lyr.symbology.classBreakValues = [250000, 999999, 4999999, 9999999,
35000000]
  lyr.symbology.classBreakLabels = ["250,000 to 999,999", "1,000,000
to 4,999,999",
                                    "5,000,000 to 9,999,999",
"10,000,000 to 35,000,000"]
arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Output\StatePopulation.pdf")
del mxd, lyrFile
```

GraphicElement (arcpy.mapping)

Top

Summary

The **GraphicElement** object provides access to properties that enables its repositioning on the page layout as well as methods that allow for duplicating and deleting existing graphic elements.

Discussion

The **GraphicElement** is a catch-all element type for most generic elements added to a page layout. It includes items such as groups of elements, inserted tables, graphs, neatlines, markers, lines, area shapes, and so on, that are added to a page layout. The most common operations on a graphic element are to get or set its page layout position and size. The ListLayoutElements function returns a Python list of page layout element objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific page element object. To return a list of only **GraphicElements**, use the GRAPHIC_ELEMENT constant for the**element_type** parameter. A wildcard can also be used to further refine the search based on the element name.

Existing graphic elements can be cloned and deleted. This capability was initially added to support the creation of dynamic graphic tables on a page layout where each cell in the table can be outlined using line graphics. To accomplish this, a map document must be authored with at least two graphic line elements: a vertical line and a horizontal line. As the information is read from a table, the lines can be cloned using the **clone** method and sized and positioned appropriately using other graphic element properties. When cloning an element it is very useful to provide a **suffix** value so that cloned elements can be easily identified while using the ListLayoutElements function with a **wildcard** and the same **suffix** value. The returned list of elements can be further modified or deleted using the **delete** method. There is a complete code sample for building a dynamic graphic table at the bottom of this topic.

Grouped graphic elements cannot be cloned. That is because grouped elements may include more than just graphic elements; they could also include items like north arrows, scale bars and so on. Use the **isGroup property to determine if a graphic element is a group element before tyring to clone it.**

ListLayoutElements will return a flattened list of elements. For example, **ListLayoutElements** on a graphic element that represents a group of three text elements will return a total of four elements: the group element and each individual text element. The **GraphicElement** can be used to reposition all items at the same time or the text element text values can be managed individually.

It is recommended that each page layout element be given a unique name so that it can be easily isolated using arcpy scripting. This is set via the **Size and Position** tab on the ***Properties*** dialog box in ArcMap.

X and Y element positions are based on the element's anchor position which is also set via the **Size and Position** tab on the ***Properties*** dialog box in ArcMap.

Page units can only be changed in ArcMap via **Customize** > **ArcMap Options** > **Layout View Tab**.

Properties

Property	Explanation	Data Type
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The x location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
isGroup (Read Only)	Returns `True` if the layout element is a group element. It is not possible to **clone** or **delete** group elements.	Boolean
name (Read and Write)	The name of the element.	String
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Data frame element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Map surround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Method Overview

Method	Explanation
clone ({suffix})	Provides a mechanism to clone an existing graphic element on a page layout.
delete ()	Provides a mechanism to delete an existing graphic element on a page layout.

Methods

clone ({suffix})

Parameter	Explanation	Data Type
suffix	An optional string that is used to tag each newly created graphic element. The new element will get the same element name as the parent graphic plus the suffix value plus a numeric sequencer. For example, if the parent element name is `Line` and the **`suffix`** value is `_copy`, the newly cloned elements would be named `Line_copy`, `Line_copy_1`, `Line_copy_2`, and so on. If a **`suffix`** is not provided, then the results would look like `Line_1`, `Line_2`, `Line_3`, and so on.	String

Grouped graphic elements can't be cloned. First check to see if a graphic element is grouped by using the **`isGroup`** property.

delete ()

It may be necessary to delete cloned elements. Cloned elements, when created, can be given a custom **`suffix`** so they can be easy to find when using the wildcard parameter on the ListLayoutElements function.

Code Sample

GraphicElement example 1

The following script will move a group element named `Title Block` to a new location on the page layout, then save the changes.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "GRAPHIC_ELEMENT"):
    if elm.name == "Title Block":
        elm.elementPositionX = 4.75
        elm.elementPositionY = 10.5
mxd.save()
del mxd
```

GraphicElement example 2

The following script will construct a graphic table based on data values from a table in the map document. The map document was authored with a vertical line named `vertLine`, a horizontal line named `horzLine`, and a text element named `TableText`. Each of the elements were authored with the appropriate symbology properties. The element's anchors were also set to the upper left position and the text element's vertical and horizontal justification were set to top left.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")

#Reference items in the map document
lyr = arcpy.mapping.ListLayers(mxd, "Accidents")[0]
horzLine = arcpy.mapping.ListLayoutElements(mxd, "GRAPHIC_ELEMENT",
"horzLine")[0]
vertLine = arcpy.mapping.ListLayoutElements(mxd, "GRAPHIC_ELEMENT",
"vertLine")[0]
tableText = arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT",
"TableText")[0]

#Get/set information about the table
numRows = int(arcpy.GetCount_management(lyr).getOutput(0))
rowHeight = 0.2
fieldNames = ["X", "Y", "Accidents"]
numColumns = len(fieldNames)
colWidth = 1.5

#Build graphic table lines based on upper left coordinate
#  set the proper size of the original, parent line, then clone it and
position appropriately
upperX = 1.0
upperY = 5.0

#Vertical lines
vertLine.elementPositionX = upperX
vertLine.elementPositionY = upperY
vertLine.elementHeight =  (rowHeight * numRows) + rowHeight #extra
line for column names
```

```
x = upperX
for vert in range(1, numColumns+1):
  x = x + colWidth
  vert_clone = vertLine.clone("_clone")
  vert_clone.elementPositionX = x

#Horizontal lines
horzLine.elementPositionX = upperX
horzLine.elementPositionY = upperY
horzLine.elementWidth = numColumns * colWidth

y = upperY - rowHeight
for horz in range(1, numRows +2 ):  #need to accommodate the extra
line for field names
  temp_horz = horzLine.clone("_clone")
  temp_horz.elementPositionY = y
  y = y - rowHeight

#Place text column names
tableText.elementPositionX = upperX + 0.05 #slight offset
tableText.elementPositionY = upperY
tableText.text = fieldNames[0]
accumWidth = colWidth
for field in range(1, numColumns):
  newFieldTxt = tableText.clone("_clone")
  newFieldTxt.text = fieldNames[field]
  newFieldTxt.elementPositionX = newFieldTxt.elementPositionX +
accumWidth
  accumWidth = accumWidth + colWidth

#Create text elements based on values from the table
table = arcpy.SearchCursor(lyr.dataSource)
y = upperY - rowHeight
for row in table:
  x = upperX + 0.05 #slight offset
  try:
    for field in fieldNames:
      newCellTxt = tableText.clone("_clone")
      newCellTxt.text = row.getValue(field)
      newCellTxt.elementPositionX = x
      accumWidth = accumWidth + colWidth
      x = x + colWidth
    y = y - rowHeight
  except:
    print"Invalid value assignment"

#Export to PDF and delete cloned elements
arcpy.mapping.ExportToPDF(mxd, r"C:\Temp\test.pdf")

for elm in arcpy.mapping.ListLayoutElements(mxd, wildcard="_clone"):
  elm.delete()
del mxd
```

LabelClass (arcpy.mapping)

Top

Summary

Provides access to a layer's label class properties

Discussion

The **`LabelClass`** object is essential for managing properties, such as label expressions or SQL queries, that are associated with a layer's individual label classes.

Access to these properties is essential when, for example, a map document's layers are redirected to a new workspace. The label classes' SQL query may need to be updated with the appropriate syntax for the new database it is being executed against. For example, field names in a personal geodatabase are surrounded by square brackets while field names in the file geodatabase are surrounded with double quotes. In addition to field names, other SQL properties (wildcard characters, other special characters, operators, and so on) may need to be changed as well. If the SQL query is not updated, layers may fail to draw. For more information on these special cases, refer to the Updating and fixing data sources within arcpy.mapping topic.

The label expression is either using the VBScript, JScript or Python parsers. The syntax and/or special characters for the parsers should not change (for example, VBScript always uses square brackets), but realize that the field names may change. `Area` and `perimeter` are commonly used fields in an expression, and these do vary in name from data source to data source.

Not all layers support the **`labelClasses`** property, so it is useful to test this ahead of time using **`the supports`** method—for example, `layer.supports("SHOWLABELS")` or `layer.supports("LABELCLASSES")`. If a layer supports labels, it also will support the labelClasses property, so you don't need to test for both.

The **`labelClasses`** property will return a list of LabelClass objects. To reference a specific LabelClass object, it will be necessary to loop through each item in the list or provide a specific index number.

Properties

Property	Explanation	Data Type
className (Read and Write)	Provides the ability to get or set a layer's individual label class name.	String
expression (Read and Write)	Provides the ability to get or set a layer's individual label class expression. This can be as simple as a single field or more advanced using either a VBScript, JScript or Python expression.	String
SQLQuery (Read and Write)	Provides the ability to get or set a layer's individual label class SQLQuery. This is useful for restricting labels to certain features.	String
showClassLabels (Read and Write)	Provides the ability get or set the visibility of individual label classes.	Boolean

Code Sample

LabelClass example 1

The following script will print the label class properties for only those layers that have labels and individual label classes turned on. The script first confirms that the layer supports the **`labelClasses`** property.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for lyr in arcpy.mapping.ListLayers(mxd):
    if lyr.supports("LABELCLASSES"):
        if lyr.showLabels:
            print "Layer name: " + lyr.name
            for lblClass in lyr.labelClasses:
                if lblClass.showClassLabels:
                    print "    Class Name:  " + lblClass.className
                    print "    Expression:  " + lblClass.expression
                    print "    SQL Query:   " + lblClass.SQLQuery
del mxd
```

LabelClass example 2

The following script will modify the SQL query that once pointed to a personal geodatabase, but now its layer references a file geodatabase. Square brackets will be changed to double quotes and the wildcard character will be modified from an asterisk (*) to a percent symbol (%). A new output map document is saved, preserving the original map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for lyr in arcpy.mapping.ListLayers(mxd):
    if lyr.supports("LABELCLASSES"):
        for lblClass in lyr.labelClasses:
            lblClass.SQLQuery = lblClass.SQLQuery.replace("[", "\"")
            lblClass.SQLQuery = lblClass.SQLQuery.replace("]", "\"")
            lblClass.SQLQuery = lblClass.SQLQuery.replace("*", "%")
mxd.saveACopy("c:/project/project2.mxd")
del mxd
```

Layer (arcpy.mapping)

Top

Summary

Provides access to layer properties and methods. It can either reference layers in a map document (.mxd) or layers in a layer (.lyr) file.

Discussion

The **Layer** object is essential for managing layers that reside within a map document (.mxd) or within a layer (.lyr) file. The **layer** object provides access to many of the common layer properties found in the ArcMap **Layer Properties** dialog box and it also provides methods for saving layer files.

The Layer function, the ListLayers function and the **listLegendItemLayers** method on the Legend object all provide ways to reference a **Layer** object.

There are numerous types of layers and not all of them support the same set of properties. For example, a feature layer supports a definition query whereas a raster layer does not, but a raster catalog does. Rather than having to work with different, individual layer objects for all possible layer types and property combinations, a **supports** method is available to help identify which layer types support which individual layer properties. The **supports** method gives you the ability to test if the layer supports a property before trying to get or set its value on a layer type that doesn't support it, therefore reducing the need for additional error trapping.

There are essentially three categories of layers in a map document: feature layers, group layers, and raster layers. The **isFeatureLayer**, **isGroupLayer**, and **isRasterLayer** properties allow you to identify or isolate the majority of layer types but not all layer types. There are a few specialized layers and datasets that don't fall into one of these three categories: annotation subclasses, dimension features, network datasets, terrain datasets, topology datasets, and so on. In these cases you may need to test other properties to isolate a layer of interest before doing something to it.

Not all layer properties are accessible through the **Layer** object. There are many properties available in the ArcMap **Layer Properties** dialog box that are not exposed to the arcpy scripting environment (for example, display properties, field aliases, selection symbology, and so on). The UpdateLayer function allows you to replace all layer properties available in the ArcMap **Layer Properties** dialog box using a layer (.lyr) file that contains the customizations.

Group layers and other sublayers (for example, annotation classes) are treated just like ordinary layers. The **ListLayers** function returns index values that are generated from top to bottom as they appear in the table of contents or the way they appear in a layer (.lyr) file. The same applies if a group layer is within another group layer. For example, a map document with a single group layer that contains three sublayers will return a list of four layer names, the group layer being the first and the three sublayers being the second, third, and fourth. There are two ways of determining if a layer is a group layer. First, you can check to see if the layer supports the **isGroupLayer**property. Second, you can evaluate the **longName**property. A layer's **longName** value will include the group name in addition to the layer name. For example, a layer named Layer1 in a group layer named Group1 will have a**longName**value of Group1\Layer1. If the **name** value is equal to **longName** value, then the layer is not a group layer or the layer is not inside a group layer.

Some layers within a map document or layer file may be password protected because the user and password information is not saved within the layer file or map document. Map documents that contain these layers will prompt the user to enter the appropriate information while the document is opening. The arcpy.mapping scripting environment will, by default, suppress these dialog boxes during execution, but that means that the layers will be treated as though they have broken data sources. In other words, secured layers will not be rendered in any output. If it is necessary for these layers to render appropriately, then there are a couple of options. First, save the user name and password information with the layers. Second, the CreateArcSDEConnectionFile geoprocessing function allows you to create a connection file that is also persisted in memory. If this function is executed prior to opening a map document (.mxd) with the MapDocument function or a layer file with the Layer function, then SDE layers will render. Currently, there is no alternative for secured web services.

The variable that references a layer file on disk will place a lock on the (.lyr) file. It is good practice to remove the object reference using the Python del command at the end of a script or within a Python try/exceptstatement.

Changing a layer's data source is a common requirement. There are two methods on the **Layer** object that help with this.

The **findAndReplaceWorkspacePath** method is intended for replacing part or all of a layer's workspace path. The **replaceDataSource** method allows you to change a layer's workspace and source dataset. For a more detailed discussion, parameter information, scenarios, and code samples, please refer to theUpdating and fixing data sources with arcpy.mapping help topic.

Depending on the symbology type, a layer's symbology can be modified. There are a limited number of supported symbology types for which properties and methods are available. It is good practice to first test the layer's**symbologyType** property. If a value of OTHER is returned, then the layer's symbology can't be modified. If the value returned is not OTHER, then the layer's **symbology** property will return one of the following symbology classes, each with their own unique set of methods and properties: GraduatedColorsSymbology, GraduatedSymbolsSymbology, RasterClassifiedSymbology, and UniqueValuesSymbology.

Time-management operations can be performed for time-enabled layers. Not all layer types support time properties. Therefore, it is good practice to first test if the layer supports time using the **supports** method. If the layer does support time, then time properties can be accessed from the LayerTime class.

Syntax

Layer (lyr_file_path)

Parameter	Explanation	Data Type
lyr_file_path	A string that includes the full path and file name of an existing layer (`.lyr`) file.	String

Properties

Property	Explanation	Data Type
brightness (Read and Write)	Provides the ability to get or set the brightness value. The default, normal brightness, is 0%. Enter any value between +100% and -100%. Enter a plus or minus sign to the left of the value to specify whether it is above or below 0. Not all layers support the`brightness` property (for example, group layers and feature layers), so it is good practice to test for this ahead of time using the`supports` method.	Integer
contrast (Read and Write)	Provides the ability to get or set the contrast value. The default, neutral contrast, is 0%. Enter any value between +100% and -100%. Enter a plus or minus sign to the left of the value to specify whether it is above or below 0. Not all layers support the`contrast` property (for example, annotation layers and fabric layers), so it is good practice to test for this ahead of time using the `supports` method.	Integer
credits (Read and Write)	Provides the ability to either get or set the layer's credits or copyright information.	String
datasetName (Read Only)	Returns the name of the layer's dataset the way it appears in the workspace, not in the TOC. Not all layers support the`datasetName` property (for example, web services), so it is good practice to test for this ahead of time using the `supports`method.	String
dataSource (Read Only)	Returns the complete path for the layer's data source. It includes the `workspacePath` and the `datasetName` properties combined. Not all layers support the `dataSource` property (for example, annotation classes and web services), so it is good practice to test for this ahead of time using the `supports` method.	String
definitionQuery (Read and Write)	Provides the ability to get or set a layer's definition query. Not all layers support the `definitionQuery` property (for example, raster layers and group layers), so it is good practice to test for this ahead of time using the `supports` method.	String
description (Read and Write)	Provides the ability to either get or set the layer's description information. Not all layers support the `description` property (for example, topology layers), so it is good practice to test for this ahead of time using the `supports` method.	String
isBroken (Read Only)	Returns `True` if a layer's data source is broken.	Boolean
isFeatureLayer (Read Only)	Returns `True` if a layer is a feature layer.	Boolean
isGroupLayer (Read Only)	Returns `True` if a layer is a group layer.	Boolean
isNetworkAnalystLayer (Read Only)	Returns `True` if a layer is an ArcGIS Network Analyst extension layer type.	Boolean
isRasterizingLayer (Read Only)	Returns `True` if a layer will cause rasterization of other vector layers in the data frame when the map is printed or exported. Rasterization of vector layers during output most often occurs when layer transparency is used but can also happen when a layer has raster-based picture symbols or field-based transparency.	Boolean
isRasterLayer (Read Only)	Returns `True` if a layer is a raster layer.	Boolean
isServiceLayer (Read Only)	Returns `True` if a layer is a GIS service layer. GIS services are automated geographic information services that are published and accessed over the web using standard technologies and protocols.	Boolean
labelClasses (Read and Write)	Provides access to a layer's label class properties by returning a list of LabelClass objects. Individual `LabelClass` object properties can be read and modified and written back to the layer. Not all layers support the `labelClasses` property (for example, raster layers and annotation layers), so it is good practice to test for this ahead of time using the `supports` method.	LabelClass
longName (Read Only)	This property is valuable when trying to determine whether a layer belongs to a group layer. If a layer does not belong to a group layer, the long name will equal the layer name. If a layer does belong to a group layer, the group layer structure will be included in the long name. For example,	String

	the name of a layer nested inside a group layer within another group layer may look something like `Group1\Group2\LayerName`. All layer types support this property.	
maxScale (Read and Write)	Provides the ability to set or get the layer's maximum scale threshold.	Double
minScale (Read and Write)	Provides the ability to set or get the layer's minimum scale threshold.	Double
name (Read and Write)	Provides the ability to set or get the name of a layer the way it would appear in the ArcMap table of contents. Spaces can be included. All layer types support this property.	String
serviceProperties (Read Only)	Provides access to connection information for ArcSDE and web service layers. The returned results are dictionary key-value pairs. There are two different dictionaries returned based on the type of layer. The first is for ArcSDE connections, and the second is for all web service layer types. The web services dictionary contains keys that work for all service layer types and also includes specific keys that work for only a particular web service type (for example, WMS has a key called WMSTitle). Either your script can check the `ServiceType` key before evaluating specific keys or you can use the `get` method that allows you to bypass keys that are not available. Not all layers support the `serviceProperties` property (for example, layers that are not ArcSDE or web service layers), so it is good practice to test for this ahead of time using the `supports` method. **Keys for an ArcSDE dictionary** • ServiceType —The property displaying the type of service. This will only be SDE for ArcSDE layer types. • Server —The name or IP address of the computer where the ArcSDE geodatabase is located. • Service —The name or port number of the process running on the ArcSDE server. • Database —The name of the enterprise RDBMS database. This is not required when using Oracle. • UserName —A user account. This will be blank if using operating system authentication. • AuthenticationMode —Geodatabase or operating system authentication. • Version —The version of the geodatabase to which you are connecting. **Keys for a web service dictionary** • ServiceType —Property displaying the type of service. These include ImageServer, IMS, MapServer, TiledInternetLayer, WMS, and WCS. • URL —Property displaying the URL to the service. If the	Dictionary

	connection to ArcGIS for Server is through a local area network (LAN), this value will be null. • Server —Property displaying the server name. If the connection to ArcGIS for Server is through the Internet (HTTP), this value will be null. • UserName —Property displaying the user name used to access a secured service. If the service is not password protected, this property will be null. • ServiceName —IMS service layers only. Property displays the name of the IMS service. • WMSName —WMS service layers only. Property displays the text string for the WMS service used for machine-to-machine communication. • WMSTitle —WMS service layers only. Property displays the description title string for the WMS service. • Name —WMS service layers only. Property displays the text string for the WMS layer used for machine-to-machine communication. • Title —WMS service layers only. Property displays the description title string for the WMS layer.	
showLabels (Read and Write)	Controls the display of labels for a layer. If set to `True`, labels will display; if set to `False`, labels will not be drawn. Not all layers support the `showLabels` property (for example, raster layers and annotation layers), so it is good practice to test for this ahead of time using the `supports` method. Layer types that support the `showLabels` property also support the `labelClasses`property.	Boolean
symbology (Read Only)	Returns a reference to the layer's symbology class. Each supported layer symbology class has its own unique set of properties. It is best to first determine the layer's `symbologyType` before attempting to modify the symbology class properties.	Object
symbologyType (Read Only)	Returns a string that represents the layer's symbology class type. Not all layer symbology class types are supported; for those that are not, the keyword `OTHER` is returned. The following is a list of possible values: • GRADUATED_COLORS — The GraduatedColorsSymbology class. • GRADUATED_SYMBOLS — The GraduatedSymbolsSymbology class. • OTHER —A string that represents an unsupported layer symbology class. • UNIQUE_VALUES —The UniqueValuesSymbology class. • RASTER_CLASSIFIED — The RasterClassifiedSymbology class for raster layers.	Object
time (Read Only)	Returns the LayerTime class that provides access to time properties of time-enabled layers.	Object

transparency (Read and Write)	Provides the ability to get or set the transparency value. This enables you to see through a layer to the layers underneath. Type `0`if you don't want a layer to be transparent. A transparency value of more than 90 percent usually results in the layer not being drawn at all. Not all layers support the `transparency` property (for example, fabric group layers and web service sublayers), so it is good practice to test for this ahead of time using the `supports` method.	Integer
visible (Read and Write)	Controls the display of a layer. This has the same effect as checking the check box next to the layer in the table of contents in ArcMap. If set to `True`, the layer will draw; if set to `False`, the layer will not be drawn. Not all layers support the `visible` property (for example, restricted web service layers), so it is good practice to test for this ahead of time using the `supports` method.	Boolean
workspacePath (Read Only)	Returns a path to the layer's workspace or connection file. Not all layers support the `workspacePath` property (for example, web services), so it is good practice to test for this ahead of time using the `supports` method.	String

Method Overview

Method	Explanation
findAndReplaceWorkspacePath (find_workspace_path, replace_workspace_path, {validate})	Finds and replaces a layer's workspace path with a new workspace path.
getExtent ({symbolized_extent})	Returns a layer's geometric or symbolized extent.
getSelectedExtent ({symbolized_extent})	Returns a layer's geometric or symbolized extent for selected features.
replaceDataSource (workspace_path, workspace_type, {dataset_name}, {validate})	Replaces a data source for a layer in a map document (`.mxd`) or layer (`.lyr`) file. It also provides the ability to switch workspace types (e.g., replaces a file geodatabase data source with an SDE data source).
save ()	Saves a layer (`.lyr`) file.
saveACopy (file_name, {version})	Saves a layer (`.lyr`) file to a different file name and, optionally, a previous version.
supports (layer_property)	Not all layers support the same set of properties. The **`supports`** property can be used to test which properties a layer does support.

Methods

findAndReplaceWorkspacePath (find_workspace_path, replace_workspace_path, {validate})

Parameter	Explanation	Data Type
find_workspace_path	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.	String
replace_workspace_path	A string that represents the workspace path or connection file you want to replace.	String
validate	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)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and fixing data sources with arcpy.mapping help topic.

getExtent ({symbolized_extent})

Parameter	Explanation	Data Type
symbolized_extent	A value of `True` will return the layer's symbolized extent; otherwise, it will return the geometric extent. The symbolized extent takes into account the area the symbology covers so that it does not get cut off by the data frame's boundary. (The default value is True)	Boolean

Return Value

Data Type	Explanation
Extent	

The **getExtent** method will honor a layer's definition query so if a subset of features are queried, **getExtent** will return the extent for only those features.

A symbolized extent takes into account the area of the feature's symbol when building the extent rectangle. Returning a symbolized extent may be best for cartographic results because symbols won't be cut off at the data frame's edges. A geometric extent may be best for analysis.

getSelectedExtent ({symbolized_extent})

Parameter	Explanation	Data Type
symbolized_extent	A value of `True` will return the layer's symbolized extent; otherwise, it will return the geometric extent. The symbolized extent takes into account the area the symbology covers so that it does not get cut off by the data frame's boundary. (The default value is True)	Boolean

Return Value

Data Type	Explanation
Extent	

A symbolized extent takes into account the area of the feature's symbol when building the extent rectangle. Returning a symbolized extent may be best for cartographic results because symbols won't be cut off at the data frame's edges. A geometric extent may be best for analysis.

replaceDataSource (workspace_path, workspace_type, {dataset_name}, {validate})

Parameter	Explanation	Data Type
workspace_path	A string that includes the workspace path to the new data or connection file.	String
workspace_type	A string keyword that represents the workspace type of the new data. • ACCESS_WORKSPACE — A personal geodatabase or Access workspace • ARCINFO_WORKSPACE — An ArcInfo coverage workspace • CAD_WORKSPACE —A CAD file workspace • EXCEL_WORKSPACE —An Excel file workspace • FILEGDB_WORKSPACE —A file geodatabase workspace • NONE —Used to skip the parameter • OLEDB_WORKSPACE —An OLE database workspace • PCCOVERAGE_WORKSPACE —A PC ARC/INFO Coverage workspace • RASTER_WORKSPACE —A raster workspace • SDE_WORKSPACE —An SDE geodatabase workspace • SHAPEFILE_WORKSPACE —A shapefile workspace • TEXT_WORKSPACE —A text file workspace • TIN_WORKSPACE —A TIN workspace • VPF_WORKSPACE —A VPF workspace	String
dataset_name	A string that represents the name of the dataset the way it appears in the new workspace (not the name of the layer in the TOC). If **dataset_name** is not provided, the **replaceDataSource** method will attempt to replace the dataset by finding a table with a the same name as the layer's current dataset property.	String
validate	If set to `True`, a workspace will only be updated if the **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 source to match the **workspace_path**, regardless of a valid match. In this case, if a match does not exist, then the data source would be broken. (The default value is True)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and fixing data sources with arcpy.mapping help topic.

save ()

There is a subtle difference between a layer (`.lyr`) file and a map layer (a layer in a map document). The **save** method only works when a variable references a layer file and will not work with a map layer. When a layer is loaded from a layer file it will remember the file name and use that when the **save** method is called. If a map layer is being referenced, a file name is not initially set, so you will need to use the **saveACopy** method instead.

saveACopy (file_name, {version})

Parameter	Explanation	Data Type
file_name	A string that includes the location and name of the output layer (`.lyr`) file.	String
version	A string that sets the output version number. The default value will use the current version. • 10.1 —Version 10.1/10.2 • 10.0 —Version 10.0 • 8.3 —Version 8.3 • 9.0 —Version 9.0/9.1 • 9.2 —Version 9.2 • 9.3 —Version 9.3 (The default value is None)	String

Provides an option to save a layer (`.lyr`) file to a different file name and, optionally, a previous version. Features that are not supported in prior versions of the software will be removed from the newly saved layer.

supports (layer_property)

Parameter	Explanation	Data Type
layer_property	The name of a particular layer property that will be tested. • BRIGHTNESS —A raster layer's brightness value • CONTRAST —A raster layer's contrast value • DATASETNAME —A layers dataset name the way it appears in the workspace • DATASOURCE —A layer's file path or connection file • DEFINITIONQUERY —A layer's definition query string • DESCRIPTION —A layer's description string • LABELCLASSES —A layer's list of label classes • LONGNAME —A layer's path including the group layer(s) it may be nested within • NAME —A layer's name • SERVICEPROPERTIES —Connection information for SDE and web service layers • SHOWLABELS —A Boolean indicating if a layer's lables are toggled on or off • SYMBOLOGY —A layer's symbology class • SYMBOLOGYTYPE —A layer's symbology class type • TIME —A layer's time properties • TRANSPARENCY —A layer's transparency value • VISIBLE —A Boolean indicating if a layer is toggled on or off in the TOC • WORKSPACEPATH —A layer's workspace or connection file path (The default value is name)	String

Return Value

Data Type	Explanation
Boolean	

There are numerous types of layers and not all of them support the same properties. For example, a feature layer supports a definition query whereas a raster layer does not, but a raster catalog does. Rather than creating individual layer objects for all possible layer types and property combinations, a **support** method was created to help identify which layer types support which properties. The support method gives you the option of testing the property before trying to get or set its value on a layer type that doesn't support it. The supports property will return a `true` if a layer supports that property.

Code Sample

Layer example 1

The following script will reference a layer (`.lyr`) file, find all layers called `Highways`, turns on labels, and save the results to a new layer file.

```
import arcpy
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Streets.lyr")
for lyr in arcpy.mapping.ListLayers(lyrFile):
    if lyr.name.lower() == "highways":
        lyr.showLabels = True
        lyr.saveACopy(r"C:\Project\Data\StreetsWithLabels.lyr")
del lyrFile

#Or with one less line using a wild card:

import arcpy
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Streets.lyr")
for lyr in arcpy.mapping.ListLayers(lyrFile, "Highways"):
    lyr.showLabels = True
    lyr.saveACopy(r"C:\Project\Data\StreetsWithLabels.lyr")
del lyrFile
```

Layer example 2

The following script will allow secured layers to render correctly by creating an SDE connection in memory before opening a map document that requires password information. This script simply defines the connection information, then exports the map document to a PDF file. It is good practice to delete this reference from memory before the script closes.

```
import arcpy, os

#Remove temporary connection file if it already exists
sdeFile = r"C:\Project\Output\TempSDEConnectionFile.sde"
if os.path.exists(sdeFile):
    os.remove(sdeFile)

#Create temporary connection file in memory
arcpy.CreateArcSDEConnectionFile_management(r"C:\Project\Output",
"TempConnection", "myServerName", "5151", "myDatabase",
"DATABASE_AUTH", "myUserName", "myPassword", "SAVE_USERNAME",
"myUser.DEFAULT", "SAVE_VERSION")

#Export a map document to verify that secured layers are present
mxd = arcpy.mapping.MapDocument(r"C:\Project\SDEdata.mxd")
arcpy.mapping.ExportToPDF(mxd, r"C:\Project\output\SDEdata.pdf")

os.remove(sdeFile)
del mxd
```

Layer example 3

The following script will print the name of each SDE or web service layer along with the appropriate service information. Similar to the example above, since some SDE layers may be secured with password information, a temporary SDE connection file is created. This example does not print information about non-SDE or web service layers.

```
import arcpy, os

#Remove temporary connection file if it already exists
sdeFile = r"C:\Project\Output\TempSDEConnectionFile.sde"
if os.path.exists(sdeFile):
    os.remove(sdeFile)

#Create temporary connection file in memory
arcpy.CreateArcSDEConnectionFile_management(r"C:\Project\Output",
"TempConnection", "myServerName", "5151", "myDatabase",
"DATABASE_AUTH", "myUserName", "myPassword", "SAVE_USERNAME",
"myUser.DEFAULT", "SAVE_VERSION")

#Report service properties for layers in a map that support
SERVICEPROPERTIES
mxd = arcpy.mapping.MapDocument(r"C:\Project\ServerData.mxd")
for lyr in arcpy.mapping.ListLayers(mxd):
    if lyr.supports("SERVICEPROPERTIES"):
        servProp = lyr.serviceProperties
        print "Layer name:" + lyr.name
        print "------------------------------------------------------"
        if lyr.serviceProperties["ServiceType"] != "SDE":
            print "Service Type: " + servProp.get('ServiceType',
'N/A')
            print "    URL:         " + servProp.get('URL', 'N/A')
            print "    Connection:  " + servProp.get('Connection',
'N/A')
            print "    Server:      " + servProp.get('Server', 'N/A')
            print "    Cache:       " + str(servProp.get('Cache',
'N/A'))
            print "    User Name:   " + servProp.get('UserName',
'N/A')
            print "    Password:    " + servProp.get('Password',
            print ""
        else:
            print "Service Type: " + servProp.get('ServiceType',
'N/A')
            print "    Database:       " + servProp.get('Database',
'N/A')
            print "    Server:         " + servProp.get('Server',
'N/A')
            print "    Service:        " + servProp.get('Service',
'N/A')
            print "    Version:        " + servProp.get('Version',
'N/A')
            print "    User name:      " + servProp.get('UserName',
'N/A')
            print "    Authentication: " +
servProp.get('AuthenticationMode', 'N/A')
            print ""
del mxd
```

Layer example 4

The following script modifies the symbology for a layer in a map document. It first updates the layer's symbology using a layer file on disk with the UpdateLayer function. The layer file contains a custom color ramp that is applied to the layer. Next, it verifies that the layer has graduated color symbology. Finally, the script modifies a number of the properties on the GraduatedColors symbology class and exports the result to PDF.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Census")[0]
lyr = arcpy.mapping.ListLayers(mxd, "StatePopulation", df)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\LYRs\Population.lyr")
arcpy.mapping.UpdateLayer(df, lyr, lyrFile, True)
if lyr.symbologyType == "GRADUATED_COLORS":
  lyr.symbology.valueField = "POP2000"
  lyr.symbology.numClasses = 4
  lyr.symbology.classBreakValues = [250000, 999999, 4999999, 9999999,
35000000]
  lyr.symbology.classBreakLabels = ["250,000 to 999,999", "1,000,000
to 4,999,999",
                                    "5,000,000 to 9,999,999",
"10,000,000 to 35,000,000"]
arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Output\StatePopulation.pdf")
del mxd, lyrFile
```

Layer example 5

The following script tests if a layer file supports time and if time properties have been set. It then uses time information (start time and end time) to calculate the time extent of a time-enabled layer.

```
import arcpy, datetime
lyr =
arcpy.mapping.Layer(r'C:\Project\Data\Time\TemperatureWithTime.lyr')
if lyr.supports("TIME"):
    lyrTime = lyr.time
    if lyr.time.isTimeEnabled:
        startTime = lyrTime.startTime
        endTime = lyrTime.endTime
        timeDelta = endTime - startTime
        print "Start Time: " + str(startTime)
        print "End Time: " + str(endTime)
        print "Time Extent: " + str(timeDelta)
    else:
        print "No time properties have been set on the layer"
else:
    print "Time is not supported on this layer"
```

LayerTime (arcpy.mapping)

Top

Summary

The **LayerTime** object provides access to time management operations for time-enabled layers.

Discussion

The **LayerTime** object provides information about how time is stored and configured in a time-enabled layer. The time properties on a layer can be set on the **Time** tab of the ***Layer Properties*** dialog box in ArcMap, ArcScene, or ArcGlobe. Learn more about setting time properties on a layer

The time properties on a layer are read-only. The UpdateLayerTime function allows you to replace all layer properties available on the **Time** tab of the ***Layer Properties*** dialog box using a layer (`.lyr`) file or another layer in a map document that contains time information.

Time information, such as the time fields containing the time values associated with the features, start and end time of the data, and the time-step interval, and so on, can be used for not only gaining knowledge about the time properties on the time-enabled layer but also for performing further data management and analysis tasks over time. Example one below shows how you can get the time extent of your time-enabled layer using the **startTime** and**endTime**. Example two below shows how you can formulate a time query using the time field and select a set of features based on time and then save those features to a separate feature class. Also, you can use the time information to ensure that the time specified for selecting the features lies within the start and end time of the layer.

Furthermore, you can use several LayerTime properties together to loop through the data in your time-enabled layer based on time. Example three shows how you can step through your data based on time using the**timeStepInterval** property and generate surfaces from features valid at various time steps. Note that the **timeStepInterval** property returns a EsriTimeDelta object.

Properties

Property	Explanation	Data Type
daylightSavings (Read Only)	Indicates whether the time values in the time field of the time-enabled layer were collected while observing Daylight Saving Time rules in the input time zone.	Boolean
displayDataCumulatively (Read Only)	Indicates whether or not data in the time-enabled layer is displayed cumulatively in the display.	Boolean
endTime (Read Only)	Gets the end date and time for a time-enabled layer.	DateTime
endTimeField (Read Only)	The name of the field containing the end time values. End time field is used along with the start time field to store start and end time values for features that are valid for a certain duration.	String
isTimeEnabled (Read Only)	Indicates whether or not time is enabled on the layer.	Boolean
startTime (Read Only)	Gets the start date and time for a time-enabled layer.	DateTime
startTimeField (Read Only)	The name of the field containing the time values. This field is used for features that are valid at a particular instant in time.	String
timeFormat (Read Only)	The format in which the time values were stored in the input time field. The time format is important when formulating a time query.	String
timeOffset (Read Only)	The time offset applied to the time values in your data. This value is a EsriTimeDelta object and is used to iterate over a period of time (for example, 2 days, 1 month, and so on).	EsriTimeDelta
timeStepInterval (Read Only)	The time-step interval defines the granularity of the temporal data. The time-step interval can be thought of as how often the time values were recorded in your data. This value is a EsriTimeDelta object and is used to iterate over a period of time (for example, 2 days, 1 month, and so on).	EsriTimeDelta
timeZone (Read Only)	The Time Zone set on the time-enabled layer.	String

Code Sample

LayerTime example 1

The following script tests if a layer file supports time and if time properties have been set. It then uses time information (start time and end time) to calculate the time extent of a time-enabled layer.

```
import arcpy, datetime
lyr =
arcpy.mapping.Layer(r'C:\Project\Data\Time\TemperatureWithTime.lyr')
if lyr.supports("TIME"):
    lyrTime = lyr.time
    if lyr.time.isTimeEnabled:
        startTime = lyrTime.startTime
        endTime = lyrTime.endTime
        timeDelta = endTime - startTime
        print "Start Time: " + str(startTime)
        print "End Time: " + str(endTime)
        print "Time Extent: " + str(timeDelta)
    else:
        print "No time properties have been set on the layer"
else:
    print "Time is not supported on this layer"
```

```
# Set the time for which you want to select features in the time-

timeSelection = datetime.datetime(2009, 9, 10, 12, 0)

# Get the start and end time of the time enabled layer
startTime = lyrTime.startTime
endTime = lyrTime.endTime

# Get the time field containing the time values associated with data
in the time-enabled layer
timeField = str(lyrTime.startTimeField)

# Check to see if the time for which you want to select features lies
within the start and end time of the time enabled layer
if (timeSelection < startTime or timeSelection > endTime):
    print "The time specified for selecting features is not within the
time extent of the layer"
else:
    # Formulate the time query
    timeQuery = "\"" + timeField + "\"" + "= date '" +
str(timeSelection) + "'"
    # Process: Feature Class to Feature Class
    arcpy.FeatureClassToFeatureClass_conversion(lyr, output_GDB,
"timeSubset", timeQuery, "", "")
```

LayerTime example 2

The following script creates a feature class from input features valid at a certain time, while ensuring that the selection time is within the time extent (start time and end time) of the time-enabled layer.

```
import arcpy, datetime

output_GDB = r"C:\Project\Output\Output.gdb"

lyr = arcpy.mapping.Layer(r"C:\Project\Data\Time\TimeLayer.lyr")
lyrTime = lyr.time
```

LayerTime example 3

The following script uses the time information (start time, end time, time-step interval) to step through data in a time-enabled layer to generate raster surfaces from points that are valid at each time step and then stores these rasters in a mosaic dataset.

```
import arcpy, datetime
# Check out the ArcGIS Spatial Analyst extension for using the IDW
interpolation tool
arcpy.CheckOutExtension("spatial")

# Get the layer time properties
lyr = arcpy.mapping.Layer(r"C:\Project\Data\Time\TimeLayer.lyr")
lyrTime = lyr.time

# Calculate the number of iterations based on the time extent and
timestep interval
startTime = lyrTime.startTime
endTime = lyrTime.endTime
timeExtent = endTime - startTime
timeStepInterval = lyrTime.timeStepInterval

iterations = timeExtent.days / timeStepInterval.interval

# Get the time field containing the time values associated
# with the data in the time-enabled layer
startTimeField = str(lyrTime.startTimeField)

# Specify the output mosaic dataset to which the interpolated rasters
will be added
outputMosaicDataset =
r"C:\Project\Output\Output.gdb\outputMosaicDataset"

i = 0
while i <= iterations:
    # Formulate the time query and increment the time by the
timeStepInterval
    currentTime = str(startTime + (i*timeStepInterval))
    timeQuery = "\"" + startTimeField + "\"" + " = date '" +


    # Create an in-memory feature layer containing points that are
valid at each timestep
    tempFeatureLyr = "tempTimeLayer" + str(i)
    arcpy.MakeFeatureLayer_management(lyr, tempFeatureLyr, timeQuery)

    # Create an interpolated raster surface using the points valid at
each timestep
    outRaster = r"C:\Project\Output\Output.gdb\raster" + str(i)
    print outRaster
    arcpy.gp.Idw_sa(tempFeatureLyr, "Temperature", outRaster)

    # Add the newly created raster surface to a Mosaic Dataset
    arcpy.AddRastersToMosaicDataset_management(outputMosaicDataset,
"Raster Dataset", outRaster)

    i = i + 1

# Calculate the statistics on the output Mosaic Dataset for
# classifying your data after new rasters are added
arcpy.CalculateStatistics_management(outputMosaicDataset,"1","1","#")
```

LegendElement (arcpy.mapping)

Top

Summary

The **LegendElement** object provides access to properties and methods that enable its repositioning and resizing on the page layout as well as modifying its title and legend items.

Discussion

Like a MapsurroundElement, the **LegendElement** object has an association with a single parent data frame. In addition, the **LegendElement** also has methods and properties for managing the contents within the legend. These are useful for controlling how new items get added to the legend, sizing the legend, updating legend item properties using a style item, and also for specifying the number of columns in a legend.

The ListLayoutElements function returns a Python list of page layout element objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific page element object. To return a list of only **LegendElements**, use the LEGEND_ELEMENT constant for the **element_type** parameter. A wildcard can also be used to further refine the search based on the element name.

Legend elements in a layout have a property in the ArcMap user interface called **Add a new item to the legend when a new layer is added to the map** which allows the legend to automatically update when layers are added or removed from a map. The **autoAdd** property toggles this behavior on/off so you can control whether or not a newly added layer should appear in the legend. For example, you may want to add orthophotography, but you don't want it to appear in the legend. Prior to using the AddLayer function, you would want to set the **autoAdd** property to `False`.

Legend elements in the ArcMap user interface also have a property called **Fixed Frame**. When this property is toggled on and too many legend items get added, there may not be enough room for all items to fit within the fixed area specified. When this happens, the legend elements that don't fit are replaced by a small icon that indicates the legend is overflowing. The **isOverflowing** property allows you to check for this, then resize the legend element or modify the legend items appropriately by providing a legend item style item that uses smaller fonts, for example.

The **removeItem** and **updateItem** properties serve a basic arcpy.mapping requirement: to provide the ability to remove legend items or modify their style items with custom settings. In the user interface, when you add a new layer to the table of contents and that feature gets automatically added to the legend, a default style is applied. The arcpy.mapping module allows you to update the individual legend item style items in a **LegendElement** on a page layout. This can be accomplished using the following workflow.

- Author a custom legend item style item using the **Style Manager**.
- Reference the custom legend item style item using the ListStyleItems function.
- Reference the legend element using the ListLayoutElements function.
- Update a specific legend item style item in the legend using the **updateItem** method on the **LegendElement** class.

It is recommended that each page layout element be given a unique name so that it can be easily isolated using ArcPy scripting. This is set via the **Size and Position Tab** on the Properties dialog box in ArcMap.

X and Y element positions are based on the element's anchor position, which is also set via the **Size and Position Tab** on the Properties dialog box in ArcMap. Page units can only be changed in ArcMap via **Customize** > **ArcMap Options** > **Layout View Tab**.

Properties

Property	Explanation	Data Type
autoAdd (Read and Write)	Controls whether a layer should be automatically added to the legend when using the AddLayer or AddLayerToGroup functions. This property mimics the Map Connection check box option labeled **Add a new item to legend when a new layer is added to the map**, found via the Legend Properties dialog box's Item tab.	Boolean
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The x location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
isOverflowing (Read Only)	Returns `True` if the legend items can't fit when the **Fixed Frame** option is set within legend properties.	Boolean

items (Read Only)	Returns a list of strings that represents the individual legend item names.	String
name (Read and Write)	The name of the element.	String
parentDataFrameName (Read Only)	A string that represents the name of the data frame for the associated element.	String
title (Read and Write)	The text string that represents the legend's title.	String
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Data frame element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Map surround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Method Overview

Method	Explanation
adjustColumnCount (column_count)	Provides a mechanism to set the number of columns in a legend.
listLegendItemLayers ()	Returns a list of Layer object references for every legend item in a legend.
removeItem (legend_item_layer, {index})	The **removeItem** method allows you to remove a legend item from a legend on a layout.
updateItem (legend_item_layer, {legend_item_style_item}, {preserve_item_sizes}, {use_visible_extent}, {show_feature_count}, {use_ddp_extent}, {index})	The **updateItem** method allows you to update a number of individual properties for a legend item within a legend on a layout.

Methods

adjustColumnCount (column_count)

Parameter	Explanation	Data Type
column_count	An integer that represents the desired number of columns. (The default value is 1)	Integer

There are plenty of cases where there is not enough space on a layout to fit all legend items in a single column. A legend can be interrogated using its **elementHeight** or **elementWidth** properties to determine the needed space on the page. Another method would be to count the number of items in a legend. Whichever method is used, **adjustColumnCount** can then be used to establish the number of desired columns in a legend.

listLegendItemLayers ()

Return Value

Data Type	Explanation
Layer	A Python list of Layer objects. Each legend item references a layer.

The **listLegendItemLayers** method is useful for determining the different layers and the number of times they are being used in a legend. A Layer object is returned for each legend item found in a legend. It is possible that a layer can appear in the table of contents only once but exist in the legend multiple times. By evaluating the returned list of Layer objects (one for each legend item), you can determine if the layer exists multiple times. If you want to remove or modify each of those instances, you will need to provide an **index** parameter value for both the **removeItem** and **updateItem** methods.

removeItem (legend_item_layer, {index})

Parameter	Explanation	Data Type
legend_item_layer	A reference to a layer that is used in a legend.	Layer
index	A single layer can be added into the same legend multiple times. The index value provides a way to reference a specific legend item. If you have more than one item and you want to remove all instances, then the **removeItem** will need to be called multiple times. By default the first legend item for a layer is removed. (The default value is 0)	Long

There may be times when you want to remove an item from the legend. For example, a legend item for orthophotography doesn't really make sense to have in a legend. The **removeItem** method allows you to remove an individual legend item from the legend on a layout after it has been automatically added. Another option is to use set the **autoAdd** property on a legend to control whether or not a newly added layer will get automatically added to the legend.

updateItem (legend_item_layer, {legend_item_style_item}, {preserve_item_sizes}, {use_visible_extent}, {show_feature_count}, {use_ddp_extent}, {index})

Parameter	Explanation	Data Type
legend_item_layer	A reference to a layer that is used in a legend.	Layer
legend_item_style_item	A reference to a legend item style item that is returned from the ListStyleItems function. This item must come from the style folder name Legend Items. (The default value is None)	Object
preserve_item_sizes	A Boolean that controls whether or not the symbol sizes can change if the size of the legend is changed. If set to `True`, the sizes authored in the style item will remain unchanged. (The default value is False)	Boolean
use_visible_extent	A Boolean that controls if only the features in the data frame's visible extent will be displayed in the legend. (The default value is False)	Boolean
show_feature_count	A Boolean that controls if feature counts will be displayed in the legend. (The default value is False)	Boolean
use_ddp_extent	A Boolean that controls if only the features within the Data Driven Pages index layer feature will be displayed in the legend. Data Driven Pages must be enabled. (The default value is False)	Boolean
index	A single layer can be added into the same legend multiple times. The index value provides a way to reference a specific legend item. If you have more than one item and you want to remove all instances, then the **updateItem** method will need to be called multiple times. By default the first legend item for a layer is updated. (The default value is 0)	Long

The **updateItem** method serves a basic arcpy.mapping requirement to provide the ability to update legend items with custom settings. In the user interface, when you add a new layer to the table to contents, and that feature is automatically added to the legend, a default style is applied. The arcpy.mapping module allows you to update the individual legend items in a **LegendElement** on a page layout.

Code Sample

LegendElement example 1

The following script will add layers to a new data frame within a map document that includes an inserted legend element named Legend. The layers will automatically get added to the legend except for the orthophoto. This is controlled using the **autoAdd** property. Finally, after the layers are added, the number of columns are adjusted to two.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
lyr1 = arcpy.mapping.Layer(r"C:\Project\Data\Parcels.lyr")
lyr2 = arcpy.mapping.Layer(r"C:\Project\Data\MapIndex.lyr")
lyr3 = arcpy.mapping.Layer(r"C:\Project\Data\Orthophoto.lyr")
legend = arcpy.mapping.ListLayoutElements(mxd, "LEGEND_ELEMENT",
"Legend")[0]
legend.autoAdd = True
arcpy.mapping.AddLayer(df, lyr1, "BOTTOM")
arcpy.mapping.AddLayer(df, lyr2, "BOTTOM")
legend.autoAdd = False
arcpy.mapping.AddLayer(df, lyr3, "BOTTOM")
legend.adjustColumnCount(2)
mxd.save()
del mxd
```

LegendElement example 2

The following script works with a legend element with Fixed Frame toggled on. If the legend items are overflowing, the script will increase the height of the legend by `0.1` inch until all items fully display.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
legendElm = arcpy.mapping.ListLayoutElements(mxd, "LEGEND_ELEMENT",
"Legend")[0]
while legendElm.isOverflowing:
  legendElm.elementHeight = legendElm.elementHeight + 0.1
del mxd
```

LegendElement example 3

The following script uses the workflow outlined above and updates a given legend item. A layer is added to the first data frame in the map document and the legend item will be updated with a custom legend item style item called`NewDefaultLegendStyle`. The custom `.style` file is saved in the user's profile location. Next, the script checks to see if the legend is overflowing, and if it does, it removes an unnecessary layer from the legend to make additional room.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.AddLayer(df, lyrFile, "TOP")
lyr = arcpy.mapping.ListLayers(mxd, 'Rivers', df)[0]
styleItem = arcpy.mapping.ListStyleItems("USER_STYLE", "Legend Items",
"NewDefaultLegendStyle")[0]
legend = arcpy.mapping.ListLayoutElements(mxd, "LEGEND_ELEMENT")[0]
legend.updateItem(lyr, styleItem)
if legend.isOverFlowing:
  removeLyr = arcpy.mapping.ListLayers(mxd, "County Boundary")[0]
  legend.removeItem(removeLyr)
del mxd
```

LegendElement example 4

The following script updates all layers in the legend to use a custom legend item style item called `MyNewStyle`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
legend = arcpy.mapping.ListLayoutElements(mxd,"LEGEND_ELEMENT")[0]
styleItem = arcpy.mapping.ListStyleItems("USER_STYLE",
                                         "Legend Items",
                                         "MyNewStyle")[0]
for lyr in legend.listLegendItemLayers():
     legend.updateItem(lyr, styleItem)
del mxd
```

MapDocument (arcpy.mapping)

Top

Summary

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

Discussion

The **MapDocument** object is usually one of the first object references created in a map automation script because it is a required parameter for many of the **arcpy.mapping** functions. It is through the **MapDocument** object that you ultimately can get access to almost all other objects within a map document (for example, data frames, layers, page layout elements). The **MapDocument** object provides access to most of the map document properties found in the **Map Document Properties** dialog box in ArcMap (**File** > **Map Document Properties**). The object also contains methods for managing map document thumbnails and methods for saving map documents that are found within the ArcMap File menu.

There are two different ways that a **MapDocument** object can be created using the MapDocument function. The first, and most recommended, method is to provide a system path to the location of the map document (.mxd) on disk. This technique is most versatile because then a script can be run outside an ArcGIS application. Referencing a specific map document on disk provides more control in terms of how the script will execute because a given script may not work on all map documents.

The second technique is to use the CURRENT keyword as an input parameter to the **MapDocument** function. This technique only works from within an ArcMap application because the **MapDocument** object references the map document that is currently loaded into the ArcMap application. Using CURRENT is very helpful when quickly testing and learning the scripting capabilities and command syntax within the Python window. You may start off learning the syntax in the Python window, then start pasting those lines of code into a more permanent script saved to disk.

Script tools that use the CURRENT keyword must be run from within ArcMap (either from a custom menu or the Catalog window). Script tools using CURRENT will not execute properly if run from within the ArcCatalog application. For this same reason, if a script tool has a validation script that includes a reference to CURRENT, an error may occur if you try to edit the validation script from ArcCatalog. Be sure to edit the script tool's validation code from within the ArcMap Catalog window.

To use the CURRENT keyword within a script tool, background processing must be disabled. Background processing runs all scripts as though they were being run as stand-alone scripts outside an ArcGIS application, and for this reason, CURRENT will not work with background processing enabled. There is a new script tool option called **Always run in foreground** that ensures that a script tool will run in the foreground even if background processing is enabled.

It is very important to understand how variables reference **MapDocument** objects in the scripting environment, especially when you are saving your results to a new file. You must understand that when you initially create a variable that references a **MapDocument** object, it will always point to the original map document on disk or currently in memory (via CURRENT). In the ArcMap application, if you perform a **SaveAs** to a new file location, all subsequent changes are directed to the new file. This is not possible within the scripting environment, and therefore, a **saveAs** method is not provided. The **MapDocument** class has **save** and **saveACopy** methods for managing modifications to a map document.

If scripting is used to modify the appearance of some map document elements while using the CURRENT map document (for example, change a layer name, the data frame extent, and so on), the map may not automatically update with each executed line of code. To refresh the map document to reflect the changes, use either the RefreshActiveView or RefreshTOC functions. These functions will refresh the map display or page layout and table of contents, respectively. The refresh functions are only needed if you want to see the application updated. Arcpy.mapping export, save, and printing functions will generate the expected updated results without using these functions.

Some layers within a map document or layer file may be password protected because the user and password information is not saved within the layer file or map document. Map documents that contain these layers will prompt the user to enter the appropriate information while the document is opening. The arcpy.mapping scripting environment will by default suppress these dialog boxes during execution, but that means that the layers will be treated as though they have broken data sources. In other words, secured layers will not be rendered in any output. If it is necessary for these layers to render appropriately, then there are a couple of options. First, save the user name and password information with the layers. Second, the CreateArcSDEConnectionFile geoprocessing function allows you to create a connection that persists in memory. If this command is used prior to opening a map document (.mxd) with the MapDocument function or a layer file with the Layer function, then SDE layers will render. Currently, there is not an alternative for secured Web services.

The variable that references the MapDocument object will place a lock on the map document file. It is good practice to remove the Map Document object reference using the Python del command at the end of a script or within a try/except statement.

Changing data sources in a map document is a common requirement. There are two methods on the **MapDocument** object that help with this.

The **findAndReplaceWorkspacePaths** method is intended for replacing part or all of a layer's or table's workspace path. The **replaceWorkspaces** method

allows you to change not only a path but also the workspace type. For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and fixing data sources with arcpy.mapping help topic.

Syntax

MapDocument (mxd_path)

Parameter	Explanation	Data Type
mxd_path	A string that includes the full path and file name of an existing map document (`.mxd`) or a string that contains the keyword CURRENT.	String

Properties

Property	Explanation	Data Type
activeDataFrame (Read Only)	Returns a DataFrame object that represents the currently active data frame in a map document (`.mxd`). The **`activeDataFrame`**property will return the appropriate data frame even if the map document is in page layout view. If you want to set the active data frame, use the **`activeView`** property.	DataFrame
activeView (Read and Write)	Provides the ability to set or get a map document's active view to either a single data frame or the page layout. The property works with a string that represents the active data frame name or the `PAGE_LAYOUT` keyword. If **`activeView`** is set to `PAGE_LAYOUT` and the map document is saved, the next time the map document is opened, it will be opened in layout mode. If **`activeView`** is set to a data frame name and the map document is saved, the next time the map document is opened, it will be opened in data view, and that particular data frame will be the active data frame.	String
author (Read and Write)	Provides the ability to either get or set the map document's author information.	String
credits (Read and Write)	Provides the ability to either get or set the map document's credits or copyright information.	String
dataDrivenPages (Read Only)	Returns a DataDrivenPages object that can then be used to manage the pages in a Data Driven Pages enabled map document.	DataDrivenPages
dateExported (Read Only)	Returns a Python **`datetime`** object that reports the date the last time the map document was exported. This value is only current if the map document was saved after the map was exported.	DateTime
datePrinted (Read Only)	Returns a Python **`datetime`** object that reports the date the last time the map document was printed. This value is only current if the map document was saved after the map was printed.	DateTime
dateSaved (Read Only)	Returns a Python **`datetime`** object that reports the date the last time the map document was saved.	DateTime
description (Read and Write)	Provides the ability to either get or set the map document's description information.	String
filePath (Read Only)	Returns a string value that reports the fully qualified map document path and file name.	String
hyperlinkBase (Read and Write)	Provides the ability to either get or set the base path or URL used for field-based hyperlinks to documents or URLs.	String
isDDPEnabled (Read Only)	Returns `True` if the map document is Data Driven Pages enabled.	Boolean
pageSize (Read Only)	Provides the ability to get the layout's page size. It returns a named tuple with the properties `width` and `height`. The following script shows a few different techniques to print a map document's page width and height. `mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")` `print mxd.pageSize` `print mxd.pageSize.width; print mxd.pageSize.height` `pageWidth, pageHeight = mxd.pageSize` `print pageWidth, pageHeight`	tuple
relativePaths (Read and Write)	Provides the ability to control if a map document stores relative paths to the data sources. A value of `True` sets relative paths; a value of `False` sets full paths to the data sources.	Boolean

summary (Read and Write)	Provides the ability to either get or set the map document's summary information.	String
tags (Read and Write)	Provides the ability to either get or set the map document's tag information. Separate tags with a single comma (,).	String
title (Read and Write)	Provides the ability to either get or set the map document's title information.	String

Method Overview

Method	Explanation
deleteThumbnail ()	Deletes a map document's (`.mxd`) thumbnail image
findAndReplaceWorkspacePaths (find_workspace_path, replace_workspace_path, {validate})	Finds old workspace paths and replaces them with new paths for all layers and tables in a map document (`.mxd`)
makeThumbnail ()	Creates a map document's (`.mxd`) thumbnail image
replaceWorkspaces (old_workspace_path, old_workspace_type, new_workspace_path, new_workspace_type, {validate})	Replaces an old workspace with a new workspace for all layers and tables in a map document (`.mxd`); also provides the ability to switch workspace types (for example, replace a file geodatabase data source with an SDE data source).
save ()	Saves a map document (`.mxd`)
saveACopy (file_name, {version})	Provides an option to save a map document (`.mxd`) to a new file, and optionally, to a previous version.

Methods

deleteThumbnail ()

This performs the same operation as clicking the **Delete Thumbnail** button on the **File** > **Map Document Properties** dialog box in ArcMap.

findAndReplaceWorkspacePaths (find_workspace_path, replace_workspace_path, {validate})

Parameter	Explanation	Data Type
find_workspace_path	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`**, depending on the value of the **`validate`** parameter.	String
replace_workspace_path	A string that represents the workspace path or connection file you want to use to replace.	String
validate	If set to `True`, a 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 all workspaces to match the **`replace_workspace_path`**, regardless of a valid match. In this case, if a match does not exist, then the layer and table's data sources would be broken. (The default value is True)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and Fixing Data Sources with arcpy.mapping help topic.

makeThumbnail ()

This performs the same operation as clicking the **Make Thumbnail** button in the **File** > **Map Document Properties** dialog box in ArcMap.

replaceWorkspaces (old_workspace_path, old_workspace_type, new_workspace_path, new_workspace_type, {validate})

Parameter	Explanation	Data Type
old_workspace_path	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 **new_workspace_path**, depending on the value of the **validate** parameter.	String
old_workspace_type	A string keyword that represents the workspace type of the old data to be replaced. If an empty string is passed, multiple workspaces can be redirected into one workspace. • ACCESS_WORKSPACE — A personal geodatabase or Access workspace • ARCINFO_WORKSPACE — An ArcInfo coverage workspace • CAD_WORKSPACE —A CAD file workspace • EXCEL_WORKSPACE —An Excel file workspace • FILEGDB_WORKSPACE —A file geodatabase workspace • NONE —Used to skip the parameter • OLEDB_WORKSPACE —An OLE database workspace • PCCOVERAGE_WORKSPACE —A PC ARC/INFO Coverage workspace • RASTER_WORKSPACE —A raster workspace • SDE_WORKSPACE —An SDE geodatabase workspace • SHAPEFILE_WORKSPACE —A shapefile workspace • TEXT_WORKSPACE —A text file workspace • TIN_WORKSPACE —A TIN workspace • VPF_WORKSPACE —A VPF workspace	String
new_workspace_path	A string that represents the new workspace path or connection file.	String
new_workspace_type	A string keyword that represents the workspace type that will replace the **old_workspace_type**. • ACCESS_WORKSPACE — A personal geodatabase or Access workspace • ARCINFO_WORKSPACE — An ArcInfo coverage workspace • CAD_WORKSPACE —A CAD file workspace • EXCEL_WORKSPACE —An Excel file workspace • FILEGDB_WORKSPACE —A file geodatabase workspace • OLEDB_WORKSPACE —An OLE database workspace • PCCOVERAGE_WORKSPACE —A PC ARC/INFO Coverage workspace • RASTER_WORKSPACE —A raster workspace • SDE_WORKSPACE —An SDE geodatabase workspace • SHAPEFILE_WORKSPACE —A shapefile workspace • TEXT_WORKSPACE —A text file workspace • TIN_WORKSPACE —A TIN workspace • VPF_WORKSPACE —A VPF workspace	String
validate	If set to `True`, a workspace will only be updated if the **new_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 all workspaces to match the **new_workspace_path**, regardless of a valid match. In this case, if a match does not exist, then the data sources would be broken. (The default value is True)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and Fixing Data Sources with arcpy.mapping help topic.

save ()

This performs the same operation as **File** > **Save** in ArcMap.

saveACopy (file_name, {version})

Parameter	Explanation	Data Type
file_name	A string that includes the location and name of the output map document (`.mxd`).	String
version	A string that sets the output version number. The default value will use the current version. • 10.1 —Version 10.1/10.2 • 10.0 —Version 10.0 • 8.3 —Version 8.3 • 9.0 —Version 9.0/9.1 • 9.2 —Version 9.2 • 9.3 —Version 9.3 (The default value is None)	String

This performs the same operation as **File** > **SaveACopy** in ArcMap. Features that are not supported in prior versions of the software will be removed from the newly saved map document.

Code Sample

MapDocument example 1

The following script creates a separate MXD file for each data frame in a map document. The output map documents will be saved in data view mode so when each map document is opened, the corresponding data frame will be active data frame. The script also sets the title property of each output map document. Because this script uses a system path to the map document, it can be executed outside an ArcMap application. Note: Python strings cannot end with a backslash, even when the string is preceded by an `r`. You must use a double backslash. This becomes important when appending dynamic file names to a folder path.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    mxd.activeView = df.name
    mxd.title = df.name
    mxd.saveACopy(r"C:\Project\Output\\" + df.name + ".mxd")
del mxd
```

MapDocument example 2

The following script demonstrates how the `CURRENT` keyword can be used within the Python window. This sample will update the first data frame's name and refresh the table of contents so the change can be see in the application. Paste the following code into the Python window within a new ArcMap document.

```
mxd = arcpy.mapping.MapDocument("CURRENT")
arcpy.mapping.ListDataFrames(mxd)[0].name = "New Data Frame Name"
arcpy.RefreshTOC()
del mxd
```

When pasted into the interactive window it will appear as follows. The three dots to the left of the code block indicate that the lines are a single block of code that will be executed together. You must press the Enter key to execute these lines.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
... arcpy.mapping.ListDataFrames(mxd)[0].name = "New Data Frame Name"
... arcpy.RefreshTOC()
... del mxd
...
```

MapDocument example 3

The following is another simple script that demonstrates the use of the `CURRENT` keyword within the Python window. Each layer name will be printed to the Python window. Loops are also possible, provided that you maintain the correct indentation. Similar to the example above, paste the code below into the Python window.

```
mxd = arcpy.mapping.MapDocument("CURRENT")
for lyr in arcpy.mapping.ListLayers(mxd):
    print lyr.name
del mxd
```

When pasted into the interactive window it will appear as follows. Again, press the Enter key to execute the lines.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
... for lyr in arcpy.mapping.ListLayers(mxd):
...     print lyr.name
... del mxd
...
```

MapDocument example 4

The following script will allow secured layers to render correctly by creating an SDE connection in memory before opening a map document that requires password information. This script simply defines the connection information and exports the map document to a PDF file. It is good practice to delete this reference from memory before the script closes.

```
import arcpy, os

#Remove temporary connection file if it already exists
sdeFile = r"C:\Project\Output\TempSDEConnectionFile.sde"
if os.path.exists(sdeFile):
    os.remove(sdeFile)

#Create temporary connection file in memory
arcpy.CreateArcSDEConnectionFile_management(r"C:\Project\Output",
"TempConnection", "myServerName", "5151", "myDatabase",
"DATABASE_AUTH", "myUserName", "myPassword", "SAVE_USERNAME",
"myUser.DEFAULT", "SAVE_VERSION")

#Export a map document to verify that secured layers are present
mxd = arcpy.mapping.MapDocument(r"C:\Project\SDEdata.mxd")
arcpy.mapping.ExportToPDF(mxd, r"C:\Project\output\SDEdata.pdf")

os.remove(sdeFile)
del mxd
```

MapsurroundElement (arcpy.mapping)

Top

Summary

Provides access to properties that enables its repositioning on the page layout as well as identifying its parent data frame.

Discussion

A **`MapsurroundElement`** object is a page element that has an association with a data frame. For example, there is a one-to-one relationship between a north arrow and a data frame. Mapsurround elements also include scale text and scale bars items. There is a property called **`parentDataFrameName`** that allows you to find the elements that are associated with a particular data frame. A legend element is also an example of a mapsurround, but because it has additional properties, it is a separate element type. The ListLayoutElements function returns a Python list of page layout element objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific page element object. To return a list of only **`MapsurroundElements`**, use the MAPSURROUND_ELEMENT constant for the **`element_type`** parameter. A wildcard can also be used to further refine the search based on the element name. It is recommended that each page layout element be given a unique name so that it can be easily isolated using arcpy scripting. X and Y element positions are based on the element's anchor position, which is set via the **Size and Position** tab on the Elements Properties dialog box in ArcMap.

Properties

Property	Explanation	Data Type
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The x location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
name (Read and Write)	The name of the element.	String
parentDataFrameName (Read Only)	A string that represents the name of the DataFrame for the associated element.	String
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Dataframe element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Mapsurround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Code Sample

MapsurroundElement example

The following script will find the mapsurround element named `ScaleBar` and change it's position.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
scaleBar = arcpy.mapping.ListLayoutElements(mxd,
"MAPSURROUND_ELEMENT", "ScaleBar")[0]
df = arcpy.mapping.ListDataFrames(mxd,
scaleBar.parentDataFrameName)[0]
scaleBar.elementPositionX = df.elementPositionX + (df.elementWidth /
2)
mxd.save()
del mxd
```

PDFDocument (arcpy.mapping)

Top

Summary

Allows manipulation of PDF documents, including facilities for merging pages, deleting pages, setting document open behavior, adding file attachments, and creating or changing document security settings.

Discussion

PDFDocumentOpen and PDFDocumentCreate are two functions that provide a reference to a **PDFDocument** object.

When using the **appendPages**, **insertPages**, or **attachFile** methods, it is important to pass a string that represents the path to an existing PDF file. If a **PDFDocument** object is passed, the method will fail. It is also required that PDF security be the same for all documents that are being appended, inserted, or attached. For example, if you want to append three documents into one multipage PDF and only the first document is protected with a password, the other two documents must also first be given the same password before **appendPages** is used.

The **deletePages** method is useful for swapping out only the pages that have been modified. It may take a long time to process dozens of pages. If only a relative few have been modified, it is faster to delete only those pages, then insert the newly updated pages using the **insertPages** method.

For more discussion on how to create map books, see the Building Map Books with ArcGIS help topic.

Syntax

PDFDocument (pdf_path)

Parameter	Explanation	Data Type
pdf_path	A string that specifies the path and file name for the resulting PDF file when the **saveAndClose** method is called.	String

The **PDFDocumentCreate** function receives a path to determine the save location and file name where a new PDF file will be created. However, no PDF file will be created until subsequent steps are performed to insert or append pages and save the PDF file. **PDFDocumentCreate** will return a PDFDocument object that your script should then manipulate and save. A common scenario for using this function is the creation of a PDF map book. The steps typically involve exporting a number of separate PDF files from map documents, creating a new PDFDocument object, appending content from the exported PDF files and other documents, and saving the final PDF map book.

Please note that it is not possible to create blank PDF files, nor does the **PDFDocumentCreate** function add any blank pages to the document contents. For the **saveAndClose** method to successfully create a file, content must be added to the PDFDocument object using the **appendPages** or **insertPages** methods.

For more discussion on how to create map books, see the Building Map Books with ArcGIS help topic.

PDFDocumentCreate example

This script will create a new PDF document, append the contents of three separate PDF documents, and save the resulting PDF file.

```
import arcpy, os

#Set file name and remove if it already exists
pdfPath = r"C:\Project\ParcelAtlasMapBook.pdf"
if os.path.exists(pdfPath):
    os.remove(pdfPath)

#Create the file and append pages
pdfDoc = arcpy.mapping.PDFDocumentCreate(pdfPath)
pdfDoc.appendPages(r"C:\Project\Title.pdf")
pdfDoc.appendPages(r"C:\Project\ParcelAtlas.pdf")
pdfDoc.appendPages(r"C:\Project\ContactInfo.pdf")

#Commit changes and delete variable reference
pdfDoc.saveAndClose()
del pdfDoc
```

Properties

Property	Explanation	Data Type
pageCount (Read Only)	Returns an integer that represents the total number of pages in the PDF document	Long

Method Overview

Method	Explanation
appendPages (pdf_path, {input_pdf_password})	Appends one PDF document to the end of another
attachFile (file_path, {description})	Attaches files to existing PDF documents (Attachments are then accessible to users when the PDF file is opened in a PDF viewer application.)
deletePages (page_range)	Provides the ability to delete one or multiple pages within an existing PDF document.
insertPages (pdf_path, {before_page_number}, {input_pdf_password})	Allows inserting the contents of one PDF document at the beginning or in between the pages of another PDFDocument
saveAndClose ()	Saves any changes made to the currently referenced **`PDFDocument`**
updateDocProperties ({pdf_title}, {pdf_author}, {pdf_subject}, {pdf_keywords}, {pdf_open_view}, {pdf_layout})	Allows updating of the PDF document metadata and can also set the certain behaviors that will trigger when the document is opened in Adobe Reader or Adobe Acrobat, such as the initial view mode and the page thumbnails view
updateDocSecurity (new_master_password, {new_user_password}, {encryption}, {permissions})	Provides the mechanism that sets password, encryption, and security restrictions on PDF files.

Methods

appendPages (pdf_path, {input_pdf_password})

Parameter	Explanation	Data Type
pdf_path	A string that includes the location and name of the input PDF document to be appended	String
input_pdf_password	A string that defines the master password to a protected file (The default value is None)	String

When appending secured PDF documents that each have different security settings, the output settings will be based on the primary document that pages are being appended to. For example, if the document that is being appended to does not have password information saved, but the appended pages do, the resulting document will not have password information saved.

attachFile (file_path, {description})

Parameter	Explanation	Data Type
file_path	A string that includes the location and name of the file to be attached to the PDF document.	String
description	An optional string to be used as a description for the attachment. The user will see this string when viewing the attachment in a PDF viewer application.	String

Use the attachFile method to attach any type of file to a PDF document.

deletePages (page_range)

Parameter	Explanation	Data Type
page_range	A string that defines the page or pages to be deleted. Delete a single page by passing in a single value as a string (for example,"3"). Multiple pages can be deleted using a comma between each value (for example, "3, 5, 7"). Ranges can also be applied (for example, "1, 3, 5-12").	String

It is important to keep track of the pages that are being deleted, because each time pages are deleted, the internal PDF page numbers are automatically adjusted. For example, page 3 becomes page 2 immediately after page 1 or page 2 are deleted. If page 1 and page 2 are deleted, page 3 becomes page 1. You need to consider this if you are using **deletePages** and then immediately using **appendPages** or **insertPages**.

insertPages (pdf_path, {before_page_number}, {input_pdf_password})

Parameter	Explanation	Data Type
pdf_path	A string that includes the location and name of the input PDF document to be inserted.	String
before_page_number	An integer that defines a page number in the currently referenced **PDFDocument** before which the new page(s) will be inserted. For example, if the **before_page_value** is 1, the inserted page will be inserted before all pages. (The default value is 1)	String
input_pdf_password	A string that defines the master password to a protected file. (The default value is None)	String

To add pages to the end of the current PDFDocument, use appendPages instead.

When inserting secured PDF documents that have different security settings, the output settings will be based on the primary document that pages are being inserted into. For example, if the document that is being inserted into does not have password information saved, but the inserted pages do, the resulting document will not have password information saved.

saveAndClose ()

The **saveAndClose** method must be used for changes to be maintained. If a script exits before **saveAndClose** is executed, changes will not be saved.

updateDocProperties ({pdf_title}, {pdf_author}, {pdf_subject}, {pdf_keywords}, {pdf_open_view}, {pdf_layout})

Parameter	Explanation	Data Type
pdf_title	A string defining the document title, a PDF metadata property. (The default value is None)	String
pdf_author	A string defining the document author, a PDF metadata property. (The default value is None)	String
pdf_subject	A string defining the document subject, a PDF metadata property. (The default value is None)	String
pdf_keywords	A string defining the document keywords, a PDF metadata property. (The default value is None)	String
pdf_open_view	A string or number that will define the behavior to trigger when the PDF file is viewed. The default value is USETHUMBS, which will show the Adobe Reader Pages panel automatically when the PDF is opened. • VIEWER_DEFAULT —Uses the application user preference when opening the file • USE_NONE —Displays the document only; does not show other panels • USE_THUMBS —Displays the document plus the Pages panel • USE_BOOKMARKS —Displays the document plus the Bookmarks panel • FULL_SCREEN —Displays the document in full-screen viewing mode • LAYERS —Displays the document plus the layers panel • ATTACHMENT —Displays the document plus the attachment panel (The default value is USE_THUMBS)	String
pdf_layout	A string or number that will define the initial view mode to trigger when the PDF file is viewed. • DONT_CARE —Uses the application user preference when opening the file	String

	• SINGLE_PAGE —Uses single-page mode • ONE_COLUMN —Uses one-column continuous mode • TWO_COLUMN_LEFT —Uses two-column continuous mode with first page on left • TWO_COLUMN_RIGHT —Uses two-column continuous mode with first page on right • TWO_PAGE_LEFT —Uses two-page mode left • TWO_PAGE_RIGHT —Uses two-page mode right (The default value is SINGLE_PAGE)	

A **pdf_open** setting of FULL_SCREEN will prompt a warning about full-screen mode when the PDF is opened. Setting **pdf_open** to a different option will not clear this setting unless **pdf_open** is set to USE_NONE.

updateDocSecurity (new_master_password, {new_user_password}, {encryption}, {permissions})

Parameter	Explanation	Data Type
new_master_password	A string that defines the master document password. This password is required for appending and inserting pages into a secured PDF.	String
new_user_password	A string that defines the user password needed to open the PDF document for viewing. (The default value is None)	String
encryption	A string that defines the encryption technique used on the PDF. • "AES_V1" —Uses 128-bit AES encryption (Acrobat 7.0 compatible) • "AES_V2" —Uses 256-bit AES encryption (Acrobat 9.0 compatible) • "RC4" —Uses 128-bit RC4 encryption (Acrobat 5.0 compatible) (The default value is RC4)	String
permissions	A string that defines the capabilities restricted by the document security settings. The **permissions** argument can accept a list of strings describing all the options to be restricted. The document restrictions can be viewed in Adobe Acrobat in the Document Properties Document Restrictions Summary page. • "ALL" —Grants all permissions • "ALL_MASTER" —Grants permissions for COPY, EDIT, EDIT_NOTES, and HIGH_PRINT • "COPY" —Grants permission to copy information from the document to the clipboard	String

	• "DOC_ASSEMBLY" —Grants permission to perform page insert, delete, and rotate, and allows creation of bookmarks and thumbnails • "EDIT" —Grants permission to edit the document in ways other than adding or modifying text notes • "EDIT_NOTES" —Grants permission to add, modify, and delete text notes • "FILL_AND_SIGN" —Grants permission to fill in or sign existing form or signature fields • "HIGH_PRINT" —Grants permission for high-quality printing • "OPEN" —Grants permission to open or decrypt the document • "PRINT" —Grants permission to print the document • "SECURE" —Grants permission to change the document's security settings (The default value is ALL)	

A password on a secured PDF document can be removed simply by setting the **new_master_password** and **new_user_password** properties to empty strings.

Code Sample

PDFDocument example 1

This script will create a new PDF document, append the contents of two separate PDF documents, and save the resulting PDF file.

```
import arcpy, os

#Set file name and remove if it already exists
pdfPath = r"C:\Project\ParcelAtlasMapBook.pdf"
if os.path.exists(pdfPath):
    os.remove(pdfPath)

#Create the file and append pages
pdfDoc = arcpy.mapping.PDFDocumentCreate(pdfPath)
pdfDoc.appendPages(r"C:\Project\Title.pdf")
pdfDoc.appendPages(r"C:\Project\ParcelAtlas.pdf")

#Commit changes and delete variable reference
pdfDoc.saveAndClose()
del pdfDoc
```

PDFDocument example 2

The following script modifies the PDF document metadata properties for the document created in example 1 above and sets the style in which the document will open.

```
import arcpy
pdfDoc =
arcpy.mapping.PDFDocumentOpen(r"C:\Project\ParcelAtlasMapBook.pdf")
pdfDoc.updateDocProperties(pdf_title="Atlas Map",
                           pdf_author="Author",
                           pdf_subject="Map Book",
                           pdf_keywords="Atlas; Map Books",
                           pdf_open_view="USE_THUMBS",
                           pdf_layout="SINGLE_PAGE")
pdfDoc.saveAndClose()
del pdfDoc
```

PDFDocument example 3

The following script will set the **user** password to `esri`, encrypt the PDF using `RC4` compression, and require a password when the document is opened.

```
import arcpy
pdfDoc =
arcpy.mapping.PDFDocumentOpen(r"C:\Project\ParcelAtlasMapBook.pdf")
pdfDoc.updateDocSecurity("master", "user", "RC4", "OPEN")
pdfDoc.saveAndClose()
del pdfDoc
```

PDFDocument example 4

The following pages will append a final page to the end of the already existing multipage document. The master password is required to make this change to a secured PDF document.

```
import arcpy
pdfDoc =
arcpy.mapping.PDFDocumentOpen(r"C:\Project\ParcelAtlasMapBook.pdf",
"master")
pdfDoc.appendPages(r"C:\Project\ContactInfo.pdf")
pdfDoc.saveAndClose()
del pdfDoc
```

PDFDocument example 5

The following script will replace a total of four pages in an existing PDF using **deletePages** followed by **insertPages**. Note how we insert the new page 3 before the current page 3 which was really page 4 before the original page 3 was removed. The same applies to the range of pages 5–7.

```
import arcpy
pdfDoc =
arcpy.mapping.PDFDocumentOpen(r"C:\Project\ParcelAtlasMapBook.pdf",
"master")
pdfDoc.deletePages("3, 5-7")
pdfDoc.insertPages(r"C:\Project\NewPage3.pdf", 3, "master")
pdfDoc.insertPages(r"C:\Project\NewPages5-7.pdf", 5, "master")
pdfDoc.saveAndClose()
del pdfDoc
```

PictureElement (arcpy.mapping)

Top

Summary

Provides access to picture properties that enable the repositioning of a picture on the page layout as well as getting and setting its data source.

Discussion

The **PictureElement** object represents a raster or image that has been inserted into the page layout. The ListLayoutElements function returns a Python list of page layout element objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific page element object. To return a list of only **PictureElements**, use the PICTURE_ELEMENT constant for the **element_type** parameter. A wild card can also be used to further refine the search based on the element name.

It is recommended that each page layout element be given a unique name so that it can be easily isolated in Python. X,Y element positions are based on the element's anchor position, which is set using the **Size and Position** tab on the **Elements Properties** dialog box in ArcMap.

The **PictureElement** object has a **sourceImage** property that allows you to read or modify the picture source location. If you plan on replacing a picture with pictures of different sizes and aspect ratios, author the map document with a picture that has a height and width set that represents the complete area you want all other pictures to occupy on the page layout. When a picture is replaced using the **sourceImage** property and has a different aspect ratio, it will be fit to the area of the original picture using the longest dimension. Replaced pictures will never be larger than the original authored size. You will also want to set the anchor position so that all new pictures are fit relative to that location. If you don't want pictures to be skewed, make sure the **Preserve Aspect Ratio** option is checked.

Properties

Property	Explanation	Data Type
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The x location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
name (Read and Write)	The name of the element.	String
sourceImage (Read and Write)	A text string that represents the path to the image data source.	String
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Data frame element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Map surround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Code Sample

PictureElement example 1

The following script will find an image by name and set its data source to a new location.

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

PictureElement example 2

The following script demonstrates how different pictures can be switched out for each page in a Data Driven Pages enabled map document. There is a different picture for each page. The pictures are named Photo1.png, Photo2.png, Photo3.png, etc to match the corresponding page numbers.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
pict = arcpy.mapping.ListLayoutElements(mxd, "PICTURE_ELEMENT",
"photo")[0]
for pageNum in range(1, mxd.dataDrivenPages.pageCount + 1):
    mxd.dataDrivenPages.currentPageID = pageNum
    pict.sourceImage = r"C:\Project\Data\Photo{0}.png".format(pageNum)
    print("Exporting page {0} of {1}"
          .format(mxd.dataDrivenPages.currentPageID,
                  mxd.dataDrivenPages.pageCount))
    arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Page{0}.pdf".format(pageNum))
del mxd
```

RasterClassifiedSymbology (arcpy.mapping)

Top

Summary

The **RasterClassifiedSymbology** class provides access to different properties that allow you to change the appearance of a layer's raster classified symbology.

Discussion

The **RasterClassifiedSymbology** class provides access to a limited number of properties and methods that allow you to automate layer symbology in a map document (`.mxd`) or layer file (`.lyr`). Basic operations, such as changing the number of classes, modifying class break values and labels, or changing the field that the symbology is based on, are some of the properties that can be modified. For access to the complete set of layer symbology properties and settings, for example, changing individual symbols for individual classes, it is necessary to author these changes in the ArcMap user interface and then save those changes to a layer file. These custom settings can then be applied to existing layers using the UpdateLayer function.

A layer can use any number of symbologies but not all of them can be modified. Not all layers will use the **RasterClassifiedSymbology** class, so it is important to first test if the layer is using this symbology class before attempting to make changes to its properties. The **symbologyType** property on the Layer class was designed for this purpose. First test if the **symbologyType** for the layer is raster classified (`if lyr.symbologyType == "RASTER_CLASSIFIED":`), then create a variable reference to the **RasterClassifiedSymbology** class for that layer (`lyrSymbolClass = lyr.symbology`).

The **symbologyType** on the Layer object is a read-only property. In other words, you can't change a raster classified symbology to a raster unique value symbology. You can only change the properties for a specific symbology class on a layer. The only way to change the symbology type is by publishing the desired result to a layer file and using the UpdateLayer function.

The classification method cannot be changed. In cases where you want to use different classification methods, you would need to preauthor layer files and use those to update a layer, then modify the properties that can be changed. The only exception to this rule is when you set **classBreakValues**. Similar to the ArcMap user interface, explicitly setting **classBreakValues** will automatically set the classification method to `manual`. Also, similar to the ArcMap user interface, once the classification method is set to `manual`, you can't change the **numClasses** parameter.

Unlike the ArcMap user interface, you set a minimum value when setting the **classBreakValues** parameter. The first value in the **classBreakValues** list is the minimum value; all other values are the class break values as they appear in the ArcMap use interface. For this reason, the **classBreakValues** list will always have one more value than the **classBreakLabels** and **classBreakDescriptions** lists.
Setting one parameter will often modify other parameters automatically. For example, if you set **numClasses**, **normalization**, or the **valueField** parameters, the **classBreakValues**, **classBreakLabels**, and**classBreakDescriptions** properties will automatically be adjusted based on the current classification method. For this reason, the order in which properties are modified is important.

The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). For example, let's say you want to update the color properties of the symbology of a layer in a map document with the symbology stored in a layer file. However, the layer in the map document and the layer file have different data sources. The minimum and maximum values and class breaks in the layer file may be different than the layer in the map document. Updating the symbology of the layer in the map document with the symbology stored in the layer file may produce unintended results (for example, the class break values will match the layer file's data source statistics as opposed to the map document layer's data source statistics. However, if you follow UpdateLayer with the **reclassify()** method, the end result is like using the color properties from the symbology in the layer file, but other characteristics are based on the map document layer's underlying source data.

If you are making these symbology modifications via the Python Window and you are referencing a map document using `CURRENT`, you may not immediately see the changes in the ArcMap application. To refresh the map document, try using the RefreshActiveView and RefreshTOC functions.

Properties

Property	Explanation	Data Type
classBreakDescriptions (Read and Write)	A sorted list of strings that represent the descriptions for each class break value that can optionally appear in a map document's legend. These values are only accessible in the ArcMap user interface by right-clicking a symbol displayed within the **Symbology**tab in the **Layer Properties** dialog box and selecting **Edit Description**. The number of descriptions in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties, so it is best practice to set these values last.	List
classBreakLabels (Read and Write)	A sorted list of strings that represent the labels for each class break that appears in the map document's table of contents and/or legend items. The number of labels in the sorted list must always be one less than the number of **classBreakValues**. This is because the **classBreakValues** list also includes a minimum value which you don't see in the user interface. These values are affected by changes to nearly all other class properties, so it is best practice to set these values last.	List
classBreakValues (Read and Write)	A sorted list of doubles that includes the minimum and maximum values that represent the class breaks. When setting**classBreakValues**, it will automatically set the **numClasses** property and will also set the classification method to `manual` as well as update other properties such as **classBreakLabels**. Unlike the ArcMap user interface, you have the ability to set a minimum value. The first value in the sorted list represents the minimum value, and the remaining values are the class breaks that appear in the user interface; therefore, there will always be one more item in the **classBreakValues** list than in the**classBreakLabels** and **classBreakDescriptions** lists. Changing this value will automatically adjust other symbology properties based on the new information.	List
excludedValues (Read and Write)	A string that represents values or ranges to exclude from the classification separated by semicolons. For example `1; 3; 5-7; and 8.5-12.1`. The values are removed from the classification, and therefore, will not be displayed.	String
normalization (Read and Write)	A string that represents a valid dataset field name used for normalization. Changing this value will automatically adjust other symbology properties based on the new information. The normalization field can be removed by setting the value to `None` (for example, `lyr.symbology.normalization = None`).	String
numClasses (Read and Write)	A long integer that represents the number of classes to be used with the current classification method. Changing this value will overwrite other symbol properties such as **classBreakValues** and **classBreakLabels**. This value cannot be set if the classification method is `manual`; therefore, **numClasses** should not be called after the **classBreakValues** property because it will automatically set the classification method to `manual`. Changing this value will automatically adjust other symbology properties based on the new information.	
valueField (Read and Write)	A string that represents a valid dataset field name used for the layer's classification symbology. Changing this value will automatically adjust other symbology properties based on the new information.	String

Method Overview

Method	Explanation
reclassify ()	Resets the layer's symbology to the layer's data source information and statistics.

Methods

reclassify ()

The **reclassify** method updates a layer's symbology properties based on the underlying source data. It is useful when a layer's symbology is updated using the UpdateLayer function with symbology stored in another layer or layer file (`.lyr`). This method will automatically update the symbology properties based on the layer's actual data source information and statistics and not the information that is persisted in a layer file. The method needs to be used cautiously because it has the potential to overwrite other symbology properties.
The **reclassify** method will regenerate **classBreakValues**, **classBreakLabels**, and **classBreakDescriptions**. It will not affect**numClasses** or **normalization**. The **reclassify** method has no affect on a `manual` classification.

Code Sample

RasterClassifiedSymbology example 1

The following script modifies the symbology for a layer in the current map document. It first verifies that the layer has raster classified symbology. Finally, it modifies a number of the properties on the **RasterClassifiedSymbology** class and updates the display.
Since this sample uses the `CURRENT` keyword to access the map document, it must be executed from within ArcMap. This is because the **MapDocument** object references the map document that is currently loaded into the ArcMap application. For more information regarding accessing map documents, see the MapDocument class help topic.

```
import arcpy
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd, "Lakes")[0]
lyr = arcpy.mapping.ListLayers(mxd, "lakene.tif", df)[0]
if lyr.symbologyType == "RASTER_CLASSIFIED":
  lyr.symbology.classBreakValues = [1, 60, 118, 165, 255]
  lyr.symbology.classBreakLabels = ["1 to 60", "61 to 118",
                                    "119 to 165", "166 to 255"]
  lyr.symbology.classBreakDescriptions = ["Class A", "Class B",
                                          "Class C", "Class D"]
  lyr.symbology.excludedValues = '0'
arcpy.RefreshActiveView()
arcpy.RefreshTOC()
del mxd, df, lyr
```

RasterClassifiedSymbology example 2

The following script modifies the symbology for a layer in a map document on disk. It first updates the layer's symbology using a layer file on disk with the UpdateLayer function. The layer file contains a custom color ramp that is applied to the layer. Next, it verifies that the layer has raster classified symbology. The script then calls the **reclassify** method. This method will update the symbology properties based on the layer's actual data source information and statistics and not the information that is persisted in a layer file. Finally, the result is exported to a PDF file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\StudyArea.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Lakes")[0]
lyr = arcpy.mapping.ListLayers(mxd, "lakene.tif", df)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Lakes\lake_blue.lyr")
arcpy.mapping.UpdateLayer(df, lyr, lyrFile, True)
if lyr.symbologyType == "RASTER_CLASSIFIED":
  lyr.symbology.reclassify()
arcpy.mapping.ExportToPDF(mxd, r"C:\Project\Output\Lakes.pdf")
del mxd, df, lyr, lyrFile
```

StyleItem (arcpy.mapping)

Top

Summary

Provides access to StyleItem class properties.

Discussion

The symbols you choose to display features, elements, or graphics in your map are stored in style files (.style). You use the ***Style Manager*** dialog box to create, view, and modify styles and their contents. Styles are organized into a defined set of categories or folder names, for example, `Colors`, `Legend Items`, `Marker Symbols`, and `Scale Bars`. Individual style items are stored in each category or folder name. For example, the `Legend Items`folder name has several different legend item style items, and each item has a different format, font size, and font type for the different elements that are displayed for a legend item.

The **`StyleItem`** class was introduced to serve a basic arcpy.mapping requirement to provide the ability to update legend item style items with custom settings. In the user interface, when you add a new layer to the table to contents, and that feature is automatically added to the legend, a default style is applied. The arcpy.mapping module allows you to update the individual legend item style items in a LegendElement on a page layout. This can be accomplished using the following workflow.

- Author a custom legend item style item using the ***Style Manager***.
- Reference the custom legend item style item using the ListStyleItems function.
- Reference the legend element using the ListLayoutElements function.
- Update a specific legend item style item in the legend using the **`updateItem`** method on the LegendElement class.

Properties

Property	Explanation	Data Type
itemName (Read Only)	A string that represents the name of the item the way it appears in the ***Style Manager*** dialog box window.	String
itemCategory (Read Only)	A string that represents the category of the item the way it appears in the ***Style Manager*** dialog box window. Categories are used to group symbols within a style.	String
styleFolderName (Read Only)	A string that represents the name of the folder the **`StyleItem`** is stored in, for example, Legend Items. Identifying the**`styleFolderName`** allows you to determine the type of the style object you have referenced.	String

Code Sample

StyleItem example

The following script uses the workflow outlined above and updates a legend's legend item style item. A layer is added to the first data frame in the map document and the legend item style item will be updated with a custom legend item style item called `NewDefaultLegendStyle`. The custom `.style` file is saved in the user's profile location.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.AddLayer(df, lyrFile, "TOP")
lyr = arcpy.mapping.ListLayers(mxd, 'Rivers', df)[0]
styleItem = arcpy.mapping.ListStyleItems("USER_STYLE", "Legend Items",
"NewDefaultLegendStyle")[0]
legend = arcpy.mapping.ListLayoutElements(mxd, "LEGEND_ELEMENT")[0]
legend.updateItem(lyr, styleItem)
del mxd
```

TableView (arcpy.mapping)

Top

Summary

Provides access to basic table properties.

Discussion

The **TableView** object is essential for managing stand-alone tables that reside within a map document (.mxd). It provides access to basic table properties such as data source information and setting a table's definition query. In addition to the TableView constructor function, the ListTableViews function also provides the ability to reference a **TableView** object.

The TableView constructor function allows you to reference a table outside of a map document within a workspace. This provides the ability to add an external table into a map document using the AddTableView function. This function is similar to MakeTableView, but the difference is that the result from MakeTableView can't be added permanently into a map document.

The ListTableViews function returns a Python list of **TableView** objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific **TableView** object. Tables can be searched within an entire map document or within a specific data frame. Wildcards can also be used to limit the search.

A definition query will not work for all workspaces. It is important to use the correct SQL syntax when working with different workspaces. For example, file geodatabases and shapefiles have double quotes around a field name (for example, "field_name"), personal geodatabases have brackets (for example, [field_name]), and SDE connections don't have any special characters (for example, field_name). For more information on updating workspaces and data sources in a map document or layer file, please refer to the Updating and Fixing Data Sources with arcpy.mapping help topic.

Tables can be removed from a map document using the RemoveTableView function.

Syntax

TableView (table_view_data_source)

Parameter	Explanation	Data Type
table_view_data_source	A string that includes the full workspace path, including the name of the table.	String

Properties

Property	Explanation	Data Type
datasetName (Read Only)	Returns the name of the table's dataset the way it appears in the workspace, not in the table of contents.	String
dataSource (Read Only)	Returns table's data source path. It includes the **workspacePath** and the **datasetName** combined.	String
definitionQuery (Read and Write)	Provides the ability to get or set a tables's definition query.	String
name (Read and Write)	Provides the ability to set or get the name of a table the way it would appear in the ArcMap table of contents. Spaces can be included.	String
isBroken (Read Only)	Returns True if a table's data source is broken.	Boolean
workspacePath (Read Only)	Returns a path to the table's workspace or connection file.	String

Method Overview

Method	Explanation
findAndReplaceWorkspacePath (find_workspace_path, replace_workspace_path, {validate})	Replaces a table's workspace with a new workspace path
replaceDataSource (workspace_path, workspace_type, {dataset_name}, {validate})	Replaces a table's data source in a map document (.mxd); also provides the ability to switch workspace types (for example, replace a file geodatabase workspace with an SDE workspace).

Methods

findAndReplaceWorkspacePath (find_workspace_path, replace_workspace_path, {validate})

Parameter	Explanation	Data Type
find_workspace_path	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.	String
replace_workspace_path	A string that represents the workspace path or connection file you want to use to replace.	String
validate	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 table's data source would be broken. (The default value is True)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and fixing data sources help topic.

replaceDataSource (workspace_path, workspace_type, {dataset_name}, {validate})

Parameter	Explanation	Data Type
workspace_path	A string that includes the workspace path to the new data or connection file.	String
workspace_type	A string keyword that represents the workspace type of the new data. • ACCESS_WORKSPACE — A personal geodatabase or Access workspace • ARCINFO_WORKSPACE — An ArcInfo coverage workspace • CAD_WORKSPACE —A CAD file workspace • EXCEL_WORKSPACE —An Excel file workspace • FILEGDB_WORKSPACE —A file geodatabase workspace • NONE —Used to skip the parameter • OLEDB_WORKSPACE —An OLE database workspace • PCCOVERAGE_WORKSPACE —A PC ARC/INFO Coverage workspace • RASTER_WORKSPACE —A raster workspace • SDE_WORKSPACE —An SDE geodatabase workspace • SHAPEFILE_WORKSPACE —A shapefile workspace • TEXT_WORKSPACE —A text file workspace • TIN_WORKSPACE —A TIN workspace • VPF_WORKSPACE —A VPF workspace	String
dataset_name	A string that represents the name of the table the way it appears in the new workspace (not the name of the table in the table of contents). If **dataset_name** is not provided, the **replaceDataSource** method will attempt to replace the dataset by finding a table with a the same name as the layer's current dataset property.	String
validate	If set to `True`, a workspace will only be updated if the **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 **workspace_path**, regardless of a valid match. In this case, if a match does not exist, then the data source would be broken. (The default value is True)	Boolean

For more detailed discussion, parameter information, scenarios, and code samples, please refer to the Updating and Fixing Data Sources help topic.

Code Sample

TableView example 1

The following script will find a table called `Customers` in a data frame named `Transportation` and will set its definition query.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
for table in arcpy.mapping.ListTableViews(mxd, "", df):
    if table.name.lower() == "trafficaccidents":
        table.definitionQuery = "\"age\" >= 18"
mxd.save()
del mxd
```

TableView example 2

The following script will reference a table in a file geodatabase and then add that table into the referenced map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
accidentsTable =
arcpy.mapping.TableView(r"C:\Project\Data\Transportation.gdb\Accidents
")
arcpy.mapping.AddTableView(df, accidentsTable)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, accidentsTable
```

TextElement (arcpy.mapping)

Top

Summary

The **TextElement** object provides access to properties that enable its repositioning on the page layout as well as modifying the text string and font size.

Discussion

The **TextElement** object represents inserted text within a page layout. This includes items such as inserted text, callouts, rectangle text, titles, and so on. It also includes text strings that have been grouped into a group element. It does not include text strings that are part of a legend or inserted table.

The ListLayoutElements function returns a Python list of page layout element objects. It is necessary to then iterate through each item in the list or specify an index number to reference a specific page element object. To return a list of **TextElements only**, use the TEXT_ELEMENT constant for the **element_type** parameter. A wildcard can also be used to further refine the search based on the element name.

Existing text elements can be cloned and deleted. This capability was initially added to support the creation of dynamic graphic tables. To accomplish this, a map document must be authored with the text elements having the appropriate symbology. If column field names have different text property settings than the cell values, then at least two text elements need to be authored. As the information is read from a table, the text can be cloned using the**clone** method and positioned appropriately using other text element properties. When cloning an element it is very useful to provide a **suffix** value so that cloned elements can be easily identified while using the ListLayoutElements function with a **wildcard** and the same **suffix** value. The returned list of elements can be further modified or deleted using the **delete** method.

To see a complete code sample on how to created a dynamic graphic table based on records in a feature class, see the code sample provided in the GraphicElement class help.

The **TextElement** is also unique from most other page elements in that it has a **text** property. It is with this property that strings can be read and modified. A common example is to perform a search and replace operation on all text elements in a page layout. Similar to the ArcMap user interface, the text string cannot be an empty string; it must include at least an empty space, for example, `textElm.text = " "`, not `textElm.text = ""`.

It is highly recommended that each page layout element be given a unique name so that it can be easily isolated using ArcPy scripting. X and Y element positions are based on the element's anchor position, which is set via the**Size and Position** tab on the element's properties dialog box in ArcMap.

Properties

Property	Explanation	Data Type
angle (Read and Write)	The text baseline angle of the element in degrees.	Double
elementHeight (Read and Write)	The height of the element in page units. The units assigned or reported are in page units.	Double
elementPositionX (Read and Write)	The X location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementPositionY (Read and Write)	The Y location of the data frame element's anchor position. The units assigned or reported are in page units.	Double
elementWidth (Read and Write)	The width of the element in page units. The units assigned or reported are in page units.	Double
fontSize (Read and Write)	The element font size in page units.	Double
name (Read and Write)	The name of the element.	String
text (Read and Write)	The text string associated with the element.	String
type (Read Only)	Returns the element type for any given page layout element. • DATAFRAME_ELEMENT —Dataframe element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Mapsurround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element	String

Method Overview

Method	Explanation
clone ({suffix})	Provides a mechanism to clone an existing graphic text on a page layout.
delete ()	Provides a mechanism to delete an existing text element on a page layout.

Methods

clone ({suffix})

Parameter	Explanation	Data Type
suffix	An optional string that is used to tag each newly created text element. The new element will get the same element name as the parent text element plus the suffix value plus a numeric sequencer. For example, if the parent element name is `FieldLabel`and the **`suffix`** value is `_copy`, the newly cloned elements would be named `FieldLabel_copy`, `FieldLabel_copy_1`,`FieldLabel_copy_2`, and so on. If a **`suffix`** is not provided, then the results would look like `FieldLabel_1`, `FieldLabel_2`,`FieldLabel_3`, and so on.	String

Grouped text elements can't be cloned. All grouped elements are graphic element types. First check to see if a graphic element is grouped by using the **`isGroup`** property on the GraphicElement object.

delete ()

It may be necessary to delete cloned elements. Cloned elements, when created, can be given a custom **`suffix`** so they can be easy to find when using the wildcard parameter on the ListLayoutElements function.

Code Sample

TextElement example 1

The following script will replace all occurrences of the year `2009` with the year `2010`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT"):
    if elm.text == "2009":
        elm.text = "2010"
mxd.save()
del mxd
```

TextElement example 2

The following script will update a static 9.3 map document date/time text element string to now incorporate the new version 10 dynamic text strings. Current time is appended as a new line below the current date. The text string is set using a couple of different Python techniques. All produce the same result.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
elm = arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT",
"DateTimeElm")[0]

#Concatenate the date and time with a new line character (\n) using
single quotes so that double quotes are treated as part of the string.
elm.text = 'Date: <dyn type="date" format="short"/> \nTime: <dyn
type="time" format=""/>'

#Use triple quotes and put the line break within the string.
elm.text = """Date: <dyn type="date" format="short"/>
Time: <dyn type="time" format=""/>"""

mxd.save()
del mxd
```

TextElement example 3

The following script will adjust a string's font size to fit a specified page width. It essentially sets the initial font size to 100 points and reduces the font size until the text string fits the desired width. The text string is set up as dynamic text that represents the map's title information. This script will only run properly if you set the **`title`** property of the map document. This can be found under **File** > **Map Document Properties**.

```
import arcpy
mxd = arcpy.mapping.MapDocument("CURRENT")
elm = arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT",
"MapTitle")[0]
elmWidth = 4.0
x = 100
elm.text = '<dyn type="document" property="title"/>'
elm.fontSize = x
while elm.elementWidth > float(elmWidth):
    elm.fontSize = x
    x = x - 1
arcpy.RefreshActiveView()
del mxd
```

UniqueValuesSymbology (arcpy.mapping)

Top

Summary

The **UniqueValuesSymbology** class provides access to different properties that allow you to change the appearance of a layer's unique value symbology.

Discussion

The **UniqueValuesSymbology** class provides access to a limited number of properties and methods that allow you to automate layer symbology in a map document (`.mxd`) or layer file (`.lyr`). Basic operations such as modifying class values and labels, or changing the field that the symbology is based on are some of the properties that can be modified. For access to the complete set of layer symbology properties and settings, for example, changing individual symbols for individual classes, it is necessary to author these changes in the ArcMap user interface and then save those changes to a layer file. These custom settings can then be applied to existing layers using the UpdateLayer function.

A layer can use any number of symbologies but not all of them can be modified. Not all layers will use the **UniqueValuesSymbology** class, so it is important to first test if the layer is using this symbology class before attempting to make changes to its properties. The **symbologyType** property on the Layer class was designed for this purpose. First test if the **symbolgyType** for the layer is unique values (`if lyr.symbologyType == "UNIQUE_VALUES":`), then create a variable reference to the **UniqueValuesSymbology** class for that layer (`lyrSymbolClass = lyr.symbology`).

The **symbologyType** on the **Layer** object is a read-only property. In other words, you can't change a unique values symbology to a graduated colors or graduated symbols symbology. You can only change the properties for a specific symbology class on a layer. The only way to change the symbology type is by publishing the desired result to a layer file and using the UpdateLayer function.

The **valueField** is used to change the field the unique values are based on. After setting the new field it makes sense to immediately call the **addAllValues** method so that an updated list of classes is automatically generated. Similar to the user interface, calling **addAllValues** will automatically sort the classes in accending order. If you want to change the order, then use the **classValues** and **classLabels** properties.

The **addAllValues** method is also useful for situations where data is continuously being updated. Layers symbolized using unique values in a map document don't dynamically update when data is changed.

The**addAllValues** method provides a mechanism to automate this process.

If you are making these symbology modifications via the Python window and you are referencing a map document using `CURRENT`, you may not immediately see the changes in the ArcMap application. To refresh the map document, use the RefreshActiveView and RefreshTOC functions.

Properties

Property	Explanation	Data Type
classDescriptions (Read and Write)	A list of strings or numbers that represent the descriptions for each unique value that can optionally appear in a map document's legend. These values are only accessible in the ArcMap user interface by right-clicking a symbol displayed within the **Symbology** tab in the **Layer Properties** dialog box and selecting **Edit Description**. The **classDescriptions** list needs to have the same number of elements and arranged in the same order as the **classValues** property.	List
classLabels (Read and Write)	A list of strings or numbers that represent the labels for each unique value that appears in the map document's table of contents and/or legend items. The **classDescriptions** list needs to have the same number of elements and built in the same order as the **classValues** property. If this property is not set, its values will be the same as **classValues**.	List
classValues (Read and Write)	A list of strings or numbers that represent the class breaks. Changing this property will automatically adjust other unique value symbology properties based on the new information. It is good practice to always set this value before setting**classDescriptions** and **classLabels**.	List
showOtherValues (Read and Write)	Setting this value to `True` will display a symbol for all values that don't match the current list of **classValues**.	Boolean
valueField (Read and Write)	A string that represents a valid dataset field name used for the layer's unique value symbology. Changing this value will automatically adjust other symbology properties based on the new information.	String

Method Overview

Method	Explanation
addAllValues ()	Adds all unique values to the symbology.

Methods

addAllValues ()

The **addAllValues** method updates a layer's symbology so that all values are displayed in the unique value symbology. This method is useful when unique values are added to a layer after the symbology has been authored in ArcMap. Calling the **addAllValues** method will insert those new values into the list of existing values.

Code Sample

UniqueValuesSymbology example 1

The following script will change the **valueField** that the unique value symbology is based on. Next, the **addAllValues** is used to update the class list.

```
import arcpy
mxd = arcpy.mapping.MapDocument("current")
lyr = arcpy.mapping.ListLayers(mxd, "Population")[0]
if lyr.symbologyType == "UNIQUE_VALUES":
  lyr.symbology.valueField = "SUB_REGION"
  lyr.symbology.addAllValues()
arcpy.RefreshActiveView()
arcpy.RefreshTOC()
del mxd
```

UniqueValuesSymbology example 2

The following script will update a layer's unique value symbology class list based on the results of an attribute query.

```
import arcpy
mxd = arcpy.mapping.MapDocument("current")
lyr = arcpy.mapping.ListLayers(mxd, "Population")[0]
arcpy.SelectLayerByAttribute_management(lyr, "NEW_SELECTION",
"\"POP2000\" > 20000000")
stateList = []
rows = arcpy.da.SearchCursor(lyr, ["STATE_NAME"])
for row in rows:
  stateList.append(row[0])

if lyr.symbologyType == "UNIQUE_VALUES":
  lyr.symbology.classValues = stateList
  lyr.symbology.showOtherValues = False

arcpy.RefreshActiveView()
arcpy.RefreshTOC()
del mxd
```

ArcPy.Mapping Functions

-with required and {optional} parameters

AddLayer (data_frame, add_layer, {add_position})

AddLayerToGroup (data_frame, target_group_layer, add_layer, {add_position})

AddTableView (data_frame, add_table)

AnalyzeForMSD (map_document)

AnalyzeForSD (sddraft)

ConvertToMSD (map_document, out_msd, {data_frame}, {msd_anti_aliasing}, {msd_text_anti_aliasing})

ConvertWebMapToMapDocument (webmap_json, {template_mxd}, {notes_gdb}, {extra_conversion_options})

CreateGISServerConnectionFile (connection_type, out_folder_path, out_name, server_url, server_type, {use_arcgis_desktop_staging_folder}, {staging_folder_path}, {username}, {password}, {save_username_password})

CreateMapSDDraft (map_document, out_sddraft, service_name, {server_type}, {connection_file_path}, {copy_data_to_server}, {folder_name}, {summary}, {tags})

DeleteMapService (connection_url_or_name, server, service_name, {folder_name}, {connection_username}, {connection_password}, {connection_domain})

ExportReport (report_source, report_layout_file, output_file, {dataset_option}, {report_title}, {starting_page_number}, {page_range}, {report_definition_query}, {extent}, {field_map})

ExportToAI (map_document, out_ai, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {picture_symbol}, {convert_markers})

ExportToBMP (map_document, out_bmp, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {rle_compression})

ExportToEMF (map_document, out_emf, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {description}, {picture_symbol}, {convert_markers})

ExportToEPS (map_document, out_eps, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {ps_lang_level}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {jpeg_compression_quality})

ExportToGIF (map_document, out_gif, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {gif_compression}, {background_color}, {transparent_color}, {interlaced})

ExportToJPEG (map_document, out_jpeg, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {jpeg_quality}, {progressive})

ExportToPDF (map_document, out_pdf, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {compress_vectors}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality})

ExportToPNG (map_document, out_png, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {background_color}, {transparent_color}, {interlaced})

ExportToSVG (map_document, out_svg, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {compress_document}, {picture_symbol}, {convert_markers}, {embed_fonts})

ExportToTIFF (map_document, out_tiff, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {tiff_compression}, {geoTIFF_tags})

InsertLayer (data_frame, reference_layer, insert_layer, {insert_position})

Layer (lyr_file_path)

ListBookmarks (map_document, {wildcard}, {data_frame})

ListBrokenDataSources (map_document_or_layer)

ListDataFrames (map_document, {wildcard})

ListLayers (map_document_or_layer, {wildcard}, {data_frame})

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

ListMapServices (connection_url_or_name, server, {connection_username}, {connection_password}, {connection_domain})

ListPrinterNames ()

ListStyleItems (style_file_path, style_folder_name, {wildcard})

ListTableViews (map_document, {wildcard}, {data_frame})

MapDocument (mxd_path)

MoveLayer (data_frame, reference_layer, move_layer, {insert_position})

PDFDocumentCreate (pdf_path)

PDFDocumentOpen (pdf_path, {user_password}, {master_password})

PrintMap (map_document, {printer_name}, {data_frame}, {out_print_file}, {image_quality})

PublishMSDToServer (msd_path, connection_url_or_name, server, service_name, {folder_name}, {service_capabilities}, {connection_username}, {connection_password}, {connection_domain})

RemoveLayer (data_frame, remove_layer)

RemoveTableView (data_frame, remove_table)

TableView (table_view_data_source)

UpdateLayer (data_frame, update_layer, source_layer, {symbology_only})

UpdateLayerTime (data_frame, update_layer, source_layer)

AddLayer (arcpy.mapping)

Top

Summary

Provides the ability to add a layer to a data frame within a map document (`.mxd`) using simple placement options.

Discussion

`AddLayer` is an easy way to add a layer or group layer into a map document. It can add a layer with auto-arrange logic that places the new layer in a data frame similarly to how the **Add Data** button works in ArcMap; it places the layer based on layer weight rules and geometry type. The other placement choices are either at the top or the bottom of a data frame. For more precise layer placement options, refer to the InsertLayer function.
`AddLayer` is the only function that can add a layer into an empty data frame. **`AddLayer`** does not allow you to add layers to a layer file or within a group layer. If interested in managing layer files, refer to the Layer and AddLayerToGroup functions.

The layer that is added must reference an already existing layer (keep in mind that a layer can be a group layer as well). The source can either come from a layer file on disk, from within the same map document and data frame, the same map document but different data frame, or even from a completely separate map document.

The way a layer appears in the table of contents (TOC) after it is added depends on the source layer and how it appears. For example, some layers are completely collapsed and do not display their symbol(s) in the TOC. This setting is built into the layer. If a layer is collasped, saved to a layer file, and then added to a map document, the layer will be collasped in the new map document when added via **`AddLayer`**.

Syntax

AddLayer (data_frame, add_layer, {add_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object within a map document.	DataFrame
add_layer	A reference to a Layer object representing the layer to be added. This reference can point to a layer file on disk or a layer within a map document.	Layer
add_position	A constant that determines the placement of the added layer within a data frame. • AUTO_ARRANGE —Automatically places the layer similar to how the **Add Data** button works in ArcMap • BOTTOM —Places the layer at the bottom of the data frame • TOP —Places the layer at the top of the data frame (The default value is AUTO_ARRANGE)	String

Code Sample

AddLayer example 1:

The following script will add a new layer from a layer (`.lyr`) file on disk and place it at the bottom of the data frame called `New Data Frame`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
addLayer = arcpy.mapping.Layer(r"C:\Project\Data\Orthophoto.lyr")
arcpy.mapping.AddLayer(df, addLayer, "BOTTOM")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, addLayer
```

AddLayer example 2:

The following script will add a new group layer called `NE_Layers` from another, independent map document and use auto-arrange to place the group layer within a data frame called `New Data Frame`.

```
import arcpy

#Reference layer in secondary map document
mxd2 = arcpy.mapping.MapDocument(r"C:\Project\ProjectTemplate.mxd")
df2 = arcpy.mapping.ListDataFrames(mxd2, "Layers")[0]
addLayer = arcpy.mapping.ListLayers(mxd2, "NE_Layers", df2)[0]

#Add layer into primary map document
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
arcpy.mapping.AddLayer(df, addLayer, "AUTO_ARRANGE")

#Save to a new map document and clear variable references
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, mxd2
```

AddLayerToGroup (arcpy.mapping)

Top

Summary

Provides the ability to add a layer to a group layer within a map document (`.mxd`) using simple placement options.

Discussion

`AddLayerToGroup` is an easy way to add a layer into an already existing group layer. It can add a layer with auto-arrange logic that places the new layer in a group layer similarly to how the **Add Data** button works in ArcMap; it places the layer based on layer weight rules and geometry type. The other placement choices are either at the top or the bottom of a group layer.
`AddLayerToGroup` is the only function that can add a layer into an empty group layer. **`AddLayerToGroup`** does not allow you to add layers to group layers within a layer file. Layer files should be managed and authored using ArcMap.
The target group layer must be a group layer. The **`isGroupLayer`** property on the Layer object is a way of confirming if the returned Layer object is truely a group layer.

The layer that is added must reference an already existing layer (keep in mind that a layer can be a group layer as well). The source can either come from a layer file on disk, from within the same map document and data frame, the same map document but different data frame, or even from a completely separate map document.

The way a layer appears in the table of contents (TOC) after it is added depends on the source layer and how it appears. For example, some layers are completely collapsed and do not display their symbol(s) in the TOC. This setting is built into the layer. If a layer is collasped, saved to a layer file, and then added to a map document, the layer will be collasped in the new map document when added via **`AddLayerToGroup`**.

Syntax

AddLayerToGroup (data_frame, target_group_layer, add_layer, {add_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the group layer to which the new layer will be added.	DataFrame
target_group_layer	A Layer object representing the group layer to which the new layer will be added. It must be a group layer.	Layer
add_layer	A reference to a Layer object representing the layer to be added. This reference can point to a layer file on disk or a layer in a map document.	Layer
add_position	A constant that determines the placement of the added layer within a data frame. • AUTO_ARRANGE —Automatically places the layer similar to how the **Add Data** button works in ArcMap • BOTTOM —Places the layer at the bottom of the data frame • TOP —Places the layer at the top of the data frame (The default value is AUTO_ARRANGE)	String

Code Sample

AddLayerToGroup example:

The following script will add a new layer from a layer (`.lyr`) file on disk and place it at the bottom of a group layer called `24000 Scale Data` in a data frame called `County Maps`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
targetGroupLayer = arcpy.mapping.ListLayers(mxd, "24000 Scale Data",
df)[0]
addLayer =
arcpy.mapping.Layer(r"C:\Project\Data\StreetsWithLabels.lyr")
arcpy.mapping.AddLayerToGroup(df, targetGroupLayer, addLayer,
"BOTTOM")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, addLayer
```

AddTableView (arcpy.mapping)

Top

Summary

Provides the ability to add a table to a data frame within a map document (`.mxd`).

Discussion

`AddTableView` provides a way to add a table into a map document. A reference to a **`TableView`** object must exist first. It can be a reference to a table in another map document by using the ListTableViews function, or it can be a reference to a table in a workspace by using the TableView function.

Syntax

AddTableView (data_frame, add_table)

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object within a map document.	DataFrame
add_table	A reference to a TableView object representing the table to be added. This reference can point to a table within an existing map document or it can reference a table in a workspace via the TableView function.	TableView

Code Sample

AddTableView example

The following script will add three tables from three different workspaces to a single data frame in a map document. The different workspaces are shapefile/dBASE, file geodatabase, and SDE.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
dbf_Table = arcpy.mapping.TableView(r"C:\Project\Data\customers.dbf")
fGDB_Table =
arcpy.mapping.TableView(r"C:\Project\Data\fGBD.gdb\customers")
SDE_Table =
arcpy.mapping.TableView(r"C:\PathToSDEConnectionfile.sde\customers")
arcpy.mapping.AddTableView(df, dbf_Table)
arcpy.mapping.AddTableView(df, fGDB_Table)
arcpy.mapping.AddTableView(df, SDE_Table)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

AnalyzeForMSD (arcpy.mapping)

Top

Summary

Legacy:

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. Please use the AnalyzeForSD function instead.

Analyzes map documents (`.mxd`) to determine sources for potential suitability and performance issues when converting a map to a map service definition (`.msd`) file.

Discussion

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. See the following help topics for more information: What to expect when migrating to ArcGIS 10.2 for Server and Migration to ArcGIS 10.2 for Server.

Consider using the AnalyzeForSD function instead.

Syntax

AnalyzeForMSD (map_document)

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument

AnalyzeForSD (arcpy.mapping)

Top

Summary

Analyzes Service Definition Draft (`.sddraft`) files to determine suitability and sources of potential performance issues before converting a Service Definition Draft file to a Service Definition (`.sd`) file.

Discussion

An important task that you can perform is analyzing your Service Definition Draft file to identify errors and other potential issues that you may need to address before you can create a Service Definition file. **`AnalyzeForSD`** can be used with Service Definition Drafts for map, geoprocessing and image services. This function returns a Python dictionary containing errors, warnings and messages. For example, when working with Service Definition Drafts for map services, this function can:

- Help you identify layer and symbology types that are not supported for optimized map drawing performance.
- Show warnings for issues that can potentially slow down display performance.
- List other information messages about your map document that can help you optimize its performance as a map service.

Analysis for a map Service Definition Draft file is based on many factors including the data used by the map; map, layer, and symbol properties; set service properties such as chosen capabilities and associated properties the service will support; and the server type hosting the service. You will see three types of messages in the Python dictionary: errors, warnings, and information.

Errors for a map Service Definition Draft file typically refer to your map's use of map layer types or display options that are not supported for map services. All errors must be repaired before you can create a Service Definition file. Warnings and other messages identify issues that may affect drawing performance and appearance. Warnings alert you to issues in which drawing performance or drawing appearance may be affected, but these issues do not preclude you from converting the Service Definition Draft file to a Service Definition file.

After analyzing the Service Definition Draft using **`AnalyzeForSD`**, it can then be converted to a fully consolidated Service Definition (`.sd`) file using the Stage Service geoprocessing tool. Staging compiles all the necessary information needed to successfully publish the GIS resource. If you have chosen to copy data to the server, the data will be added when the Service Definition Draft is staged. Finally, the Service Definition file can be uploaded and published as a GIS service to a specified GIS server using the Upload Service Definition geoprocessing tool.

This step takes the Service Definition file, copies it onto the server, extracts required information and publishes the GIS resource. For more information, see the overview of the Publishing toolset.

The functions to create Service Definition Drafts for map, geoprocessing, image and geocoding services are:

CreateMapSDDraftCreateGPSDDraftCreateImageSDDraftCreateGeocodeSDDraft

Syntax

AnalyzeForSD (sddraft)

Parameter	Explanation	Data Type
sddraft	A string that represents the path and file name for the Service Definition Draft (`.sddraft`) file.	String

Return Value

Data Type	Explanation
Dictionary	Returns a Python Dictionary of information messages, warnings, and errors.

Code Sample

AnalyzeForSD example 1

The following script analyzes a Service Definition Draft (`.sddraft`) file to identify potential performance bottlenecks and map errors that you may need to address before you can create a Service Definition (`.sd`) file.

```
import arcpy

analysis = arcpy.mapping.AnalyzeForSD(r"C:\Project\Counties.sddraft")

for key in ('messages', 'warnings', 'errors'):
  print "----" + key.upper() + "---"
  vars = analysis[key]
  for ((message, code), layerlist) in vars.iteritems():
    print "    ", message, " (CODE %i)" % code
    print "       applies to:",
    for layer in layerlist:
        print layer.name,
    print
```

AnalyzeForSD example 2

The following script demonstrates the complete publishing of map services using arcpy.mapping workflow. Automating the publishing of map services can be accomplished by using a combination of **`arcpy.mapping`** functions and the geoprocessing tools in the Publishing toolset. The workflow begins with a map document that you want to publish. First, use the **`arcpy.mapping`** function **`CreateMapSDDraft`** to create a service definition draft. Next, you should analyze the service definition draft for issues that could prevent you from publishing successfully by using the **`AnalyzeForSD`** function. After analyzing the service definition draft and addressing serious issues, it is time to stage the service definition. Staging takes the service definition draft and consolidates all the information needed to publish the service into a complete service definition. Use the Stage_Service geoprocessing tool to stage the service definition. Finally, use the Upload_Service_Definition geoprocessing tool to upload the service definition to the server and publish the map service.

```
import arcpy

# define local variables
wrkspc = 'C:/Project/'
mapDoc = arcpy.mapping.MapDocument(wrkspc + 'counties.mxd')
con = r'GIS Servers\arcgis on MyServer_6080 (admin).ags'
service = 'Counties'
sddraft = wrkspc + service + '.sddraft'
sd = wrkspc + service + '.sd'

# create service definition draft
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER', con, True, None)
# analyze the service definition draft
analysis = arcpy.mapping.AnalyzeForSD(sddraft)

# stage and upload the service if the sddraft analysis did not contain
errors
if analysis['errors'] == {}:
    # Execute StageService
    arcpy.StageService_server(sddraft, sd)
    # Execute UploadServiceDefinition
    arcpy.UploadServiceDefinition_server(sd, con)
else:
    # if the sddraft analysis contained errors, display them
    print analysis['errors']
```

ConvertToMSD (arcpy.mapping)

Top

Summary

Legacy:

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. Please use the CreateMapSDDraft function instead.
Converts a map to a map service definition (`.msd`) file.

Discussion

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. See the following help topics for more information: What to expect when migrating to ArcGIS 10.2 for Server and Migration to ArcGIS 10.2 for Server.
Consider using the CreateMapSDDraft function instead.

Syntax

ConvertToMSD (map_document, out_msd, {data_frame}, {msd_anti_aliasing}, {msd_text_anti_aliasing})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_msd	A string that represents the path and file name for the output MSD file.	String
data_frame	A variable that references a DataFrame object. If more than one data frame exists, it is important to specify which data frame will be the active data frame for the published service. The default value is the active data frame. (The default value is USE_ACTIVE_VIEW)	DataFrame
msd_anti_aliasing	A string that sets antialiasing for nontext items within the saved MSD. • NONE — No antialiasing is performed. • FASTEST — Minimal antialiasing is performed, optimized for speed. • FAST — Some antialiasing is performed, optimized for speed with better quality than FASTEST. • NORMAL — A good balance of speed and quality. • BEST — The best quality antialiasing. This option takes the longest to render. (The default value is NONE)	String
msd_text_anti_aliasing	A string that sets antialiasing for text items within the saved MSD. • FORCE — Text is always drawn with antialiasing, regardless of the individual font's parameters. • NORMAL — Antialiasing is performed as determined by the font. Each individual font has parameters created within it by the font author that defines which sizes the font should draw with antialiasing. • NONE — No antialiasing is performed. (The default value is NONE)	

ConvertWebMapToMapDocument (arcpy.mapping)

Top

Summary

Converts a web map (in JSON format) that you intend to print or export to a map document. The map document can be further modified before finally being printed or exported.

Discussion

The **`ConvertWebMapToMapDocument`** function will convert a web map that you intend to print or export to a map document. Once the document is converted, the full state of the web map exists in the map document. The map document can then be further modified before finally being printed or exported to a common format such as PDF. **`ConvertWebMapToMapDocument`** would most commonly be used when printing a map from a web application using the ArcGIS Web APIs. The **`ConvertWebMapToMapDocument`** function is primarily intended for workflows where the web map needs to be modified or exported using arcpy.mapping functions. Some example workflows that can be met using the**`ConvertWebMapToMapDocument`** function are as follows:

- **Swapping out service layers for local vector data**—In arcpy.mapping scripts, service layers can be identified and swapped out for layers that point to local data. This is commonly desired when a vector output is wanted instead of service layers. For example, vector PDF output supports the following in PDF viewing applications: toggling layer visibility, viewing of feature attributes, and viewing of map coordinates. One way to accomplish swapping service layers for vector data would be to stage template map documents that contain vector equivalents of all the possible service layers. After executing the **`ConvertWebMapToMapDocument`** function, loop through all the layers in the output map document, removing all layers except the vector layers that correspond to the service layers in the web map. This workflow would be used when swapping your own services with the corresponding vector data that you already have.
- **Creating map books**—A map series can be generated if Data Driven Pages is enabled on the **`template_mxd`**. Moreover, the output map document can be exported as a PDF and inserted into other PDFs using the PDFDocument class to create a complete map book.
- **Exporting using advanced options**—All of the arcpy.mapping export functions have many advanced options. For example, the ExportToPDF function has parameters for controlling raster and vector compression, defining colorspace, embedding fonts, and so on.
- **Controlling the appearance of the legend**—The arcpy.mapping module provides the ability to remove legend items or modify their style items with custom settings using the LegendElement class and ListStyleItemsfunction.

Note:

ArcGIS for Server also includes a geoprocessing service named PrintingTools. The PrintingTools service can be used in a web application to generate a high-cartographic-quality printable image. For more information regarding the PrintingTools service, see the following:

Printing in web applications

Once you have a Python script that prepares the map for printing, you can encapsulate it in a geoprocessing script tool. You can then publish the script tool as an ArcGIS Server geoprocessing service. Each ArcGIS Web API has a **Print Task** that you can use in your web applications. The **Print Task** has a URL property that will point to the REST URL of the geoprocessing service you created. For more information on using the ArcGIS Web APIs, consult the help for your preferred developer platform at http://resources.arcgis.com/content/web/web-apis. For more information on geoprocessing services, see the following:

What is a geoprocessing service?A quick tour of authoring and sharing geoprocessing servicesA quick tour of publishing a geoprocessing serviceEssential vocabulary for geoprocessing services

When using **`ConvertWebMapToMapDocument`** in a geoprocessing service in the ArcGIS Web APIs, the parameter names of the script tool must match the ArcGIS Web API **Print Task** parameters:

Parameter Name	Data Type	Explanation
Web_Map_as_JSON	String	A JSON representation of the state of the map to be exported as it appears in the web application. The ArcGIS Web APIs (JavaScript, Flex, and Silverlight) allow developers to easily get this JSON string from the web application.
Output_File	File	Output file name. The extension of the file depends on the **`Format`** parameter.
Format	String	The format in which the map image for printing will be delivered. The following strings are accepted: • `PNG8` (default if the parameter is left blank) • `PNG` • `PDF` • `PNG32` • `JPG` • `GIF` • `EPS` • `SVG` • `SVGZ`
Layout_Template	String	Either a name of a template from the list or the keyword MAP_ONLY. When MAP_ONLY is chosen or an empty string is passed in, the output map does not contain any page layout surroundings (for example, title, legends, and scale bar).

Tip:

Any number of additional user-defined parameters can also be added. The ability to pass extra parameters into a custom print task is useful, as it allows you to collect any number of extra parameters from the web application and pass them into the Python script.

`ConvertWebMapToMapDocument` would most commonly be used when printing a map from a web application. When you use the ArcGIS Web API **Print Task**, you don't need to create the web map JSON; the APIs take care of it for you. However, before the script can be published and used in the web APIs, it has to be run locally. Any valid web map JSON string can be used when running the script locally. A JSON string similar to what your web application will be returning may be required for your script to run successfully. See ExportWebMap specification to understand how this text should be formatted. A sample string is below:

```
{
    "layoutOptions": {
        "titleText": "Simple WebMap JSON example"
    },
    "operationalLayers": [
        {
            "url":
"http://maps1.arcgisonline.com/ArcGIS/rest/services/USA_Federal_Lands/
MapServer",
            "visibility": true
        }
    ],
    "exportOptions": {
        "outputSize": [
            1500,
            1500
        ]
    },
    "mapOptions": {
        "extent": {
            "xmin": -13077000,
            "ymin": 4031000,
            "xmax": -13023000,
            "ymax": 4053000
        }
    },
    "version": "1.4"
}
```

When running the script tool, the JSON string can be copied and pasted into the `Web_Map_as_JSON` input parameter. However, the line breaks must be removed for the string to be valid input. A sample JSON string with line breaks removed is below:

```
{"layoutOptions": {"titleText": "Simple WebMap JSON
example"},"operationalLayers": [{"url":
"http://maps1.arcgisonline.com/ArcGIS/rest/services/USA_Federal_Lands/
MapServer","visibility": true}],"exportOptions": {"outputSize":
[1500,1500]},"mapOptions": {"extent": {"xmin": -13077000,"ymin":
4031000,"xmax": -13023000,"ymax": 4053000}},"version": "1.4"}
```

Tip:

For publishing purposes, you can leave the `Web_Map_as_JSON` input parameter blank, as the ArcGIS Web APIs will provide the web map JSON in the web application. You can leave the `Web_Map_as_JSON` input parameter blank provided the Python script was written in such a way as to not fail with blank input. For example, the script doesn't look for web map layers by name. If the script fails on the server due to an error in the script, it will have to be fixed locally and republished to the server. Therefore, it is good practice to ensure the script works locally before publishing by testing with a valid web map JSON string or using a debug map document that contains all the elements that would be in the web map JSON.

Tip:

As mentioned before, the JSON string returned from the ArcGIS Web APIs contains the full state of the web map. The `layoutOptions` object in the JSON string warrants extra discussion, as it will automatically update layout elements that can be staged in the `template_mxd`. For example, if the JSON has a `titleText`setting, and the `template_mxd` has a Title dynamic text element, the title in the template map document layout will be updated with the `titleText` value. For more information, see the `layoutOptions` section in ExportWebMap specification.

When you encapsulate the Python script that uses **`ConvertWebMapToMapDocument`** in a geoprocessing service, you need to make sure that ArcGIS for Server can see the template map documents and data used in the web application. It is best practice to use a folder that is registered with ArcGIS for Server. For more information on registering data, see the following:

Making your data accessible to ArcGIS ServerAbout registering your data with ArcGIS ServerRegistering your data with ArcGIS Server using ArcGIS for Desktop

When authoring template map documents in the registered folder, it is best practice to use relative paths. That way, ArcGIS for Server will be able to find the data in the registered folder relative to the location of the map documents. See Referencing data in the map for more information.

Caution:

When using **`ConvertWebMapToMapDocument`**, relative paths to layers must be pointing to data at, below, or one level above the path of the **`template_mxd`**.

Caution:

If you choose to not use registered folders, template map documents and data will be packaged and copied to the server. During packaging, data may be moved and re-sourced with relative paths to a folder structure that **`ConvertWebMapToMapDocument`** cannot resolve. For this reason, usage of registered folders is recommended.

In addition to creating your own template map documents, you can also use the preauthored templates that ship with the software. They are located at`<installation_directory>\Templates\ExportWebMapTemplates`. These templates contain map elements such as a legend, current date dynamic text, a scale bar, and scale text.

By default, notes overlays or client-side graphics from the web application will be stored in an in-memory workspace. The in-memory workspace is temporary and will be deleted when the application is closed. To make a permanent copy of the output map document that contains notes overlays, specify a **`notes_gdb`**; then use the Consolidate Map or Package Map geoprocessing tools to make a copy of the output map document. The only time you would need to specify a **`notes_gdb`** is if you plan on making a permanent copy of the output map document. See code example 2 below. If the web map does not contain notes overlays, you can use the **`saveACopy`** method from the MapDocument class to make a permanent copy of the output map document.

The arcpy.mapping module also provides methods to control the appearance of the legend in the output map document's layout. See the **removeItem** method on the LegendElement class to remove legend items. The advanced tutorial listed below demonstrates this workflow. See the **updateItem** method on the LegendElement class and the ListStyleItems method to update legend items. Be sure to read the following tutorials. They demonstrate the entire vector printing and exporting with **ConvertWebMapToMapDocument** workflow: authoring the staged template map documents, authoring the Python script, creating the geoprocessing service, and creating the web application.

- Tutorial: Basic web map printing and exporting using arcpy.mapping
- Tutorial: Advanced web map printing and exporting using arcpy.mapping

Syntax

ConvertWebMapToMapDocument (webmap_json, {template_mxd}, {notes_gdb}, {extra_conversion_options})

Parameter	Explanation	Data Type
webmap_json	The web map for printing in JavaScript Object Notation (JSON). See the ExportWebMap JSON specification for more information. The ArcGIS Web APIs (JavaScript, Flex, and Silverlight) allow developers to easily get this JSON string from the web application.	String
template_mxd	A string representing the path and file name to a map document (.mxd) to use as the template for the page layout. The contents of the web map will be inserted into the data frame that was active at the time the **template_mxd** was saved. Layers in the**template_mxd** file's active data frame (and all other data frames) will be preserved in the output **mapDocument**. (The default value is None)	String
notes_gdb	A string representing the path to a new or existing file geodatabase or an existing enterprise geodatabase connection where graphic features should be written. This parameter should only be used if graphic features from the web map JSON need to be preserved permanently. In most cases, this parameter is not required, as a temporary in-memory workspace will be used to store graphic features. This parameter allows you to save graphic features to persistent storage, which is essential if you plan to use the map document for operations that require saving or loading from disk (for example, packaging or consolidating). The path must end with a .gdb or .sde extension. (The default value is None)	String
extra_conversion_options	A dictionary of credentials for secured services. This parameter is required if the services in the web map JSON require a user name and password to view. Keys accepted in the dictionary are as follows: • SERVER_CONNECTION_FILE • WMS_CONNECTION_FILE • WMTS_CONNECTION_FILE An example of key value pairs is as follows: `credentials = {"SERVER_CONNECTION_FILE":r"Z:\ArcGIS2on MyServer (user).ags", WMS_CONNECTION_FILE":r"Z:\USA on MyServer.wms"} result = arcpy.mapping.ConvertWebMapToMapDocument(json, extra_conversion_options=credentials)` (The default value is None)	Dictionary

Return Value

Data Type	Explanation
tuple	Returns a Python named Tuple of web map and request properties: • mapDocument —The map document object created as output of the function. • DPI —The requested DPI of the export from the web app. • outputSizeHeight —The height of the image as specified from the web app. For use when performing a data view export. • outputSizeWidth —The width of the image as specified from the web app. For use when performing a data view export. **Caution:** **ConvertWebMapToMapDocument** will create temporary map documents (.mxd files) in the system temp folder. On Windows Vista and 7, this is located at C:\Users\<user name>\AppData\Local\Temp. It is the user's responsibility to manage the map documents in this folder. To delete these temporary map documents in scripts, see the clean upsection in the code samples below.

Code Sample

ConvertWebMapToMapDocument example 1

In this example, the script reads in a web map JSON, a template map document, and existing PDF documents to which the web map will be appended. The output map document from the **ConvertWebMapToMapDocument**function is exported as a PDF and inserted into other PDF files using the PDFDocument class.

```
import arcpy
import os
import uuid

# The template location in the server data store
templatePath = '//MyMachine/MyDataStore/WebMap'

# Input WebMap json
Web_Map_as_JSON = arcpy.GetParameterAsText(0)

# Input Layout template
Layout_Template = arcpy.GetParameterAsText(1)
if Layout_Template == '#' or not Layout_Template:
    Layout_Template = "Landscape11x17"

# PDF Title Page
PDF_Title = arcpy.GetParameterAsText(2)
if PDF_Title == '#' or not PDF_Title:
    PDF_Title = "TitlePage.PDF"

# PDF End Page
PDF_End = arcpy.GetParameterAsText(3)
if PDF_End == '#' or not PDF_End:
    PDF_End = "ContactInfo.PDF"

# Get the requested map document
templateMxd = os.path.join(templatePath, Layout_Template + '.mxd')

# Get the requested PDF files
PDF_Title_File = os.path.join(templatePath, PDF_Title)
PDF_End_File = os.path.join(templatePath, PDF_End)

result = arcpy.mapping.ConvertWebMapToMapDocument(Web_Map_as_JSON,
                                                  templateMxd)
mxd = result.mapDocument

# Use the uuid module to generate a GUID as part of the output name
# This will ensure a unique output name
WebMapPDF = os.path.join(arcpy.env.scratchFolder,
                         'WebMap_{}.pdf'.format(str(uuid.uuid1())))

# Export the WebMap to PDF
arcpy.mapping.ExportToPDF(mxd, WebMapPDF)

# Create a new "master" output PDF Document
# Append Title, WebMap and End PDFs to it
Output_Name = os.path.join(arcpy.env.scratchFolder,

'OutputWithWebMap_{}.pdf'.format(str(uuid.uuid1())))
pdfDoc = arcpy.mapping.PDFDocumentCreate(Output_Name)
pdfDoc.appendPages(PDF_Title_File)
pdfDoc.appendPages(WebMapPDF)
pdfDoc.appendPages(PDF_End_File)
pdfDoc.saveAndClose()

# Set the output parameter to be the output PDF
arcpy.SetParameterAsText(4, Output_Name)

# Clean up - delete the map document reference
filePath = mxd.filePath
del mxd, result
os.remove(filePath)
```

ConvertWebMapToMapDocument example 2

In this example, the ConsolidateMap geoprocessing tool is used to make a permanent copy of the output map document, including the notes overlays.

```
import arcpy
import os
import uuid

# The template location in the registered folder
templatePath = '//MyMachine/Austin/WebMap'

# Input web map json
Web_Map_as_JSON = arcpy.GetParameterAsText(0)

# Format for output
Format = arcpy.GetParameterAsText(1)
if Format == '#' or not Format:
    Format = "PDF"

# Input Layout template
Layout_Template = arcpy.GetParameterAsText(2)
if Layout_Template == '#' or not Layout_Template:
    Layout_Template = "Landscape11x17"

# Get the requested map document
templateMxd = os.path.join(templatePath, Layout_Template + '.mxd')

# Since we are making a permanent copy of the notes overlays,
# we need to specify a notes geodatabase
notes = os.path.join(arcpy.env.scratchFolder, 'mynotes.gdb')

# Convert the web map to a map document
result = arcpy.mapping.ConvertWebMapToMapDocument(Web_Map_as_JSON,
templateMxd, notes)
mxd = result.mapDocument

# Save the web map and notes overlays to a new map document using
ConsolidateMap
arcpy.ConsolidateMap_management(mxd.filePath,
                                os.path.join(arcpy.env.scratchFolder,


# Clean up - delete the map document reference
filePath = mxd.filePath
del mxd, result
os.remove(filePath)
```

ConvertWebMapToMapDocument example 3

In this example, a staged template map document is used that contains vector equivalents of all the possible service layers. After executing the **`ConvertWebMapToMapDocument`** function, the script loops through all the layers in the output map document, removing all layers except the vector layers that correspond to the service layers in the web map JSON. The output map document layout is then exported to either PDF or PNG.

```
import arcpy
import os
import uuid

# The template location in the registered folder
templatePath = '//MyComputerName/MyDataStore/USA'

# Input web map json
Web_Map_as_JSON = arcpy.GetParameterAsText(0)

# Format for output
Format = arcpy.GetParameterAsText(1)
if Format == '#' or not Format:
    Format = "PDF"

# Input Layout template
Layout_Template = arcpy.GetParameterAsText(2)
if Layout_Template == '#' or not Layout_Template:
    Layout_Template = "NorthwesternUSA"

# Get the requested map document
templateMxd = os.path.join(templatePath, Layout_Template + '.mxd')

# Convert the web map to a map document
result = arcpy.mapping.ConvertWebMapToMapDocument(Web_Map_as_JSON,
templateMxd)
mxd = result.mapDocument

# Reference the data frame that contains the web map
# Note: ConvertWebMapToMapDocument renames the active dataframe in the
template_mxd to "Webmap"
df = arcpy.mapping.ListDataFrames(mxd, 'Webmap')[0]

# Get a list of all service layer names in the map
serviceLayersNames = [slyr.name for slyr in
arcpy.mapping.ListLayers(mxd, data_frame=df)
                      if slyr.isServiceLayer and slyr.visible and not
slyr.isGroupLayer]

# Create a list of all possible vector layer names in the map that
could have a
# corresponding service layer
vectorLayersNames = [vlyr.name for vlyr in
arcpy.mapping.ListLayers(mxd, data_frame=df)
                     if not vlyr.isServiceLayer and not
vlyr.isGroupLayer]

# Get a list of all vector layers that don't have a corresponding
service layer
removeLayerNameList = [vlyrName for vlyrName in vectorLayersNames
                       if vlyrName not in serviceLayersNames]

# Remove all vector layers that don't have a corresponding service
layer
for lyr in arcpy.mapping.ListLayers(mxd, data_frame=df):
    if not lyr.isGroupLayer \
    and not lyr.isServiceLayer \
    and lyr.name in removeLayerNameList \
    and lyr.name in vectorLayersNames:
        arcpy.mapping.RemoveLayer(df, lyr)

# Remove all service layers
# This will leave only vector layers that had corresponding service
layers
for slyr in arcpy.mapping.ListLayers(mxd, data_frame=df):
    if slyr.isServiceLayer:
        arcpy.mapping.RemoveLayer(df, slyr)

# Use the uuid module to generate a GUID as part of the output name
# This will ensure a unique output name
output = 'WebMap_{}.{}'.format(str(uuid.uuid1()), Format)
Output_File = os.path.join(arcpy.env.scratchFolder, output)
```

```
# Export the web map
if Format.lower() == 'pdf':
    arcpy.mapping.ExportToPDF(mxd, Output_File)
elif Format.lower() == 'png':
    arcpy.mapping.ExportToPNG(mxd, Output_File)

# Set the output parameter to be the output file of the server job
arcpy.SetParameterAsText(3, Output_File)

# Clean up - delete the map document reference
filePath = mxd.filePath
del mxd, result
os.remove(filePath)
```

ConvertWebMapToMapDocument example 4

In this example, a staged template map document is used that contains vector equivalents of all the possible service layers. After executing the **`ConvertWebMapToMapDocument`** function, the script loops through all the layers in the output map document, removing all layers except the vector layers that correspond to the service layers in the web map JSON. The data view of the output map document is then exported to PNG using the DPI, output height, and output width values returned from the **`ConvertWebMapToMapDocument`** function, as specified in the web app.

```
import arcpy
import os
import uuid

# Input web map json
Web_Map_as_JSON = arcpy.GetParameterAsText(0)

# The template location in the registered folder
templatePath = '//MyComputerName/MyDataStore/FederalLands'

# Get the template map document
templateMxd = os.path.join(templatePath, 'FederalLands.mxd')

# Convert the web map to a map document
result = arcpy.mapping.ConvertWebMapToMapDocument(Web_Map_as_JSON,
templateMxd)
mxd = result.mapDocument

# Reference the data frame that contains the web map
# Note: ConvertWebMapToMapDocument renames the active dataframe in the
template_mxd to "Webmap"
df = arcpy.mapping.ListDataFrames(mxd, 'Webmap')[0]

# Get a list of all service layer names in the map
serviceLayersNames = [slyr.name for slyr in
arcpy.mapping.ListLayers(mxd, data_frame=df)
                      if slyr.isServiceLayer and slyr.visible and not
slyr.isGroupLayer]

# Create a list of all possible vector layer names in the map that
could have a
vectorLayersNames = [vlyr.name for vlyr in
arcpy.mapping.ListLayers(mxd, data_frame=df)
                     if not vlyr.isServiceLayer and not
vlyr.isGroupLayer]

# Get a list of all vector layers that don't have a corresponding
service layer
removeLayerNameList = [vlyrName for vlyrName in vectorLayersNames
                       if vlyrName not in serviceLayersNames]

# Remove all vector layers that don't have a corresponding service
layer
for lyr in arcpy.mapping.ListLayers(mxd, data_frame=df):
    if not lyr.isGroupLayer \
    and not lyr.isServiceLayer \
    and lyr.name in removeLayerNameList \
    and lyr.name in vectorLayersNames:
        arcpy.mapping.RemoveLayer(df, lyr)

# Remove all service layers
# This will leave only vector layers that had corresponding service
layers
for slyr in arcpy.mapping.ListLayers(mxd, data_frame=df):
    if slyr.isServiceLayer:
        arcpy.mapping.RemoveLayer(df, slyr)

# Use the uuid module to generate a GUID as part of the output name
# This will ensure a unique output name
output = 'WebMap_{}.png'.format(str(uuid.uuid1()))
Output_File = os.path.join(arcpy.env.scratchFolder, output)

# Export the web map
arcpy.mapping.ExportToPNG(mxd, Output_File, df,
result.outputSizeWidth,
                          result.outputSizeHeight, result.DPI)

# Set the output parameter to be the output file of the server job
arcpy.SetParameterAsText(1, Output_File)
```

```
# Clean up - delete the map document reference
filePath = mxd.filePath
del mxd, result
os.remove(filePath)
```

ConvertWebMapToMapDocument example 5

In this example, the user supplies a page range that corresponds to the Data Driven Pages enabled in the template map document. The page range is then exported to a multipage PDF document.

```
import arcpy
import os
import uuid

# The template location in the registered folder
templatePath = '//MyComputerName/MyDataStore/WebMap'

# Input WebMap json
Web_Map_as_JSON = arcpy.GetParameterAsText(0)

# Data Driven Page numbers as comma delimited string
DDP_Pages = arcpy.GetParameterAsText(1)
if DDP_Pages == '#' or not DDP_Pages:
    DDP_Pages = "1, 3, 10-13"

# Get the template map document
templateMxd = os.path.join(templatePath, 'DDP.mxd')

# Convert the WebMap to a map document
result = arcpy.mapping.ConvertWebMapToMapDocument(Web_Map_as_JSON,
                                                  templateMxd)
mxd = result.mapDocument

# Use the uuid module to generate a GUID as part of the output name
# This will ensure a unique output name
Output_Name = os.path.join(arcpy.env.scratchFolder,
                           'WebMap_{}.pdf'.format(str(uuid.uuid1())))

# Export the WebMap Data Driven Pages to PDF
mxd.dataDrivenPages.exportToPDF(Output_Name, "RANGE", DDP_Pages)

# Set the output parameter to be the output PDF
arcpy.SetParameterAsText(2, Output_Name)

# Clean up - delete the map document reference
```

```
del mxd, result
os.remove(filePath)
```

CreateGISServerConnectionFile (arcpy.mapping)

Top

Summary

This function creates a connection file that can be used to connect to a GIS Server.

Discussion

This function creates a connection file that can be used to connect to ArcGIS for Server or Spatial Data Server. The connection file can then be used to publish GIS services. For example, the CreateGISServerConnectionFile can be used in conjunction with CreateMapSDDraft, Stage Service, and Upload Service Definition to completely automate the map publishing process.

Related functions:

CreateMapSDDraftCreateGPSDDraftCreateImageSDDraftCreateGeocodeSDDraft

Syntax

CreateGISServerConnectionFile (connection_type, out_folder_path, out_name, server_url, server_type, {use_arcgis_desktop_staging_folder}, {staging_folder_path}, {username}, {password}, {save_username_password})

Parameter	Explanation	Data Type
connection_type	A string that represents the connection type. You can create a connection to use, publish, or administer GIS Services. • USE_GIS_SERVICES —Use GIS Services. • PUBLISH_GIS_SERVICES —Publish GIS Services. • ADMINISTER_GIS_SERVICES —Administer GIS Services.	String
out_folder_path	The folder path where the .ags file will be stored. Entering a value of `GIS Servers` will put the .ags file in the GIS Servers node in the ***Catalog*** window.	String
out_name	The name of the .ags file. The output file extension must end with .ags.	String
server_url	A string that represents the URL to the server.	String
server_type	A string that represents the type of server. • ARCGIS_SERVER —ArcGIS for Server server type • SPATIAL_DATA_SERVER —A Spatial Data Server server type (The default value is ARCGIS_SERVER)	String
use_arcgis_desktop_staging_folder	A Boolean that determines whether to use ArcGIS for Desktop's staging folder. If set to true, you do not have to enter a staging_folder_path as ArcGIS for Desktop's staging folder will be used. This parameter is only used if the connection_type is PUBLISH_GIS_SERVICES or ADMINISTER_GIS_SERVICES. (The default value is True)	Boolean
staging_folder_path	A string that represents the staging folder path. If you will be using this connection to create and save service definitions, you can choose where the service definition files will be staged on disk. By default, they are staged in a folder on your local machine. If this parameter is set to None, ArcGIS for Desktop's staging folder will be used. This parameter is only used if the connection_type is PUBLISH_GIS_SERVICES or ADMINISTER_GIS_SERVICES.	String
username	A string that represents the user name to the GIS server.	String
password	A string that represents the password to the GIS server.	String
save_username_password	A Boolean that represents whether the user name and password to the GIS Server will be saved in the connection file. • SAVE_USERNAME — Save the user name and password in the connection file. • DO_NOT_SAVE_USERNAME — Do not save the user name and password in the connection file. (The default value is True)	Boolean

Code Sample

CreateGISServerConnectionFile example 1

The following script creates an ArcGIS for Server administration connection file in a user-defined folder.

```
import arcpy
outdir = 'C:/Project'
out_folder_path = outdir
out_name = 'test.ags'
server_url = 'http://MyServer:6080/arcgis/admin'
use_arcgis_desktop_staging_folder = False
staging_folder_path = outdir
username = 'admin'
password = 'admin'

arcpy.mapping.CreateGISServerConnectionFile("ADMINISTER_GIS_SERVICES",
                                            out_folder_path,
                                            out_name,
                                            server_url,
                                            "ARCGIS_SERVER",

use_arcgis_desktop_staging_folder,
                                            staging_folder_path,
                                            username,
                                            password,
                                            "SAVE_USERNAME")
```

CreateGISServerConnectionFile example 2

The following script creates an ArcGIS for Server user connection file in the GIS Servers node in the ***Catalog*** window.

```
import arcpy
out_folder_path = 'GIS Servers'
out_name = 'test.ags'
server_url = 'http://MyServer:6080/arcgis/services'

arcpy.mapping.CreateGISServerConnectionFile("USE_GIS_SERVICES",
                                            out_folder_path,
                                            out_name,
                                            server_url,
                                            "ARCGIS_SERVER",
                                            username='admin',
                                            password='admin',

save_username_password=True)
```

CreateMapSDDraft (arcpy.mapping)

Top

Summary

Converts Map Document (.mxd) files to Service Definition Draft (.sddraft) files.

Discussion

CreateMapSDDraft is the first step to automating the publishing of a Map Document to a GIS Server using ArcPy. The output created from the **CreateMapSDDraft** is a Service Definition Draft (.sddraft) file. A Service Definition Draft is the combination of a Map Document, information about the server, and a set of service properties.

Information about the server includes the server connection or server type being published to, the type of service being published, metadata for the service (Item info), and data references (whether or not data is being copied to the server).

Service properties include whether the service supports caching and, if so, the cache settings. Also included are any additional capabilities of the service, such as Feature access or OGC capabilities, along with an appropriate property set for the chosen capability. This method initially uses a set of default service properties. Users can edit these properties using standard third-party XML editors. Moreover, users can automate the modification of these properties using third-party XML libraries such as the xml.dom.minidom standard Python library. See the Modify SDDraft examples below.

Note:

A draft service definition does not contain data. A draft service alone cannot be used to publish a service.

Similar to the AnalyzeForSD function, **CreateMapSDDraft** also returns a Python dictionary containing errors and other potential issues that you should address prior to creating your Service Definition file. For more information about the types of errors, warnings and information messages, and how to access them, see AnalyzeForSD .

A Service Definition Draft can be authored without knowing the specific server connection information. In this case, the **connection_file_path** parameter may be omitted; however, the **server_type** must be provided. A server connection can be provided later when the Service Definition Draft is published using the Upload Service Definition tool.

The Service Definition Draft can then be converted to a fully consolidated Service Definition (.sd) file using the Stage Service tool. Staging compiles all the necessary information needed to successfully publish the GIS resource. If your data is not registered with the server, the data will be added when the Service Definition Draft is staged. Finally, the Service Definition file can be uploaded and published as a GIS service to a specified GIS server using theUpload Service Definition tool. This step takes the Service Definition file, copies it onto the server, extracts required information, and publishes the GIS resource. For more information, see the overview of the Publishing toolset.

Once the .sddraft file has been staged and uploaded to the server, the tools within the Caching toolset can be used to create the tiling scheme for services that have caching enabled, such as the Create Map Server Cachetool. Moreover, the tools within the Caching toolset can be used to modify caching and tiling properties for services that have caching enabled. For example, the Manage Map Server Cache Scales tool can be used to add new scales or delete existing scales from a cache. Modify SDDraft example 7 demonstrates this. The tiling scheme can also be modified by editing the .sddraft file using third-party XML libraries such as the xml.dom.minidom standard Python library. However, due to the complexity of the tiling scheme XML structure, it is recommended to use the Caching toolset whenever possible.

When publishing hosted services to ArcGIS Online or Portal for ArcGIS, sign in information is obtained from the **File > Sign In** dialog box on the ArcGIS for Desktop main menu. Moreover, the Sign In To Portal tool can be used to specify sign in information for some security configurations for Portal for ArcGIS. For more information on hosted services and signing in to ArcGIS Online or Portal for ArcGIS, see the following topics:

What are ArcGIS Online hosted services?Signing into ArcGIS Online in ArcGIS for DesktopManaging portal connections from ArcGIS for Desktop

Note:

The ArcGIS for Server Administrator API also allows you to script common actions you do with your server. For example, stop and start services, edit properties of services, such as the maximum number of instances, grant and revoke user permissions on services, and so on.

You can also create Service Definition Draft files for geoprocessing, image, and geocoding services. See the following related functions:

CreateGPSDDraftCreateImageSDDraftCreateGeocodeSDDraft

Syntax

CreateMapSDDraft (map_document, out_sddraft, service_name, {server_type}, {connection_file_path}, {copy_data_to_server}, {folder_name}, {summary}, {tags})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_sddraft	A string that represents the path and file name for the output Service Definition Draft (`.sddraft`) file.	String
service_name	A string that represents the name of the service. This is the name people will see and use to identify the service. The name can only contain alphanumeric characters and underscores. No spaces or special characters are allowed. The name cannot be more than 120 characters in length.	String
server_type	A string representing the server type. If a **`connection_file_path`** parameter is not supplied, then a **`server_type`** must be provided. If a **`connection_file_path`** parameter is supplied, then the **`server_type`** is taken from the connection file. In this case, you can choose `FROM_CONNECTION_FILE` or skip the parameter entirely. • ARCGIS_SERVER —ArcGIS for Server server type • FROM_CONNECTION_FILE —Get the **`server_type`** as specified in the **`connection_file_path`** parameter • SPATIAL_DATA_SERVER —Spatial Data Server server type • MY_HOSTED_SERVICES —My Hosted Services server type for ArcGIS Online or Portal for ArcGIS (The default value is ARCGIS_SERVER)	String
connection_file_path	A string that represents the path and file name to the ArcGIS for Server connection file (`.ags`). When the **`server_type`** is set to `MY_HOSTED_SERVICES`, **`connection_file_path`** is not required.	String
copy_data_to_server	A Boolean that indicates whether the data referenced in the map document will be copied to the server or not. The**`copy_data_to_server`** parameter is only used if the **`server_type`** is `ARCGIS_SERVER` and the **`connection_file_path`** isn't specified. If the **`connection_file_path`** is specified, then the server's registered data stores are used. For example, if the data in the **`map_document`** is registered with the server, then **`copy_data_to_server`** will always be `False`. Conversely, if the data in the **`map_document`** is not registered with the server, then **`copy_data_to_server`** will always be `True`. When the **`server_type`** is set to `SPATIAL_DATA_SERVER`, **`copy_data_to_server`** will always be `False`. Spatial Data Server services always use registered data and will therefore never copy data to the server. When the **`server_type`** is set to `MY_HOSTED_SERVICES`, **`copy_data_to_server`** will always be `True`. My Hosted Maps services always copy data to the server. (The default value is False)	Boolean
folder_name	A string that represents a folder name to which you want to publish the service definition. If the folder does not currently exist, it will be created. The default folder is the server root level. (The default value is None)	String
summary	A string that represents the Item Description Summary. By default, the **Summary** from the ArcMap ***Map Properties*** dialog box or ***Catalog*** window ***Item Description*** dialog box for the**`map_document`** will be used. Use this parameter to override the user interface summary, or to provide a summary if one does not exist. The summary provided here will not be persisted in the map document. (The default value is None)	String
tags	A string that represents the Item Description Tags. By default, the **Tags** from the ArcMap ***Map Properties*** dialog box or ***Catalog*** window ***Item Description*** dialog box for the**`map_document`** will be used. Use this parameter to override the user interface tags, or to provide tags if they do not exist. The tags provided here will not be persisted in the map document. (The default value is None)	String

Return Value

Data Type	Explanation
Dictionary	Returns a Python Dictionary of information messages, warnings, and errors.

Code Sample

CreateMapSDDraft example 1

The following script demonstrates the complete publishing of map services using **arcpy.mapping** workflow. Automating the publishing of map services can be accomplished by using a combination of **arcpy.mapping** functions and the geoprocessing tools in the Publishing toolset. The workflow begins with a map document that you want to publish. First, use the **arcpy.mapping** function **CreateMapSDDraft** to create a service definition draft. Note that the Item Description, Summary, and Tags for the input map document are overwritten using the **summary** and **tags** parameters. Next, you should analyze the service definition draft for issues that could prevent you from publishing successfully. After analyzing the service definition draft and addressing serious issues, it is time to stage the service definition. Staging takes the service definition draft and consolidates all the information needed to publish the service into a complete service definition. Use the Stage Service geoprocessing tool to stage the service definition. Finally, use the Upload Service Definition geoprocessing tool to upload the service definition to the server and publish the map service.

```
import arcpy

# define local variables
wrkspc = 'C:/Project/'
mapDoc = arcpy.mapping.MapDocument(wrkspc + 'counties.mxd')
con = 'GIS Servers/arcgis on MyServer_6080 (publisher).ags'
service = 'Counties'
sddraft = wrkspc + service + '.sddraft'
sd = wrkspc + service + '.sd'
summary = 'Population Density by County'
tags = 'county, counties, population, density, census'

# create service definition draft
analysis = arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER',
                                          con, True, None, summary,
tags)

# stage and upload the service if the sddraft analysis did not contain
errors
if analysis['errors'] == {}:
    # Execute StageService
    arcpy.StageService_server(sddraft, sd)
    # Execute UploadServiceDefinition
    arcpy.UploadServiceDefinition_server(sd, con)
else:
    # if the sddraft analysis contained errors, display them
    print analysis['errors']
```

CreateMapSDDraft example 2

The following sample script creates a Service Definition Draft (`.sddraft`) file from a Map Document (`.mxd`). It then prints the Python Dictionary of errors, warnings, and information, which is returned from the **CreateMapSDDraft** function. The analysis information contained in the Python dictionary helps to identify potential performance bottlenecks and map errors that you may need to address before you can create a Service Definition (`.sd`) file. This script also demonstrates how to create a Service Definition Draft without specifying the server connection information.

```
import arcpy

mapDoc = arcpy.mapping.MapDocument('C:/Project/counties.mxd')
service = 'Counties'
sddraft = 'C:/Project/' + service + '.sddraft'

analysis = arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER')

for key in ('messages', 'warnings', 'errors'):
  print "----" + key.upper() + "---"
  vars = analysis[key]
  for ((message, code), layerlist) in vars.iteritems():
    print "    ", message, " (CODE %i)" % code
    print "       applies to:",
    for layer in layerlist:
        print layer.name
```

CreateMapSDDraft example 3

The following sample script creates a Service Definition Draft (`.sddraft`) file from a Map Document (`.mxd`) for My Hosted Services. If the service definition draft contains no analysis errors, it is staged using the Stage Servicegeoprocessing tool. Then the Upload Service Definition geoprocessing tool is used to upload the service definition to ArcGIS Online or Portal for ArcGIS. ArcGIS Online sign in information is obtained from the **File > Sign In** dialog box on the ArcGIS for Desktop main menu.

```
import arcpy

mapDoc = arcpy.mapping.MapDocument('C:/Project/counties.mxd')
service = 'Counties'
sddraft = 'C:/Project/{}.sddraft'.format(service)
sd = 'C:/Project/{}.sd'.format(service)

# create service definition draft
analysis = arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'MY_HOSTED_SERVICES')

# stage and upload the service if the sddraft analysis did not contain
errors
if analysis['errors'] == {}:
    # create service definition
    arcpy.StageService_server(sddraft, sd)
    # publish to My Hosted Services
    arcpy.UploadServiceDefinition_server(sd, 'My Hosted Services')
else:
    # if the sddraft analysis contained errors, display them
    print analysis['errors']
```

Modify SDDraft example 1

The following script modifies the Item Information Description element using the xml.dom.minidom standard Python library. The modified Service Definition Draft (`.sddraft`) file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM
# the new description
newDesc = 'US Counties Map'
xml = r"C:\Project\Counties.sddraft"
doc = DOM.parse(xml)
# find the Item Information Description element
descriptions = doc.getElementsByTagName('Description')
for desc in descriptions:
    if desc.parentNode.tagName == 'ItemInfo':
        # modify the Description
        if desc.hasChildNodes():
            desc.firstChild.data = newDesc
        else:
            txt = doc.createTextNode(newDesc)
            desc.appendChild(txt)
# output to a new sddraft
outXml = r"C:\Project\Output\CountiesForWeb.sddraft"
f = open(outXml, 'w')
doc.writexml( f )
f.close()

# analyze the new sddraft for errors
analysis = arcpy.mapping.AnalyzeForSD(outXml)
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print
```

Modify SDDraft example 2

The following script modifies the Staging Settings TextAntialiasingMode element using the xml.dom.minidom standard Python library. The modified Service Definition Draft (`.sddraft`) file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM
# the new TextAntiAliasingMode value
newTextAntialiasingMode = 'Normal'
xml = r"C:\Project\Counties.sddraft"
doc = DOM.parse(xml)
keys = doc.getElementsByTagName('Key')
for key in keys:
    if key.hasChildNodes():
        if key.firstChild.data == 'textAntialiasingMode':
           # modify the TextAntiAliasingMode value
           key.nextSibling.firstChild.data = newTextAntialiasingMode

# output to a new sddraft
outXml = r"C:\Project\Output\CountiesForWeb.sddraft"
f = open(outXml, 'w')
doc.writexml( f )
f.close()

# analyze the new sddraft for errors
analysis = arcpy.mapping.AnalyzeForSD(outXml)
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print
```

Modify SDDraft example 3

The following sample script creates a Service Definition Draft (`.sddraft`) file from a Map Document (`.mxd`). It then enables WMSServer capabilities and sets the Title property by modifying the `.sddraft` file using the xml.dom.minidom standard Python library. The modified `.sddraft` file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM

# Reference map document for CreateSDDraft function.
mapDoc = arcpy.mapping.MapDocument('C:/project/counties.mxd')
# Create service and sddraft variables for CreateSDDraft function.
service = 'Counties'
sddraft = 'C:/Project/' + service + r'.sddraft'

# Create sddraft.
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER')

# These are the properties we will change in the sddraft xml.
soe = 'WMSServer'
soeProperty = 'title'
soePropertyValue = 'USACounties'

# Read the sddraft xml.
doc = DOM.parse(sddraft)
# Find all elements named TypeName. This is where the server object
extension (SOE) names are defined.
typeNames = doc.getElementsByTagName('TypeName')
for typeName in typeNames:
    # Get the TypeName whose properties we want to modify.
    if typeName.firstChild.data == soe:
        extension = typeName.parentNode
        for extElement in extension.childNodes:
            # Enabled SOE.
            if extElement.tagName == 'Enabled':
                extElement.firstChild.data = 'true'
            # Modify SOE property. We have to drill down to the
            if extElement.tagName == 'Props':
                for propArray in extElement.childNodes:
                    for propSet in propArray.childNodes:
                        for prop in propSet.childNodes:
                            if prop.tagName == "Key":
                                if prop.firstChild.data ==
soeProperty:
                                    if
prop.nextSibling.hasChildNodes():

prop.nextSibling.firstChild.data = soePropertyValue
                                    else:
                                        txt =
doc.createTextNode(soePropertyValue)

prop.nextSibling.appendChild(txt)

# Output to a new sddraft.
outXml = "C:/Project/Output/CountiesForWeb.sddraft"
f = open(outXml, 'w')
doc.writexml( f )
f.close()

# Analyze the new sddraft for errors.
analysis = arcpy.mapping.AnalyzeForSD(outXml)
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print
```

Modify SDDraft example 4

The following sample script creates a Service Definition Draft (`.sddraft`) file from a Map Document (`.mxd`). It then disables KmlServer capabilities by modifying the `.sddraft` file using the xml.dom.minidom standard Python library. The modified `.sddraft` file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM

# Reference map document for CreateSDDraft function.
mapDoc = arcpy.mapping.MapDocument('C:/project/counties.mxd')
# Create service and sddraft variables for CreateSDDraft function.
service = 'Counties'
sddraft = 'C:/Project/' + service + '.sddraft'

# Create sddraft.
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER')

# The Server Object Extension (SOE) to disable.
soe = 'KmlServer'

# Read the sddraft xml.
doc = DOM.parse(sddraft)
# Find all elements named TypeName. This is where the server object
extension (SOE) names are defined.
typeNames = doc.getElementsByTagName('TypeName')
for typeName in typeNames:
    # Get the TypeName we want to disable.
    if typeName.firstChild.data == soe:
        extension = typeName.parentNode
        for extElement in extension.childNodes:
            # Disabled SOE.
            if extElement.tagName == 'Enabled':
                extElement.firstChild.data = 'false'

# Output to a new sddraft.
outXml = "C:/Project/Output/CountiesForWeb.sddraft"
doc.writexml( f )
f.close()

# Analyze the new sddraft for errors.
analysis = arcpy.mapping.AnalyzeForSD(outXml)
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print
```

Modify SDDraft example 5

The following sample script creates a Service Definition Draft (`.sddraft`) file for the Spatial Data Server **`server_type`** from a Map Document (`.mxd`). It then changes the service's default web capabilities by modifying the `.sddraft` file using the xml.dom.minidom standard Python library. The modified `.sddraft` file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM

# Reference map document for CreateSDDraft function.
mapDoc = arcpy.mapping.MapDocument('C:/project/SDS.mxd')
# Create service and sddraft variables for CreateSDDraft function.
service = 'CountiesSDS'
sddraft = 'C:/Project/{}.sddraft'.format(service)

# Create sddraft.
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'SPATIAL_DATA_SERVER')

# Read the sddraft xml.
doc = DOM.parse(sddraft)

# Get all the value tags.
values = doc.getElementsByTagName('Value')
for value in values:
    if value.hasChildNodes():
        # Change the default WebCapabilities from
'Query,Create,Update,Delete,Uploads,Editing' to just 'Query'.
        if value.firstChild.data ==
'Query,Create,Update,Delete,Uploads,Editing':
            value.firstChild.data = 'Query'

# Output to a new sddraft.
outXml = "C:/Project/Output/CountiesForSDS.sddraft"
f = open(outXml, 'w')
doc.writexml( f )
f.close()
```

```
analysis = arcpy.mapping.AnalyzeForSD(outXml)
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print
```

Modify SDDraft example 6

The following sample script creates a Service Definition Draft (`.sddraft`) file for the `ARCGIS_SERVER` **`server_type`** from a Map Document (`.mxd`). It then enables caching on the service by modifying the `.sddraft` file using the xml.dom.minidom standard Python library. The modified `.sddraft` file is then saved to a new file. Finally, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function.

```
import arcpy
import xml.dom.minidom as DOM
import os

# define local variables
wrkspc = 'C:/Project/'
mapDoc = arcpy.mapping.MapDocument(wrkspc + 'counties.mxd')
con = 'GIS Servers\arcgis on MyServer_6080 (admin).ags'
service = 'Counties'
sddraft = wrkspc + service + '.sddraft'
sd = os.path.join(wrkspc, "output", service + '.sd')

# create sddraft
if os.path.exists(sddraft): os.remove(sddraft)
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER')

# read sddraft xml
doc = DOM.parse(sddraft)

# turn on caching in the configuration properties
configProps = doc.getElementsByTagName('ConfigurationProperties')[0]
propArray = configProps.firstChild
propSets = propArray.childNodes
for propSet in propSets:
    keyValues = propSet.childNodes
    for keyValue in keyValues:
        if keyValue.tagName == 'Key':
            if keyValue.firstChild.data == "isCached":
                # turn on caching
                keyValue.nextSibling.firstChild.data = "true"

outXml = "C:\Project\Output\CountiesForWeb.sddraft"
if os.path.exists(outXml): os.remove(outXml)
f = open(outXml, 'w')
doc.writexml( f )
f.close()

# analyze new sddraft for errors
analysis = arcpy.mapping.AnalyzeForSD(outXml)

# print dictionary of messages, warnings and errors
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print

del f, doc, mapDoc
```

Modify SDDraft example 7

The following sample script creates a Service Definition Draft (`.sddraft`) file for the `ARCGIS_SERVER` **`server_type`** from a Map Document (`.mxd`). It then enables caching on the service by modifying the `.sddraft` file using the xml.dom.minidom standard Python library. The modified `.sddraft` file is then saved to a new file. Next, the new `.sddraft` file is analyzed for errors using the AnalyzeForSD function. After analyzing the service definition draft, it is time to stage the service definition. Use the Stage Service geoprocessing tool to stage the service definition. Then use the Upload Service Definition geoprocessing tool to upload the service definition to the server and publish the map service. Once the service has been published, the script then calls the Manage Map Server Cache Scales geoprocessing tool which updates the scale levels in an existing cached map or image service. Use this tool to add new scales or delete existing scales from a cache. Finally, the script calls the Manage Map Server Cache Tiles geoprocessing tool to create map service cache tiles.

```
import arcpy
import xml.dom.minidom as DOM
import os

# define local variables
wrkspc = 'C:/Project'
systemFolder =
'C:/Users/<username>/AppData/Roaming/ESRI/Desktop10.2/ArcCatalog'
server = 'arcgis on MyServer_6080 (publisher)'
service = 'Counties'

# build paths to data
mapDoc = arcpy.mapping.MapDocument(os.path.join(wrkspc,
'counties.mxd'))
connection = os.path.join(systemFolder, server + '.ags')
mapServer = os.path.join(systemFolder, server, service + '.MapServer')
sddraft = os.path.join(wrkspc, service + '.sddraft')
sd = os.path.join(wrkspc, 'output', service + '.sd')

# create sddraft
if os.path.exists(sddraft): os.remove(sddraft)
arcpy.mapping.CreateMapSDDraft(mapDoc, sddraft, service,
'ARCGIS_SERVER')

doc = DOM.parse(sddraft)

# turn on caching in the configuration properties
configProps = doc.getElementsByTagName('ConfigurationProperties')[0]
propArray = configProps.firstChild
propSets = propArray.childNodes
for propSet in propSets:
    keyValues = propSet.childNodes
    for keyValue in keyValues:
        if keyValue.tagName == 'Key':
            if keyValue.firstChild.data == "isCached":
                # turn on caching
                keyValue.nextSibling.firstChild.data = "true"

# output to a new sddraft
outXml = os.path.join(wrkspc, 'output', service + '.sddraft')
if os.path.exists(outXml): os.remove(outXml)
f = open(outXml, 'w')
doc.writexml( f )
f.close()

# analyze new sddraft for errors
analysis = arcpy.mapping.AnalyzeForSD(outXml)

# print dictionary of messages, warnings and errors
for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), layerlist) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
        print "       applies to:",
        for layer in layerlist:
            print layer.name,
        print

# stage and upload the service if the sddraft analysis did not contain
errors
if analysis['errors'] == {}:
    # Execute StageService
```

```
    if os.path.exists(sd): os.remove(sd)
    arcpy.StageService_server(outXml, sd)
    # Execute UploadServiceDefinition
    print "Uploading Service Definition..."
    arcpy.UploadServiceDefinition_server(sd, connection)
    # Print messaging from UploadServiceDefinition
    x = 0
    while x < arcpy.GetMessageCount():
        arcpy.AddReturnMessage(x)
        x = x + 1
else:
    print "{} contained errors. StageService and
UploadServiceDefinition aborted.".format(sddraft)
    exit()

print "Updating cache scales..."
scaleValues =
"100000000;10000000;5000000;1000000;500000;250000;125000;64000;5250"
arcpy.ManageMapServerCacheScales_server(mapServer, scaleValues)

print "Creating tiles..."
arcpy.ManageMapServerCacheTiles_server(mapServer, "100000000",
"RECREATE_ALL_TILES", "3")

print "Uploaded service and created tiles."
```

Modify SDDraft example 8

The following sample script shows how to update an existing feature service that is hosted on ArcGIS Online. For example, an organization may want to push updates to keep synchronized with the daily changes made by itsArcGIS for Desktop users. This sample script will demonstrate how to:

- Turn a map document into an `.sddraft` file.
- Modify the XML inside with the appropriate settings.
- Analyze the `.sddraft` file for errors.
- Stage the `.sddraft` file into an `.sd` (service definition) file.
- Upload the service to ArcGIS Online. Note that this code shares the feature service with everyone on ArcGIS Online.

Update the variables with your path to the MXD file. The script will create temporary drafts in the same location in which the script is saved. ArcGIS Online sign in information is obtained from the **File > Sign In** dialog box on theArcGIS for Desktop main menu.

```
import arcpy, os, sys
import xml.dom.minidom as DOM

arcpy.env.overwriteOutput = True

# Update these variables
# The tempPath variable is a relative path which is the same directory
# this script is saved to. You can modify this value to a path on your
# system to hold the temporary files.
serviceName = "importantPoints"
tempPath = sys.path[0]
path2MXD = r"C:\path2MXD\pts.mxd"

# All paths are built by joining names to the tempPath
SDdraft = os.path.join(tempPath, "tempdraft.sddraft")
newSDdraft = os.path.join(tempPath, "updatedDraft.sddraft")
SD = os.path.join(tempPath, serviceName + ".sd")

mxd = arcpy.mapping.MapDocument(path2MXD)
arcpy.mapping.CreateMapSDDraft(mxd, SDdraft, serviceName,
"MY_HOSTED_SERVICES")

# Read the contents of the original SDDraft into an xml parser
doc = DOM.parse(SDdraft)
```

```
# The follow 5 code pieces modify the SDDraft from a new MapService
# with caching capabilities to a FeatureService with Query,Create,
# Update,Delete,Uploads,Editing capabilities. The first two code
# pieces handle overwriting an existing service. The last three pieces
# change Map to Feature Service, disable caching and set appropriate
# capabilities. You can customize the capabilities by removing items.
# Note you cannot disable Query from a Feature Service.
tagsType = doc.getElementsByTagName('Type')
for tagType in tagsType:
    if tagType.parentNode.tagName == 'SVCManifest':
        if tagType.hasChildNodes():
            tagType.firstChild.data =
"esriServiceDefinitionType_Replacement"

tagsState = doc.getElementsByTagName('State')
for tagState in tagsState:
    if tagState.parentNode.tagName == 'SVCManifest':
        if tagState.hasChildNodes():
            tagState.firstChild.data = "esriSDState_Published"

# Change service type from map service to feature service
typeNames = doc.getElementsByTagName('TypeName')
for typeName in typeNames:
    if typeName.firstChild.data == "MapServer":
        typeName.firstChild.data = "FeatureServer"

# Turn off caching
configProps = doc.getElementsByTagName('ConfigurationProperties')[0]
propArray = configProps.firstChild
propSets = propArray.childNodes
for propSet in propSets:
    keyValues = propSet.childNodes
    for keyValue in keyValues:
        if keyValue.tagName == 'Key':
            if keyValue.firstChild.data == "isCached":
                keyValue.nextSibling.firstChild.data = "false"

# Turn on feature access capabilities
configProps = doc.getElementsByTagName('Info')[0]
```

```
propArray = configProps.firstChild
propSets = propArray.childNodes
for propSet in propSets:
    keyValues = propSet.childNodes
    for keyValue in keyValues:
        if keyValue.tagName == 'Key':
            if keyValue.firstChild.data == "WebCapabilities":
                keyValue.nextSibling.firstChild.data =
"Query,Create,Update,Delete,Uploads,Editing"

# Write the new draft to disk
f = open(newSDdraft, 'w')
doc.writexml( f )
f.close()

# Analyze the service
analysis = arcpy.mapping.AnalyzeForSD(newSDdraft)

if analysis['errors'] == {}:
    # Stage the service
    arcpy.StageService_server(newSDdraft, SD)

    # Upload the service. The OVERRIDE_DEFINITION parameter allows you
to override the
    # sharing properties set in the service definition with new
values. In this case,
    # the feature service will be shared to everyone on ArcGIS.com by
specifying the
    # SHARE_ONLINE and PUBLIC parameters. Optionally you can share to
specific groups
    # using the last parameter, in_groups.
    arcpy.UploadServiceDefinition_server(SD, "My Hosted Services",
serviceName,
                                         "", "", "", "",
"OVERRIDE_DEFINITION", "SHARE_ONLINE",
                                         "PUBLIC",
"SHARE_ORGANIZATION", "")

    print "Uploaded and overwrote service"
```

```
    # If the sddraft analysis contained errors, display them and quit.
    print analysis['errors']
```

DeleteMapService (arcpy.mapping)

Top

Summary

Legacy:

This method has been deprecated starting at ArcGIS 10.1 and will return a runtime error. Please consult the ArcGIS documentation for the usage of the new ArcGIS for Server Administrator API.

Deletes a map service from a designated ArcGIS for Server.

Discussion

This method has been deprecated at ArcGIS 10.1 and will return a runtime error.

Starting at version 10.1, ArcGIS for Server has a new architecture which may require you to adjust how you work with the server. See the following help topics for more information: What to expect when migrating to ArcGIS 10.2 for Server and Migration to ArcGIS 10.2 for Server.

You can delete Map Services using the ArcGIS for Server Administrator API which is available through the ArcGIS Site Directory. The default URL of the ArcGIS for Server Site Directory is as follows:

`http://<server name>:6080/arcgis/admin`

Note:

The URL of the Site Directory may vary if you have configured ArcGIS for Server Web Adaptor to work with your site. For example, the inclusion of the port number, 6080, may not be required. Check your web adaptor configuration to obtain the correct URL.

An example of using Python and the ArcGIS for Server Administrator API to delete Map Services is below:

```
import json
import urllib
import urllib2

def gentoken(url, username, password, expiration=60):
    query_dict = {'username':   username,
                  'password':   password,
                  'expiration': str(expiration),
                  'client':     'requestip'}
    query_string = urllib.urlencode(query_dict)
    return json.loads(urllib.urlopen(url + "?f=json",
query_string).read())['token']

def deleteservice(server, servicename, username, password, token=None,
port=6080):
    if token is None:
        token_url =
"http://{}:{}/arcgis/admin/generateToken".format(server, port)
        token = gentoken(token_url, username, password)
    delete_service_url =
"http://{}:{}/arcgis/admin/services/{}/delete?token={}".format(server,
port, servicename, token)
    urllib2.urlopen(delete_service_url, ' ').read() # The ' ' forces
POST

# if you need a token, execute this line:
deleteservice("<server>", "<service>.MapServer", "<admin username>",
"<admin password>")

# if you already have a token, execute this line:

deleteservice("<server>", "<service>.MapServer", None, None,
token='<token string>')
```

Syntax

DeleteMapService (connection_url_or_name, server, service_name, {folder_name}, {connection_username}, {connection_password}, {connection_domain})

Parameter	Explanation	Data Type
connection_url_or_name	A string that represents the URL of the ArcGIS for Server for which you want to delete a service.	String
server	A string that represents the ArcGIS for Server host name.	String
service_name	A string that represents the name of the service. This is the name people will see and use to identify the service. The name can only contain alphanumeric characters and underscores. No spaces or special characters are allowed. The name cannot be more than 120 characters in length.	String
folder_name	A string that represents a folder name.	String
connection_username	A string that represents a user name used to connect to ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_password	A string that represents a password used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_domain	A string that represents a domain name used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String

ExportReport (arcpy.mapping)

Top

Summary

Exports a formatted, tabular report using data from layers or stand-alone tables in a map document along with the report template information that is provided in a report layout file (`.rlf`).

Discussion

The **`ExportReport`** function provides a mechanism to automate the generation of reports that are originally authored in a map document (`.mxd`) using the reporting tools available in ArcMap. A report layout file authored in ArcMap is a template that stores information about the content and placement of the items in a report. The report layout file is used along with the source data in a map document to create output reports. The source data can also have associated joins and related table information that is used within the report.

Note:

The **`ExportReport`** function has a dependency on the ArcMap installation. Therefore, **`ExportReport`** can only be executed on machines that have ArcMap installed. **`ExportReport`** will not run using stand-alone installations of ArcGIS Engine or ArcGIS for Server. Moreover, **`ExportReport`** will not work as a Geoprocessing Service.

The source data must exist in a map document (`.mxd`) or layer file (`.lyr`), and the dataset field names must match those in the report layout file in order for **`ExportReport`** to execute properly. A report layout file can be used with different data sources provided that the data source has the same dataset field names. If the data is similar but the data source field names are different, then the **`field_map`** parameter can be used to remap the report layout file fields to different field names in the source dataset.

There are numerous elements authored in a report layout file within ArcMap that are not directly exposed through the **`ExportReport`** function. It may be necessary to author a collection of specific templates for generating and exporting different report scenarios.

The reporting tools in the ArcMap user interface have a button called **Dataset options**. This allows the author to choose only one of four ways in which records will be processed: **All**, **Selected Set**, **Visible Extent**, and **Definition Query**.

The **`dataset_option`** parameter serves this same purpose and takes a keyword. A **`dataset_option`** of `ALL` or `SELECTED` will simply process the appropriate records. If the **`dataset_option`** is set to`DEFINITION_QUERY`, then a valid string needs to be provided for the **`report_definition_query`** parameter. If the **`dataset_option`** is set to `EXTENT`, then a valid Extent object needs to be provided for the **`extent`**parameter.

Similar to the user interface, it is only possible to use one **`dataset_option`** keyword at a time. For example, in the user interface, it is not possible to process only the selected set of records and only those in the visible extent. The same rule applies to the **`ExportReport`** function; only one parameter can be set at a time. If more than one of these parameters is set, they will overwrite each other. However, standard ArcPy attribute and/or spatial query functions can be combined together and coupled with a value of `SELECTED` for the **`dataset_option`** parameter to generate the final, desired report. An example is provided below.

The **`page_range`** parameter allows you to generate a report for only a subset of pages. This can be a continuous group of pages (`5-12`) or even a discontinuous set of pages (`3,7`). In these cases, when you generate a report and the **`starting_page_number`** is set to `1`, the printed page numbers will match what you've entered (`Page 3, Page 7`). This was designed for scenarios where only a selected number of pages will be reprinted and inserted into already existing reports. If the scenario requires that the output page numbers be continuous, for example (`Page 1, Page 2),` then you must set the **`starting_page_number`** to equal the page number of interest, and set the **`page_range`** to be a single page. This means that **`ExportReport`** will need to be executed once for each page. For example, once with `starting_page_number = 3, page_range="3"` and again with`starting_page_number = 7, page_range="7"`.

The **`field_map`** parameter is only needed if the dataset field names are different between the **`source_data`** and the field names used to build the report layout file. When creating the **`field_map`**, only the fields used in the report need to be added to the dictionary. If fields are dropped from the field map dictionary, those fields will be dropped from the report. The mapped fields must have identical data types, and the field names in the **`field_map`**are case sensitive.

It is possible to use arcpy.mapping to build reports so you can put a map in your report. Pictures in a report have a property called **Source Image**. Arcpy.mapping does not have access to this value but if the source image value is pointing to a path on disk, arcpy.mapping can update the file on disk using an export operation and the report engine will use whatever image is currently available. In a Data Driven Pages example, you could change the picture based on the current extent, for example, with arcpy.mapping before calling the next page. An example is provided below.

For more information about reporting in ArcGIS, see the following introductory topics:

- What are reports in ArcGIS
- Creating a report

Syntax

ExportReport (report_source, report_layout_file, output_file, {dataset_option}, {report_title}, {starting_page_number}, {page_range}, {report_definition_query}, {extent}, {field_map})

Parameter	Explanation	Data Type
report_source	A reference to a Layer or TableView object.	Object
report_layout_file	A string that represents the path and file name of the report layout file (`.rlf`).	String
output_file	A string that represents the path and file name of the output file. The specified file extension controls the output format. The following extensions/formats are supported: `.htm`, `.html`, `.pdf`, `.rtf`, `.tif`, `.tiff`, `.txt`, and `.xls`.	String
dataset_option	A keyword that specifies which dataset rows will be processed in the output report. This value will override the **Dataset Options**value stored in the report layout file which is found in the **Report Properties** dialog box. If the **`dataset_option`** parameter is not set, it will default to the value stored in the report layout file. If the **`dataset_option`** is set to `DEFINITION_QUERY`, then a valid string needs to be provided for the **`report_definition_query`** parameter. If the **`dataset_option`** is set to `EXTENT`, then a valid Extent object needs to be provided for the **`extent`** parameter. Because the **`dataset_option`** keyword controls which additional parameter to use, only one of these parameters can be set at a time, just like in the user interface. • ALL —Override the report layout file dataset option and process all data source records. • DEFINITION_QUERY —Override the report layout file dataset option and provide a new or updated definition query. • EXTENT —Override the report layout file dataset option and provide a new or updated extent. • SELECTED —Override the report layout file dataset option and process only the selected records. • USE_RLF —Use the settings saved in the report layout file. (The default value is USE_RLF)	String
report_title	A string that represents the report's title which appears in the report layout file header section.	String
starting_page_number	A number that represents the printed page number for the first page of the output report. This value is useful for offsetting page numbers for reports that get appended to the end of existing documents. (The default value is 1)	Long
page_range	A string that identifies the report pages to be exported to file (for example, `1, 3, 5-12`).	String
report_definition_query	A string that represents a valid definition query that controls which rows will be exported to the output report. This parameter can only be set if the **`dataset_option`** parameter is set to `DEFINITION_QUERY`.This value will overwrite any settings stored in the report layout file. If the **`report_source`** layer or table has an existing definition query, then the **`report_definition_query`** will be applied to the existing subset of records.	String
extent	A geoprocessing Extent object. This parameter can only be set if the **`dataset_option`** parameter is set to `EXTENT`. When an**`extent`** object is passed into this parameter, the rows will be based on those features that intersect the extent.	Extent
field_map	This parameter allows you to use a report layout file with a data source that has similar data types but different field names. A dictionary of mapped field names is used to remap the fields used in the report layout file with the new fields in the data source. The following shows an example of the **`field_map`** dictionary structure: `field_map={'rlf_field1':'data_source_field1', 'rlf_field2':'data_source_field2'}`	Dictionary

Code Sample

ExportReport example 1

The following script will export a report to a PDF file using the layer's selected features in the map document. Because other optional parameters are being skipped, the **`extent`** parameter name is explicitly entered.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
lyr = arcpy.mapping.ListLayers(mxd, "Accidents", df)[0]
arcpy.mapping.ExportReport(lyr,
                           r"C:\Project\Project.rlf",
                           r"C:\Project\Output\ProjectReport.pdf",
                           "EXTENT",
                           extent=df.extent)
del mxd
```

ExportReport example 2

The following script extends the example above to demonstrate how to combine a spatial selection and an attribute selection to generate the desired report. This is required because only one `dataset_option` can be used at a time. The results are combined into a single selection and uses a value of `SELECTED` for the `dataset_option` parameter.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
lyr = arcpy.mapping.ListLayers(mxd, "Accidents", df)[0]
#The DataFrame extent object is converted into a polygon feature so it
can be used with the SelectLayerByLocation function.
dfAsFeature = arcpy.Polygon(arcpy.Array([df.extent.lowerLeft,
df.extent.lowerRight, df.extent.upperRight, df.extent.upperLeft]),
                            df.spatialReference)
arcpy.SelectLayerByLocation_management(lyr, "INTERSECT", dfAsFeature,
"", "NEW_SELECTION")
arcpy.SelectLayerByAttribute_management(lyr, "SUBSET_SELECTION",
"\"Accidents\" > 3")
arcpy.mapping.ExportReport(lyr,
                           r"C:\Project\Project.rlf",
                           r"C:\Project\Output\ProjectReport.pdf",
                           "SELECTED")
del mxd
```

ExportReport example 3

The following script uses an existing report layout file against a different dataset with different field names. A new title is used to overwrite the report layout file's title and the fields are remapped using the `field_map` parameter.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
lyr = arcpy.mapping.ListLayers(mxd, "Traffic Lights", df)[0]
arcpy.mapping.ExportReport(lyr,
                           r"C:\Project\Project.rlf",
                           r"C:\Project\Output\ProjectReport.pdf",
                           report_title="Traffic Light Coordinates",

field_map={'Accidents':'LightID','X':'Longitude','Y':'Latitude'})
del mxd
```

ExportReport example 4

The following script demonstrates how to insert a map into a report. A multipage report will be generated. Each page has a large picture that displays the current data frame extent and a report that provides information about the features within the extent. After each data frame extent is changed, the data frame is exported to a `.emf` file, and **`ExportReport`** is used to create a single `PDF`. All pages are combined into a single, final `PDF`.

```
import arcpy, os
path = os.getcwd()  #a relative path allowing for easy packaging

#Create PDF and remove if it already exists
pdfPath = path + r"\States_SubRegions.pdf"
if os.path.exists(pdfPath):
  os.remove(pdfPath)
pdfDoc = arcpy.mapping.PDFDocumentCreate(pdfPath)

subRegionList = ["East North Central", "East South Central", "Middle
Atlantic", "Mountain", "New England", "Pacific", "South Atlantic",
"West North Central", "West South Central"]
```

```
df = arcpy.mapping.ListDataFrames(mxd)[0]
lyr = arcpy.mapping.ListLayers(mxd, "States")[0]

pageCount = 1
for region in subRegionList:
  #Generate image for each sub region
  whereClause = "\"SUB_REGION\" = '" + region + "'"
  lyr.definitionQuery = whereClause
  arcpy.SelectLayerByAttribute_management(lyr, "NEW_SELECTION",
whereClause)
  df.extent = lyr.getSelectedExtent()
  arcpy.SelectLayerByAttribute_management(lyr, "CLEAR_SELECTION")
  arcpy.mapping.ExportToEMF(mxd, path + "\RegionalPicture.emf", df)
#single file

  #Generate report
  arcpy.mapping.ExportReport(report_source=lyr,
                             report_layout_file=path +
r"\States_SubRegions.rlf",
                             output_file=path + r"\temp" +
str(pageCount) + ".pdf",
                             starting_page_number=pageCount)

  #Append pages into final output
  pdfDoc.appendPages(path + r"\temp" + str(pageCount) + ".pdf")
  os.remove(path + r"\temp.pdf")
  pageCount = pageCount + 1

pdfDoc.saveAndClose()
del mxd
```

ExportToAI (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to Adobe Illustrator (AI) format.

Discussion

Adobe Illustrator (AI) files are an excellent format for postprocessing in Adobe Illustrator as well as an interchange format for publishing. The ArcMap AI format preserves most layers from the ArcMap table of contents. However, the Adobe Illustrator file format that ArcMap writes does not support font embedding, so users who do not have the Esri fonts installed may not be able to view AI files with the proper symbology. AI exports from ArcMap can define colors in CMYK or RGB values.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters.
Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.
Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToAI (map_document, out_ai, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {picture_symbol}, {convert_markers})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_ai	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 300)	Integer
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample ratio of 3. • FASTER —An output image quality resample ratio of 4. • FASTEST —An output image quality resample ratio of 5. (The default value is BEST)	String

colorspace	A string that defines which color space will be written to the export file. • CMYK —Cyan, Magenta,Yellow, and blacK color model. • RGB —Red, Green, and Blue color model. (The default value is RGB)	String
picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture marker/fill. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String
convert_markers	A Boolean that controls the coversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to True disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean

Code Sample

ExportToAI example 1

This script opens a map document and exports the page layout to an Adobe Illustrator file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToAI(mxd, r"C:\Project\Output\Project.ai")
del mxd
```

ExportToAI example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **df_export_width** and **df_export_height** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToAI(mxd,
r"C:\Project\Output\ProjectDataFrame.ai", df,
                         df_export_width=1600,
                         df_export_height=1200)
del mxd
```

ExportToBMP (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Microsoft Windows Bitmap (BMP) format.

Discussion

BMP files are simple, native Windows raster images. BMPs can store pixel data at several bit depths and can be compressed using the lossless RLE method. However, in general, BMPs are much larger than formats such as JPEG or PNG. They do not scale as well as vector files and may appear blocky or jagged when increased in size. BMPs generated from the data view in ArcMap can be generated with an accompanying world file so that they can be used as georeferenced raster data. BMPs without a world file are commonly used as inserted graphics in other documents.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters. Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.

World files are not generated for page layouts; a referenced data frame must be provided or the export will fail.

Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToBMP (map_document, out_bmp, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {rle_compression})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_bmp	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 96)	Integer
world_file	If set to `True`, a georeferenced world file is created. The file contains pixel scale information and real-world coordinate information. (The default value is False)	Boolean
color_mode	This value specifies the number of bits used to describe color. • 24-BIT_TRUE_COLOR —24-bit true color. • 8-BIT_PALETTE —8-bit palette. • 8-BIT_GRAYSCALE —8-bit grayscale. • 1-BIT_MONOCHROME_MASK —1-bit monochrome mask. • 1-BIT_MONOCHROME_THRESHOLD —1-bit	String

	monochrome threshold. (The default value is 24-BIT_TRUE_COLOR)	
rle_compression	This value represents a compression scheme. • NONE —Compression is not applied. • RLE —Run-length encoded compression. (The default value is NONE)	String

Code Sample

ExportToBMP example 1

This script opens a map document and exports the page layout to a BMP file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToBMP(mxd, r"C:\Project\Output\Project.bmp")
del mxd
```

ExportToBMP example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **df_export_width** and **df_export_height** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail. Setting **world_file = True** generates a georeferenced world file in the same directory as the output file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToBMP(mxd,
r"C:\Project\Output\ProjectDataFrame.bmp", df,
                          df_export_width=1600,
                          df_export_height=1200,
                          world_file=True)
del mxd
```

ExportToEMF (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Enhanced Metafile (EMF) format.

Discussion

EMF files are native Windows graphics files that can contain a mixture of vector and raster data. They are useful for embedding in Windows documents because the vector portions of the EMF can be resized without loss of quality. However, since EMF does not support font embedding and is exclusively a Windows format, it is not commonly used as an interchange format between users.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters.
Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.
Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToEMF (map_document, out_emf, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {description}, {picture_symbol}, {convert_markers})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_emf	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 300)	Integer
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample ratio of 3. • FASTER —An output image quality resample ratio of 4. • FASTEST —An output image quality resample ratio of 5. (The default value is BEST)	String
description	A string that assigns a description to the output file.	String

picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture marker/fill. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String
convert_markers	A Boolean that controls the coversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to True disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean

Code Sample

ExportToEMF example 1

This script opens a map document and exports the page layout to an EMF file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToEMF(mxd, r"C:\Project\Output\Project.emf")
del mxd
```

ExportToEMF example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToEMF(mxd,
r"C:\Project\Output\ProjectDataFrame.emf", df,
                          df_export_width=1600,
                          df_export_height=1200)
del mxd
```

ExportToEPS (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to an Encapsulated Postscript (EPS) format.

Discussion

EPS files use the PostScript page description language to describe vector and raster objects. PostScript is the publishing industry standard for high-end graphics files, cartography, and printing. EPS files can be edited in many drawing applications or placed as a graphic in most page layout applications. EPS files exported from ArcMap support embedding of fonts so that users who do not have Esri Fonts installed can still view the proper symbology. EPS exports from ArcMap can define colors in CMYK or RGB values.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters.
Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.
Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToEPS (map_document, out_eps, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {ps_lang_level}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {jpeg_compression_quality})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_eps	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 300)	Integer
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample ratio of 3. • FASTER —An output image quality resample	String

	ratio of 4. • FASTEST —An output image quality resample ratio of 5. (The default value is BEST)	
colorspace	A string that defines the colorspace of the export file. • CMYK —Cyan, Magenta,Yellow, and blacK color model. • RGB —Red, Green, and Blue color model. (The default value is RGB)	String
ps_lang_level	A number that represents the PostScript Language level. Level 3 is the most recent release, but some older PostScript interpreters may not be able to read files created using this version. Valid levels are 2 and 3. (The default value is 3)	Integer
image_compression	A string that defines the compression scheme used to compress image or raster data in the output file. • ADAPTIVE —Automatically selects the best compression type for each image on the page. JPEG will be used for large images with many unique colors. DEFLATE will be used for all other images. • JPEG —A lossy data compression. • DEFLATE —A lossless data compression. • LZW —Lempel-Ziv-Welch, a lossless data compression. • NONE —Compression is not applied. • RLE —Run-length encoded compression. (The default value is ADAPTIVE)	String
picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture marker/fill. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String
convert_markers	A Boolean that controls the coversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to True disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean
embed_fonts	A Boolean that controls the embedding of fonts in export files. Font embedding allows text and character markers to be displayed correctly when the document is viewed on a computer that does not have necessary fonts installed. (The default value is True)	Boolean
jpeg_compression_quality	A number that controls compression quality value when **`image_compression`** is set to ADAPTIVE or JPEG. The valid range is 1 to 100. A **`jpeg_compression_quality`** of 100 provides the best quality images but creates large export files. The recommended range is between 70 and 90. (The default value is 80)	Integer

Code Sample

ExportToEPS example 1

This script opens a map document and exports the page layout to an EPS file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToEPS(mxd, r"C:\Project\Output\Project.eps")
del mxd
```

ExportToEPS example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToEPS(mxd,
r"C:\Project\Output\ProjectDataFrame.eps", df,
                          df_export_width=1600,
                          df_export_height=1200)
del mxd
```

ExportToGIF (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Graphic Interchange (GIF) format.

Discussion

GIF files are a standard raster format for use on the Web. GIFs cannot contain more than 256 colors (8 bits per pixel), which along with optional lossless RLE or LZW compression makes them smaller than other file formats. They are a good choice for maps that contain a limited number of colors, but may not display continuous raster data correctly due to the color limitation. GIF files also have the ability to define a transparent color; part of the image can display as transparent in a Web browser, allowing backgrounds, images, or colors to show through. GIFs exported from the data view in ArcMap can be generated with an accompanying world file for use as georeferenced raster data.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters. Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.

World files are not generated for page layouts; a referenced data frame must be provided or the export will fail.

Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToGIF (map_document, out_gif, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {gif_compression}, {background_color}, {transparent_color}, {interlaced})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_gif	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 96)	Integer
world_file	If set to `True`, a georeferenced world file is created. The file contains pixel scale information and real-world coordinate information. (The default value is False)	Boolean
color_mode	This value specifies the number of bits used to describe color. • 24-BIT_TRUE_COLOR —24-bit true color. • 8-BIT_PALETTE —8-bit palette. • 8-BIT_GRAYSCALE —8-bit grayscale. • 1-BIT_MONOCHROME_MASK —1-bit monochrome mask. • 1-BIT_MONOCHROME_THRESHOLD —1 bit	String

	monochrome threshold. (The default value is 8-BIT_PALETTE)	
gif_compression	This value represents a compression scheme. • LZW —Lempel-Ziv-Welch, a lossless data compression. • NONE —Compression is not applied. • RLE —Run-length encoded compression. (The default value is NONE)	String
background_color	A defined color is used as a background to the image, or as a mask in the case of monochrome masked exports. (The default value is 255, 255, 255)	String
transparent_color	A defined color to be displayed as transparent in the image.	String
interlaced	If set to `True`, an interlaced image will be created. An interlaced image displays as a series of scan lines rather than as a whole image at one time. (The default value is False)	Boolean

Code Sample

ExportToGIF example 1

This script opens a map document and exports the page layout to a GIF file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToGIF(mxd, r"C:\Project\Output\Project.gif")
del mxd
```

ExportToGIF example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail. Setting **`world_file = True`** generates a georeferenced world file in the same directory as the output file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToGIF(mxd,
r"C:\Project\Output\ProjectDataFrame.gif", df,
                          df_export_width=1600,
                          df_export_height=1200,
                          world_file=True)
del mxd
```

ExportToJPEG (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Joint Photographic Experts Group (JPEG) format.

Discussion

JPEG files are compressed image files. They support 24-bit color and can be substantially more compact than many other file types. The JPEG compression algorithm is lossy and is not as well suited for line drawings and other textual or iconic graphics, and thus the PNG and GIF formats are preferred for these types of images.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters. Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.

World files are not generated for page layout exports; a referenced data frame must be provided or the export will fail.

Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToJPEG (map_document, out_jpeg, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {jpeg_quality}, {progressive})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_jpeg	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 96)	Integer
world_file	If set to `True`, a georeferenced world file is created. The file contains pixel scale information and real-world coordinate information. (The default value is False)	Boolean
color_mode	This value specifies the number of bits used to describe color. • 24-BIT_TRUE_COLOR —24-bit true color. • 8-BIT_PALETTE —8-bit palette. • 8-BIT_GRAYSCALE —8-bit grayscale. • 1-BIT_MONOCHROME_MASK —1-bit monochrome mask.	String

	• 1-BIT_MONOCHROME_THRESHOLD —1-bit monochrome threshold. (The default value is 24-BIT_TRUE_COLOR)	
jpeg_quality	This value (0–100) controls the amount of compression applied to the output image. For JPEG, image quality is adversely affected the more compression is applied. A higher quality (highest = 100) setting will produce sharper images and larger file sizes. A lower quality setting will produce more image artifacts and smaller files. (The default value is 100)	Integer
progressive	If set to `True`, a progressive JPEG file will be created. A progressive image is one that displays in a series of scans of increasing quality rather than displaying the whole image at once. (The default value is False)	Boolean

Code Sample

ExportToJPEG example 1

This script opens a map document and exports the page layout to a JPEG file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToJPEG(mxd, r"C:\Project\Output\Project.jpg")
del mxd
```

ExportToJPEG example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail. Setting **`world_file = True`** generates a georeferenced world file in the same directory as the output file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToJPEG(mxd,
r"C:\Project\Output\ProjectDataFrame.jpg", df,
                           df_export_width=1600,
                           df_export_height=1200,
                           world_file=True)
del mxd
```

ExportToPDF (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Portable Document Format (PDF).

Discussion

PDF files are designed to be consistently viewable and printable across different platforms. They are commonly used for distributing documents on the Web and are becoming a standard interchange format for content delivery. ArcMap PDFs are editable in many graphics applications and retain annotation, labeling, and attribute data for map layers from the ArcMap table of contents. PDF exports from ArcMap support embedding of fonts and thus can display symbology correctly even if the user does not have Esri fonts installed. PDF exports from ArcMap can define colors in CMYK or RGB values.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters.
Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.
Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToPDF (map_document, out_pdf, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {colorspace}, {compress_vectors}, {image_compression}, {picture_symbol}, {convert_markers}, {embed_fonts}, {layers_attributes}, {georef_info}, {jpeg_compression_quality})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_pdf	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 300)	Integer
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample	String

	ratio of 3. • FASTER —An output image quality resample ratio of 4. • FASTEST —An output image quality resample ratio of 5. (The default value is BEST)	
colorspace	A string that defines the colorspace of the export file. Valid values are CYMK and RGB. • CMYK —Cyan, Magenta,Yellow, and blacK color model. • RGB —Red, Green, and Blue color model. (The default value is RGB)	String
compress_vectors	A Boolean that controls compression of vector and text portions of the output file. Image compression is defined separately. (The default value is True)	Boolean
image_compression	A string that defines the compression scheme used to compress image or raster data in the output file. • ADAPTIVE —Automatically selects the best compression type for each image on the page. JPEG will be used for large images with many unique colors. DEFLATE will be used for all other images. • JPEG —A lossy data compression. • DEFLATE —A lossless data compression. • LZW —Lempel-Ziv-Welch, a lossless data compression. • NONE —Compression is not applied. • RLE —Run-length encoded compression. (The default value is ADAPTIVE)	String
picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture marker/fill. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String

convert_markers	A Boolean that controls the coversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to True disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean
embed_fonts	A Boolean that controls the embedding of fonts in the export file. Font embedding allows text and character markers to be displayed correctly when the document is viewed on a computer that does not have the necessary fonts installed. (The default value is True)	Boolean
layers_attributes	A string that controls inclusion of PDF layer and PDF object data (attributes) in the export file. • LAYERS_ONLY —Export PDF layers only. • LAYERS_AND_ATTRIBUTES —Export PDF layers and feature attributes. • NONE —None. (The default value is LAYERS_ONLY)	String
georef_info	A Boolean that enables the export of coordinate system information for each data frame into the output PDF file. (The default value is True)	Boolean
jpeg_compression_quality	A number that controls compression quality value when **`image_compression`** is set to ADAPTIVE or JPEG. The valid range is 1 to 100. A **`jpeg_compression_quality`** of 100 provides the best quality images but creates large export files. The recommended range is between 70 and 90. (The default value is 80)	Integer

Code Sample

ExportToPDF example 1

This script opens a map document and exports the page layout to a PDF file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToPDF(mxd, r"C:\Project\Output\Project.pdf")
del mxd
```

ExportToPDF example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToPDF(mxd,
r"C:\Project\Output\ProjectDataFrame.pdf", df,
                          df_export_width=1600,
                          df_export_height=1200)
del mxd
```

ExportToPNG (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Portable Network Graphics (PNG) format.

Discussion

PNG is a raster format designed for use on the Web as an alternative to GIF. It supports 24-bit color and is compressed using a lossless compression. PNG files also have the ability to define a transparent color; part of the image can display as transparent in a Web browser, allowing backgrounds, images, or colors to show through. On most images, PNG can achieve greater compression (and thus smaller file sizes) than GIF. PNGs exported from the data view in ArcMap can be generated with an accompanying world file for use as georeferenced raster data. This format is gaining popularity in the Web design community.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters. Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.

World files are not generated for page layouts; a referenced data frame must be provided or the export will fail.

Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToPNG (map_document, out_png, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {background_color}, {transparent_color}, {interlaced})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_png	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 96)	Integer
world_file	If set to `True`, a georeferenced world file is created. The file contains pixel scale information and real-world coordinate information. (The default value is False)	Boolean
color_mode	This value specifies the number of bits used to describe color. • 24-BIT_TRUE_COLOR —24-bit true color • 8-BIT_PALETTE —8-bit palette • 8-BIT_GRAYSCALE —8-bit grayscale • 1-BIT_MONOCHROME_MASK —1-bit monochrome mask • 1-BIT_MONOCHROME_THRESHOLD —1-bit	String

	monochrome threshold (The default value is 24-BIT_TRUE_COLOR)	
background_color	A defined color is used as a background to the image, or as a mask in the case of monochrome masked exports. (The default value is 255, 255, 255)	String
transparent_color	A defined color to be displayed as transparent in the image.	String
interlaced	If set to True, an interlaced image will be created. An interlaced image displays as a series of scan lines rather than as a whole image at one time. (The default value is False)	Boolean

Code Sample

ExportToPNG example 1

This script opens a map document and exports the page layout to a PNG file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToPNG(mxd, r"C:\Project\Output\Project.png")
del mxd
```

ExportToPNG example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail. Setting **`world_file = True`** generates a georeferenced world file in the same directory as the output file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToPNG(mxd,
r"C:\Project\Output\ProjectDataFrame.png", df,
                          df_export_width=1600,
                          df_export_height=1200,
                          world_file=True)
del mxd
```

ExportToSVG (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Scalable Vector Graphics (SVG) format.

Discussion

SVG is an XML-based file format that has been specifically designed for viewing on the Web. SVG can contain both vector and raster information. This is a good choice for displaying maps on a web page because it is rescalable and more easily edited than raster files. SVG has been gaining in popularity since the World Wide Web Consortium (W3C) selected it as their standard vector Web format. Some Web browsers may require a plug-in to view SVG files; older browsers may not be able to view SVG files at all. SVG supports font embedding, so users who do not have the Esri fonts installed can still view ArcMap SVG exports with proper symbology. ArcMap can also produce compressed SVG files. The file extension changes to *.SVGZ when this option is enabled.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters.
Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.
Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToSVG (map_document, out_svg, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {image_quality}, {compress_document}, {picture_symbol}, {convert_markers}, {embed_fonts})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_svg	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the height of the export image in pixels for a data frame export. **`df_export_height`** is only used when exporting a data frame. Exporting a page layout uses the map document page height instead of **`df_export_height`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in dots per inch (DPI). (The default value is 300)	Integer
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample ratio of 3. • FASTER —An output image quality resample ratio of 4. • FASTEST —An output image quality resample ratio of 5.	String

	(The default value is BEST)	
compress_document	If set to `True`, a compressed export will be created. For SVG, the entire document is compressed and it changes the file extension to *.svgz. (The default value is False)	Boolean
picture_symbol	A string that defines whether picture markers and picture fills will be converted to vector or rasterized on output. • RASTERIZE_BITMAP — Rasterize layers with bitmap markers/fills. • RASTERIZE_PICTURE —Rasterize layers with any picture marker/fill. • VECTORIZE_BITMAP —Vectorize layers with bitmap markers/fills. (The default value is RASTERIZE_BITMAP)	String
convert_markers	A Boolean that controls the coversion of character-based marker symbols to polygons. This allows the symbols to appear correctly if the symbol font is not available or cannot be embedded. However, setting this parameter to `True` disables font embedding for all character-based marker symbols, which can result in a change in their appearance. (The default value is False)	Boolean
embed_fonts	A Boolean that controls the embedding of fonts in export files. Font embedding allows text and character markers to be displayed correctly when the document is viewed on a computer that does not have the necessary fonts installed. (The default value is False)	Boolean

Code Sample

ExportToSVG example 1

This script opens a map document and exports the page layout to an SVG file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToSVG(mxd, r"C:\Project\Output\Project.svg")
del mxd
```

ExportToSVG example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToSVG(mxd,
r"C:\Project\Output\ProjectDataFrame.svg", df,
                          df_export_width=1600,
                          df_export_height=1200)
del mxd
```

ExportToTIFF (arcpy.mapping)

Top

Summary

Exports the page layout or data frame of a map document (`.mxd`) to the Tagged Image File Format (TIFF).

Discussion

TIFF files are the most versatile raster format. TIFFs can store pixel data at several bit depths and can be compressed with either lossy or loss less compression techniques depending on file size and accuracy requirements. They are the best choice for importing into image editing applications across operating systems. However, they cannot be natively viewed by a web browser. ArcMap TIFFs exported from the data view also support georeferencing information in GeoTIFF tags or in a separate world file for use as raster data.

To export a single data frame instead of the entire page layout, pass a DataFrame object to the function's **`data_frame`** parameter. Because data frame exports do not have an associated page to provide height and width information, you must provide this via the **`df_export_width`** and **`df_export_height`** parameters. Controlling graphic quality of the generated image differs for page layout exports versus data frame exports. When exporting a page layout, control image detail by changing the **`resolution`** parameter. When exporting a data frame, keep the **`resolution`** parameter at its default value, and change the **`df_export_width`** and **`df_export_height`** parameters to alter image detail. The height and width parameters directly control the number of pixels generated in the export file and are only used when exporting a data frame. Images with larger numbers of pixels will have higher image detail. For most page layout exports, the default parameter values should generate good results and nice looking export images on the first try. For data frame exports, you may need to experiment with the **`df_export_width`** and **`df_export_height`** values a few times before getting the result you want.

World files are not generated for page layouts; a referenced data frame must be provided or the export will fail.

Refer to the Exporting your map topic in ArcGIS Help for more detailed discussions on exporting maps.

Syntax

ExportToTIFF (map_document, out_tiff, {data_frame}, {df_export_width}, {df_export_height}, {resolution}, {world_file}, {color_mode}, {tiff_compression}, {geoTIFF_tags})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
out_tiff	A string that represents the path and file name for the output export file.	String
data_frame	A variable that references a DataFrame object. Use the string/constant "PAGE_LAYOUT" to export the map document's page layout instead of an individual data frame. (The default value is PAGE_LAYOUT)	Object
df_export_width	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 640)	Integer
df_export_height	A number that defines the width of the export image in pixels for a data frame export. **`df_export_width`** is only used when exporting a data frame. Exporting a page layout uses the map document page width instead of **`df_export_width`**. (The default value is 480)	Integer
resolution	A number that defines the resolution of the export file in DPI (dots per inch). (The default value is 96)	Integer
world_file	If set to `True`, a georeferenced world file is created. The file contains pixel scale information and real-world coordinate information. (The default value is False)	Boolean
color_mode	This value specifies the number of bits used to describe color. • 24-BIT_TRUE_COLOR —24-bit true color. • 8-BIT_PALETTE —8-bit palette. • 8-BIT_GRAYSCALE —8-bit grayscale. • 1-BIT_MONOCHROME_MASK —1-bit monochrome mask.	String

	• 1-BIT_MONOCHROME_THRESHOLD —1-bit monochrome threshold. (The default value is 24-BIT_TRUE_COLOR)	
tiff_compression	This value represents a compression scheme. • DEFLATE —A lossless data compression. • JPEG —JPEG compression. • LZW —Lempel-Ziv-Welch, a lossless data compression. • NONE —Compression is not applied. • PACK_BITS —Pack bits compression. (The default value is LZW)	String
geoTIFF_tags	If set to `True`, georeferencing tags are included in the structure of the TIFF export file. The tags contain pixel scale information and real-world coordinate information. These tags can be read by applications that support GeoTIFF format. (The default value is False)	Boolean

Code Sample

ExportToTIFF example 1

This script opens a map document and exports the page layout to a TIFF file using default values for all options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.ExportToTIFF(mxd, r"C:\Project\Output\Project.tif")
del mxd
```

ExportToTIFF example 2

This script will export a single data frame instead of the entire page layout, similar to exporting from data view in the ArcMap application. The default values for **`df_export_width`** and **`df_export_height`** are 640 and 480. By passing larger values for these parameters, we are able to produce an output image with higher detail. Setting **`geoTIFF_tags=True`** generates georeference information inside of the TIFF file header.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
arcpy.mapping.ExportToTIFF(mxd,
r"C:\Project\Output\ProjectDataFrame.tif", df,
                           df_export_width=1600,
                           df_export_height=1200,
                           geoTIFF_tags=True)
del mxd
```

InsertLayer (arcpy.mapping)

Top

Summary

Provides the ability to insert a layer at a specific location within a data frame or within a group layer in a map document (`.mxd`).

Discussion

`InsertLayer` is a more precise way of positioning a layer into a data frame or a group layer because a reference layer is used to specify the exact location. The layer is either added before or after the reference layer.
If the reference layer references a layer at the root level of a data frame, the inserted layer will be added to the root level of the data frame. If the reference layer references a layer within a group layer, the inserted layer will be added into the group. Because a reference layer is a required parameter, it is not possible to use **`InsertLayer`** to add a layer into an empty data frame or empty group layer. The AddLayer or AddLayerToGroup functions do allow you to add a layer into an empty data frame or group layer, respectively.

The layer that is inserted must reference an already existing layer (keep in mind that a layer can be a group layer as well). The source can either come from a layer file on disk, from within the same map document and data frame, the same map document but different data frame, or even from a completely separate map document.

The way a layer appears in the table of contents (TOC) after it is added depends on the source layer and how it appears. For example, some layers are completely collapsed and do not display their symbol(s) in the TOC. This setting is built into the layer. If a layer is collasped, saved to a layer file, and then added to a map document, the layer will be collasped in the new map document when added via **`InsertLayer`**.

Syntax

InsertLayer (data_frame, reference_layer, insert_layer, {insert_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object into which the new layer will be inserted.	DataFrame
reference_layer	A Layer object representing an existing layer that determines the location where the new layer will be inserted.	Layer
insert_layer	A reference to a Layer object representing the layer to be inserted.	Layer
insert_position	A constant that determines the placement of the added layer relative to the reference layer. • AFTER —Inserts the new layer after or below the reference layer • BEFORE —Inserts the new layer before or above the reference layer (The default value is BEFORE)	String

Code Sample

InsertLayer example 1:

The follwing script will insert a new layer from a layer (`.lyr`) file on disk and place it before a layer called `Lakes` which is in a data frame called **`County Maps`**.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
refLayer = arcpy.mapping.ListLayers(mxd, "Lakes", df)[0]
insertLayer = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.InsertLayer(df, refLayer, insertLayer, "BEFORE")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, insertLayer
```

InsertLayer example 2:

The following script will insert a layer called `Rivers`, from another, independant map document, above a layer called `Lakes` in a data frame called `County Maps`.

```
import arcpy

#Reference layer in secondary map document
mxd2 = arcpy.mapping.MapDocument(r"C:\Project\ProjectTemplate.mxd")
df2 = arcpy.mapping.ListDataFrames(mxd2, "Layers")[0]
insertLayer = arcpy.mapping.ListLayers(mxd2, "Rivers", df2)[0]

#Insert layer into primary map document
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
refLayer = arcpy.mapping.ListLayers(mxd, "Lakes", df)[0]
arcpy.mapping.InsertLayer(df, refLayer, insertLayer, "BEFORE")

#Save to a new map document and clear variable references
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, mxd2, insertLayer
```

Layer (arcpy.mapping)

Top

Summary

References a layer (`.lyr`) file stored on disk.

Discussion

For a more complete discussion, refer to the Layer Class help topic.

Syntax

Layer (lyr_file_path)

Parameter	Explanation	Data Type
lyr_file_path	A string that includes the full path and file name of an existing layer (`.lyr`) file.	String

Return Value

Data Type	Explanation
Layer	The Layer object provides access to a layer file's layer properties and provides different options for saving a layer (`.lyr`) file to disk.

ListBookmarks (arcpy.mapping)

Top

Summary

Returns a Python list of named tuples that provide access to each spatial bookmark's name and extent.

Discussion

`ListBookmarks` always returns a Python list of named tuples. Each tuple provides the bookmark's name as a string and the bookmark's extent as an Extent object. In order to return a specific tuple, an index value must be used on the list (for example, `bkmk = arcpy.mapping.ListBookmarks(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each tuple in the list (for example, `for bkmk in arcpy.mapping.ListBookmarks(mxd):`).
Wildcards are used on the **`name`** property and are not case sensitive. A wildcard string of `"so*"` will return a spatial bookmark with the name `South East`. Wildcards can be skipped in the scripting syntax by passing an empty string (""), an asterisk (*), or entering `wildcard=None`, or nothing at all if it is the last optional parameter in the syntax.

Avoid having spatial bookmarks in a single data frame that have the same name because the name property is really the only practical way of identifying a spatial extent. Bookmarks can have the same name if they are in different data frames.

Syntax

ListBookmarks (map_document, {wildcard}, {data_frame})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. It is used to filter spatial bookmark names. (The default value is None)	String
data_frame	A variable that references a DataFrame object. This is used to find a spatial bookmark associated with a specific data frame. (The default value is None)	DataFrame

Return Value

Data Type	Explanation
List	A Python list of named tuples. • extent —A GP Extent object • name —A string that represents the name of a spatial bookmark

Code Sample

ListBookmarks example 1

This script will print the name of each spatial bookmark in a data frame named `Transportation`. The wildcard parameter is skipped using a blank string.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
for bkmk in arcpy.mapping.ListBookmarks(mxd, "", df):
    print bkmk.name
del mxd
```

ListBookmarks example 2

Similar to example 1, the following script will loop through each bookmark in the `Transportation` data frame, set the data frame extent, and export the data frame to a JPEG file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
for bkmk in arcpy.mapping.ListBookmarks(mxd, data_frame=df):
    df.extent = bkmk.extent
    outFile = r"C:\Project\Output\\" + bkmk.name + ".jpg"
    arcpy.mapping.ExportToJPEG(mxd, outFile, df)
del mxd
```

ListBookmarks example 3

This sample will convert each bookmark in a map document to a feature. The resulting feature class can be used for a variety of workflows, including use as a Data Driven Pages index layer for creating map books.

```
import arcpy, os

# The map with the bookmarks
mxd = arcpy.mapping.MapDocument(r"C:\Project\Counties.mxd")

# The output feature class to be created -
# This feature class will store the bookmarks as features
outFC = r'C:\Project\Counties.gdb\Bookmarks'

# A template feature class that contains the attribute schema
# Including a "Name" field to store the bookmark name
template = r'C:\Project\Counties.gdb\Template'

if arcpy.Exists(outFC):
    arcpy.Delete_management(outFC)
arcpy.CreateFeatureclass_management(os.path.dirname(outFC),
                                    os.path.basename(outFC),
                                    "POLYGON", template,
                                    spatial_reference=template)

cur = arcpy.da.InsertCursor(outFC, ["SHAPE@", "Name"])
array = arcpy.Array()
for bkmk in arcpy.mapping.ListBookmarks(mxd):
    array.add(arcpy.Point(bkmk.extent.XMin, bkmk.extent.YMin))
    array.add(arcpy.Point(bkmk.extent.XMin, bkmk.extent.YMax))
    array.add(arcpy.Point(bkmk.extent.XMax, bkmk.extent.YMax))
    array.add(arcpy.Point(bkmk.extent.XMax, bkmk.extent.YMin))
    # To close the polygon, add the first point again
    array.add(arcpy.Point(bkmk.extent.XMin, bkmk.extent.YMin))
    cur.insertRow([arcpy.Polygon(array), bkmk.name])
    array.removeAll()
```

ListBrokenDataSources (arcpy.mapping)

Top

Summary

Returns a Python list of Layer objects within a map document (.mxd) or layer (.lyr) file that have broken connections to their original source data.

Discussion

ListBrokenDataSources always returns a Python list object even if only one broken layer is returned. In order to return a single layer object, an index value must be used on the list (e.g., `brkLyr = arcpy.mapping.ListBrokenDataSources(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in the list (e.g., `for brkLyr in arcpy.mapping.ListBrokenDataSources(mxd):`).

Some layers within a map document or layer file may be password protected because the user and password information is not saved within the layer file or map document. Map documents that contain these layers typically prompt the user to enter the appropriate information while the document is opening. The arcpy.mapping scripting environment will, by default, supress these dialogs during execution but that means that the layers will be treated as though they have broken data sources. In otherwords, secured layers will not be rendered in any output. If it is necessary for these layers to render appropriately then there are a couple of options. First, save the username and password information with the layers. Second, the CreateArcSDEConnectionFile geoprocessing function allows you to create a connection that is persisted in memory. If this command is used prior to opening a map document (.mxd) with the MapDocument function or a layer file with the Layer function, then SDE layers will render and not appear as broken. Currently, there is not an alternative for secured web services. See the Layer help for a code example.

To learn more about automating the repair of broken layers, refer to: Updating and Fixing Data Sources.

Syntax

ListBrokenDataSources (map_document_or_layer)

Parameter	Explanation	Data Type
map_document_or_layer	A variable that references a MapDocument or Layer object.	Object

Return Value

Data Type	Explanation
Layer	A Python list of Layer objects.

Code Sample

ListBrokenDataSources example:

This script will search for broken data sources in all map documents that exist in a single folder. A report with map document names and broken sources will be printed.

```
import arcpy, os
path = r"C:\Project"
for fileName in os.listdir(path):
    fullPath = os.path.join(path, fileName)
    if os.path.isfile(fullPath):
        basename, extension = os.path.splitext(fullPath)
        if extension == ".mxd":
            mxd = arcpy.mapping.MapDocument(fullPath)
            print "MXD: " + fileName
            brknList = arcpy.mapping.ListBrokenDataSources(mxd)
            for brknItem in brknList:
                print "\t" + brknItem.name
del mxd
```

ListDataFrames (arcpy.mapping)

Top

Summary

Returns a Python list of DataFrame objects that exist within a single map document (`.mxd`).

Discussion

`ListDataFrames` always returns a Python list object even if only one data frame is returned. In order to return a **`DataFrame`** object, an index value must be used on the list (e.g., `df = arcpy.mapping.ListDataFrames(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in a list (e.g., `for df in arcpy.mapping.ListDataFrames(mxd):`).

Wildcards are not case sensitive. A wildcard string of `"la*"` will return a data frame with a name `Layers`.

It is possible that there might be data frames in a map document that have the same name. If that is the case, then other properties may need to be used to isolate a specific data frame. Properties such as a data frame's **`credits`** or **`description`** could be used to do this. It is ideal that all data frames be uniquely named.

Syntax

ListDataFrames (map_document, {wildcard})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. (The default value is None)	String

Return Value

Data Type	Explanation
DataFrame	A Python list of DataFrame objects.

Code Sample

DataFrame example:

This script will search for a data frame with the name `Transportation` and set the scale and rotation to the appropriate values.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd, "t*"):
    if df.name.lower == "transportation":
        df.scale = 24000
        df.rotation = 5.5
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

ListLayers (arcpy.mapping)

Top

Summary

Returns a Python list of Layer objects that exist within a map document (.mxd), a data frame within a map document, or layers within a layer (.lyr) file.

Discussion

ListLayers always returns a Python list object even if only one layer is returned. In order to return a **Layer** object, an index value must be used on the list (e.g., `lyr = arcpy.mapping.ListLayers(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in a list (e.g., `for lyr in arcpy.mapping.ListLayers(mxd):`).

When working with layer files, the **data_frame** parameter should not be used because layer files don't support data frames; if it is, it will be ignored.

The Layer() function is for referencing layer (.lyr) files stored on disk.

Group layers are treated just like layers. The index values are simply generated from top to bottom as they appear in the table of contents or the way they would appear in a layer file. The same applies if a group layer is within another group layer. A map document with a single group layer with three layers within it will return a Python list of four layer objects, the group layer being the first. One way of determining if a layer is inside a group layer is to interrogate the **longName** property. A layer's **longName** will include the group layer name as part of the name.

Wildcards are used on the **name** property and are not case sensitive. A wildcard string of `"so*"` will return a layer with the name `Soils`. Wildcards can be skipped in the scripting syntax simply by passing an empty string (""), an asterisk (*), or entering `wildcard=None`, or nothing at all if it is the last optional parameter in the syntax.

It is possible that there might be layers in a map document or layer file that have the same name. If that is the case, then other properties may need to be used to isolate a specific layer. Properties such as a layer's **description** or a layer's **dataSource** could be used to do this. It is ideal that all layers in a map document or at least in a layer be uniquely named.

Syntax

ListLayers (map_document_or_layer, {wildcard}, {data_frame})

Parameter	Explanation	Data Type
map_document_or_layer	A variable that references a MapDocument or Layer object.	Object
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. (The default value is None)	String
data_frame	A variable that references a DataFrame object. (The default value is None)	DataFrame

Return Value

Data Type	Explanation
Layer	A Python list of Layer objects.

Code Sample

ListLayers example 1:

This script will print the name of the first layer in a data frame named `Traffic Analysis`. The wildcard parameter is skipped using a blank string.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Traffic Analysis")[0]
print arcpy.mapping.ListLayers(mxd, "", df)[0].name
del mxd
```

ListLayers example 2:

The following script will find a layer called `Lakes` in a data frame named `County Maps`, turn the layer on (to be visible) and set the transparency to 50%. An issue is that there happens to be two layers with the same name in the same data frame so the **`description`** property is used to further isolate the layer of interest. Ideally, all layers would have a unique name but that isn't always the case so other properties have to be used to isolate the layer of interest. In this example, the **`description`** is used.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
for lyr in arcpy.mapping.ListLayers(mxd, "Lakes", df):
    if lyr.description == "USGS Lakes":
        lyr.visible = True
        lyr.transparency = 50
mxd.save()
del mxd
```

ListLayoutElements (arcpy.mapping)

Top

Summary

Returns a Python list of layout elements that exist within a map document (`.mxd`) layout.

Discussion

`ListLayoutElements` always returns a Python list object even if only one page element is returned. In order to return an element object, an index value must be used on the list (e.g., `elm = arcpy.mapping.ListLayoutElements(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in the list (e.g., `for elm in arcpy.mapping.ListLayoutElements(mxd):`).

`ListLayoutElements` only returns elements from a page layout and not map annotation elements that may exist within a data frame.

Each page element has a **`name`** property that can be set within the element properties dialog box within ArcMap (located on the **Size and Position** tab). It is the map document author's responsibility to ensure each page element is given a unique name so that elements can be uniquely identified. If two elements have the same name, there is no way for certain to ensure it is the element you want to reference.

`ListLayoutElements` will also return the elements within a group element into a flattened list. This makes it possible to easily search and replace text strings, for example, without having to navigate through a group element structure.

The **`element_type`** parameter can be skipped simply by passing an empty string ("") or entering `element_type=None`.

Wildcards are used on the **`name`** property and are not case sensitive. A wildcard string of `"*title"` will return a page element with a name `Main Title`.

Wildcards can be skipped in the scripting syntax simply by passing an empty string (""), an asterisk (*), or entering `wildcard=None`, or nothing at all if it is the last optional parameter in the syntax.

Refer to the individual element objects for more information: DataFrame, GraphicElement, LegendElement, MapsurroundElement, PictureElement, and TextElement.

Syntax

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

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
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 (The default value is None)	String
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. (The default value is None)	String

Return Value

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

Code Sample

ListLayoutElements example 1:

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`.

```
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.

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

ListMapServices (arcpy.mapping)

Top

Summary

Legacy:

This method has been deprecated starting at ArcGIS 10.1 and will return a runtime error. Please consult the ArcGIS documentation for the usage of the new ArcGIS for Server Administrator API.
Lists the names of map services for a designated ArcGIS for Server.

Discussion

This method has been deprecated starting at ArcGIS 10.1 and will return a runtime error.

Starting at version 10.1, ArcGIS for Server has a new architecture which may require you to adjust how you work with the server. See the following help topics for more information: What to expect when migrating to ArcGIS 10.2 for Server and Migration to ArcGIS 10.2 for Server.
You can list Map Services using the ArcGIS for Server Administrator API which is available through the ArcGIS Site Directory. The default URL of the ArcGIS for Server Site Directory is as follows:
`http://<server name>:6080/arcgis/admin`

Note:

The URL of the Site Directory may vary if you have configured ArcGIS for Server Web Adaptor to work with your site. For example, the inclusion of the port number, 6080, may not be required. Check your web adaptor configuration to obtain the correct URL.
An example of using Python and the ArcGIS for Server Administrator API to list Map Services is below:

Note:

A code sample to generate an ArcGIS for Server token can be found here: DeleteMapService.

```
import json, urllib2

server = "<server>"
port = "6080"
token = '<token string>'
baseUrl = "http://{}:{}/arcgis/admin/services".format(server, port)
  catalog = json.load(urllib2.urlopen(baseUrl + "/" + "?f=json&token="
+ token))
  print 'Root'
  if "error" in catalog: return
  services = catalog['services']
  for service in services:
    response = json.load(urllib2.urlopen(baseUrl + '/' +
service['serviceName'] + '/' + service['type'] + "?f=json&token=" +
token))
    print '  %s %s (%s)' % (service['serviceName'], service['type'],
'ERROR' if "error" in response else 'SUCCESS')
  folders = catalog['folders']
  for folderName in folders:
    catalog = json.load(urllib2.urlopen(baseUrl + "/" + folderName +
"?f=json&token=" + token))
    print folderName
    if "error" in catalog: return
    services = catalog['services']
    for service in services:
      response = json.load(urllib2.urlopen(baseUrl + '/' +
service['serviceName'] + '/' + service['type'] + "?f=json&token=" +
token))
      print '  %s %s (%s)' % (service['serviceName'], service['type'],
'ERROR' if "error" in response else 'SUCCESS')

getCatalog(token)
```

You can also list Map Services using the ArcGIS for Server REST API which is available through the ArcGIS Services Directory. The default URL of the ArcGIS for Server Services Directory is as follows:
`http://<server name>:6080/arcgis/rest/services`
To get started using the ArcGIS for Server Services Directory and REST API, see the help within the Services Directory.

An example of using Python and the ArcGIS for Server REST API to list Map Services is below:

```
import json, urllib2

server = "<server>"
port = "6080"
baseUrl = "http://{}:{}/arcgis/rest/services".format(server, port)

def getCatalog():
  catalog = json.load(urllib2.urlopen(baseUrl + "/" + "?f=json"))
  print 'ROOT'
  if "error" in catalog: return
  services = catalog['services']
  for service in services:
    response = json.load(urllib2.urlopen(baseUrl + '/' +
service['name'] + '/' + service['type'] + "?f=json"))
    print '  %s %s (%s)' % (service['name'], service['type'], 'ERROR'
if "error" in response else 'SUCCESS')
  folders = catalog['folders']
  for folderName in folders:
    catalog = json.load(urllib2.urlopen(baseUrl + "/" + folderName +
"?f=json"))
    print folderName
    if "error" in catalog: return
    services = catalog['services']
    for service in services:
      response = json.load(urllib2.urlopen(baseUrl + '/' +
service['name'] + '/' + service['type'] + "?f=json"))
      print '  %s %s (%s)' % (service['name'], service['type'],
'ERROR' if "error" in response else 'SUCCESS')

getCatalog()
```

Syntax

ListMapServices (connection_url_or_name, server, {connection_username}, {connection_password}, {connection_domain})

Parameter	Explanation	Data Type
connection_url_or_name	A string that represents the URL of the ArcGIS for Server to which you want to get a list of services.	String
server	A string that represents the ArcGIS for Server host name.	String
connection_username	A string that represents a user name used to connect to the ArcGIS for Server. In order to get a list of map services this user name should be a member of the ArcGIS for Server admin group. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_password	A string that represents a password used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_domain	A string that represents a domain name used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String

Return Value

Data Type	Explanation
List	A Python list of map services.

ListPrinterNames (arcpy.mapping)

Top

Summary

Returns a Python list of available printers on the local computer.

Discussion

ListPrinterNames always returns a list object even if only one printer name is returned. In order to return a single printer, an index value must be used on the list (e.g., `printer = arcpy.mapping.ListPrinterNames()[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in the list (e.g., `for printer in arcpy.mapping.ListPrinterNames():`).

ListPrinterNames is an easy way to identify the names of the printers currently available to the local computer. These string values can then be used as input parameters with the PrintMap() function or the **printPages** method on the DataDrivenPages object.

Note:

Driver based printing is not supported on ArcGIS for Server. However, non-driver based printing is supported in web applications. For more information, see Printing in web applications.

Syntax

ListPrinterNames ()

Return Value

Data Type	Explanation
String	A Python list of printer names.

Code Sample

ListPrinterNames example:

This script will print the names of the availble printers.

```
import arcpy
for printerName in arcpy.mapping.ListPrinterNames():
    print printerName
```

ListStyleItems (arcpy.mapping)

Top

Summary

Returns a Python list of StyleItem objects. A referenced legend item from a style file (`.style` or `.ServerStyle`) can then be used to update already existing legend items in a layout.

Discussion

In ArcGIS for Desktop, style items are stored in a `.style` file. In ArcGIS for Server, style items are stored in a `.ServerStyle` file. Style items in a `.style` file can be viewed and managed in the ***Style manager*** window. Style files are organized into different subfolders with unique names, for example, Marker Symbols or Legend Items. Currently, the only style items that can be used with other arcpy.mapping methods are legend items. Legend items define how a layer appears in a legend. For example, they store information such as legend arrangement, layer name symbol, and default override patch.

Style items are exposed to the arcpy.mapping API so that users can control the appearance of legend item style items. A common requirement is to be able to control the default font size or font type for newly added legend items. To accomplish this, you must first author a custom legend item and then reference it with the **`ListStyleItems`** function. Next, you would reference the appropriate Layer that appears in a Legend, and then use its **`updateItem`** method. An example of this workflow is provided as a code sample below.

Syntax

ListStyleItems (style_file_path, style_folder_name, {wildcard})

Parameter	Explanation	Data Type
style_file_path	A full path to an existing style (`.style`) or server style (`.ServerStyle`) file. There are two additional shortcuts that don't require a full path. First, type the name of the ArcGIS system style file, for example,`"ESRI.style"` or `"ESRI.ServerStyle"` or `"Transportation.style"`. The function will automatically search for the style in the appropriate ArcGIS installation style folder. Second, with ArcGIS for Desktop installations, you can use the keyword`"USER_STYLE"`. This will automatically search the local user profile rather than requiring the full path. If the style file does not exist in either of these two known system locations, then the full path including the file extension must be provided, for example,`"C:/Project/CustomStyles.style"`.	String
style_folder_name	The name of the style folder in the style file the way it appears in the ***Style manager*** window. Currently, only `Legend Items` can be used with other arcpy.mapping methods.	String
wildcard	A combination of asterisks (*) and characters can be used to help limit the results based on the style item name property. (The default value is None)	String

Code Sample

ListStyleItems example

The following script uses the workflow outlined above and updates a legend's legend item style item. A layer is added to the first data frame in the map document and the legend item style item will be updated with a custom legend item style item named `NewDefaultLegendStyle`. The custom `.style` file is saved in the user's profile location.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd)[0]
lyrFile = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.AddLayer(df, lyrFile, "TOP")
styleItem = arcpy.mapping.ListStyleItems("USER_STYLE", "Legend Items",
"NewDefaultLegendStyle")[0]
lyr = arcpy.mapping.ListLayers(mxd, 'Rivers', df)[0]
legend = arcpy.mapping.ListLayoutElements(mxd, "LEGEND_ELEMENT")[0]
legend.updateItem(lyr, styleItem)
del mxd
```

ListTableViews (arcpy.mapping)

Top

Summary

Returns a Python list of TableView objects that exist within a map document (.mxd).

Discussion

ListTableViews always returns a list object even if only one table is returned. In order to return a **TableView** object, an index value must be used on the list (e.g., `aTable = arcpy.mapping.ListTableViews(mxd)[0]`). `For` loops on a list provide an easy mechanism to iterate through each item in the list (e.g., `for aTable in arcpy.mapping.ListTableViews(mxd):`).
Wildcards are used on the **name** property and are not case sensitive. A wildcard string of `"so*"` will return a layer with a name `Soils`. Wildcards can be skipped in the scripting syntax simply by passing an empty string (""), an asterisk (*), or entering `wildcard=None`, or nothing at all if it is the last optional parameter in the syntax.
It is possible that there might be tables in a map document that have the same name. If that is the case, then other properties may need to be used to isolate a specific layer. Properties such as a tables's **datasource** or**definitionQuery** could be used to do this. It is ideal that all tables in a map document be uniquely named.

Syntax

ListTableViews (map_document, {wildcard}, {data_frame})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. (The default value is None)	String
data_frame	A variable that references a DataFrame object. (The default value is None)	DataFrame

Return Value

Data Type	Explanation
TableView	A Python list of TableView objects.

Code Sample

ListTableViews example

The following script finds a table called `TrafficAccidents` in a data frame named `Transportation` and sets a definition query.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"c:\project\project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Transportation")[0]
table = arcpy.mapping.ListTableViews(mxd, "TrafficAccidents", df)[0]
table.definitionQuery = "[Accidents] > 5"
mxd.save()
del mxd
```

MapDocument (arcpy.mapping)

Top

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)

Parameter	Explanation	Data Type
mxd_path	A string that includes the full path and file name of an existing map document (.mxd) or a string that contains the keyword CURRENT.	String

Return Value

Data Type	Explanation
MapDocument	The MapDocument object provides access to map document properties and methods. A reference to this object is essential for most map scripting operations.

Code Sample

MapDocument example 1

The following script creates a separate MXD file for each data frame in a map document. The output map documents will be saved in data view mode so when each map document is opened, the corresponding data frame will be active data frame. The script also sets the title property of each output map document. Because this script uses a system path to the map document, it can be executed outside an ArcMap application. Note: Python strings cannot end with a backslash, even when the string is preceded by an r. You must use a double backslash. This becomes important when appending dynamic file names to a folder path.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    mxd.activeView = df.name
    mxd.title = df.name
    mxd.saveACopy(r"C:\Project\Output\\" + df.name + ".mxd")
del mxd
```

MapDocument example 2

The following script demonstrates how the CURRENT keyword can be used within the Python window. This sample will update the first data frame's name and refresh the table of contents so the change can be see in the application. Paste the following code into the Python window within a new ArcMap document.

```
mxd = arcpy.mapping.MapDocument("CURRENT")
arcpy.mapping.ListDataFrames(mxd)[0].name = "New Data Frame Name"
arcpy.RefreshTOC()
del mxd
```

When pasted into the interactive window it will appear as follows. The three dots to the left of the code block indicate that the lines are a single block of code that will be executed together. You must press the Enter key to execute these lines.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
... arcpy.mapping.ListDataFrames(mxd)[0].name = "New Data Frame Name"
... arcpy.RefreshTOC()
... del mxd
...
```

MapDocument example 3

The following is another simple script that demonstrates the use of the `CURRENT` keyword within the Python window. Each layer name will be printed to the Python window. Loops are also possible, provided that you maintain the correct indentation. Similar to the example above, paste the code below into the Python window.

```
mxd = arcpy.mapping.MapDocument("CURRENT")
for lyr in arcpy.mapping.ListLayers(mxd):
    print lyr.name
del mxd
```

When pasted into the interactive window it will appear as follows. Again, press the Enter key to execute the lines.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
... for lyr in arcpy.mapping.ListLayers(mxd):
...     print lyr.name
... del mxd
...
```

MapDocument example 4

The following script will allow secured layers to render correctly by creating an SDE connection in memory before opening a map document that requires password information. This script simply defines the connection information and exports the map document to a PDF file. It is good practice to delete this reference from memory before the script closes.

```
import arcpy, os

#Remove temporary connection file if it already exists
sdeFile = r"C:\Project\Output\TempSDEConnectionFile.sde"
if os.path.exists(sdeFile):
    os.remove(sdeFile)

#Create temporary connection file in memory
arcpy.CreateArcSDEConnectionFile_management(r"C:\Project\Output",
"TempConnection", "myServerName", "5151", "myDatabase",
"DATABASE_AUTH", "myUserName", "myPassword", "SAVE_USERNAME",
"myUser.DEFAULT", "SAVE_VERSION")

#Export a map document to verify that secured layers are present
mxd = arcpy.mapping.MapDocument(r"C:\Project\SDEdata.mxd")
arcpy.mapping.ExportToPDF(mxd, r"C:\Project\output\SDEdata.pdf")

os.remove(sdeFile)
del mxd
```

MoveLayer (arcpy.mapping)

Top

Summary

Provides the ability to move a layer to a specific location within a data frame or group layer in a map document (`.mxd`).

Discussion

`MoveLayer` will move a layer within a data frame and also within group layers in the same data frame. The **`moved_layer`** and **`reference_layer`** must reside within the same data frame. A layer cannot be moved from one data frame to a different data frame even within the same map document. Add_Layer, Insert_Layer, and RemoveLayer functions can be used to accomplish this requirement.

Syntax

MoveLayer (data_frame, reference_layer, move_layer, {insert_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object within which the layer will be moved.	DataFrame
reference_layer	A reference to a Layer object representing an existing layer that determines the location in relation to where the layer will be moved.	Layer
move_layer	A reference to a Layer object representing the layer to be moved.	Layer
insert_position	A constant that determines the placement of the moved layer relative to the referenced layer. • AFTER —Inserts the new layer after or below the referenced layer • BEFORE —Inserts the new layer before or above the referenced layer (The default value is BEFORE)	String

Code Sample

MoveLayer example:

The following script will move a layer called `Rivers` above a reference layer called `Lakes`. Both layers are in a data frame called `County Maps`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
for lyr in arcpy.mapping.ListLayers(mxd, "", df):
    if lyr.name.lower() == "rivers":
        moveLayer = lyr
    if lyr.name.lower() == "lakes":
        refLayer = lyr
arcpy.mapping.MoveLayer(df, refLayer, moveLayer, "BEFORE")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

PDFDocumentCreate (arcpy.mapping)

Top

Summary

Creates an empty PDFDocument object in memory.

Discussion

The **PDFDocumentCreate** function receives a path to determine the save location and file name where a new PDF file will be created. However, no PDF file will be created until subsequent steps are performed to insert or append pages and save the PDF file. **PDFDocumentCreate** will return a PDFDocument object that your script should then manipulate and save. A common scenario for using this function is the creation of a PDF map book. The steps typically involve exporting a number of separate PDF files from map documents, creating a new PDFDocument object, appending content from the exported PDF files and other documents, and saving the final PDF map book.

Please note that it is not possible to create blank PDF files, nor does the **PDFDocumentCreate** function add any blank pages to the document contents. For the **saveAndClose** method to successfully create a file, content must be added to the PDFDocument object using the **appendPages** or **insertPages** methods.

For more discussion on how to create map books, see the Building Map Books with ArcGIS help topic.

Syntax

PDFDocumentCreate (pdf_path)

Parameter	Explanation	Data Type
pdf_path	A string that specifies the path and file name for the resulting PDF file when the **saveAndClose** method is called.	String

Code Sample

PDFDocumentCreate example

This script will create a new PDF document, append the contents of three separate PDF documents, and save the resulting PDF file.

```
import arcpy, os

#Set file name and remove if it already exists
pdfPath = r"C:\Project\ParcelAtlasMapBook.pdf"
if os.path.exists(pdfPath):
    os.remove(pdfPath)

#Create the file and append pages
pdfDoc = arcpy.mapping.PDFDocumentCreate(pdfPath)
pdfDoc.appendPages(r"C:\Project\Title.pdf")
pdfDoc.appendPages(r"C:\Project\ParcelAtlas.pdf")
pdfDoc.appendPages(r"C:\Project\ContactInfo.pdf")

#Commit changes and delete variable reference
pdfDoc.saveAndClose()
del pdfDoc
```

PDFDocumentOpen (arcpy.mapping)

Top

Summary

Returns a PDFDocument object (the contents of the object come from a PDF file on disk, and subsequent operations followed by a call to **saveAndClose** will modify the original PDF file).

Discussion

Use the **PDFDocumentOpen** function to get a reference to an existing PDF file and modify its contents. For your changes to be committed to disk, be sure to call **saveAndClose** after performing the **PDFDocument** operations.

Syntax

PDFDocumentOpen (pdf_path, {user_password}, {master_password})

Parameter	Explanation	Data Type
pdf_path	A string that specifies the path and file name of the PDF file to open.	String
user_password	A string that specifies the user password. User passwords are typically used to restrict opening and specific master-defined operations for a PDF file.	String
master_password	A string that specifies the master password. Master passwords are typically used to restrict setting of user permissions for a PDF file.	String

Code Sample

PDFDocumentOpen example

The following script modifies the PDF document metadata properties and sets the style in which the document will open.

```
import arcpy
pdfDoc =
arcpy.mapping.PDFDocumentOpen(r"C:\Project\ParcelAtlasMapBook.pdf")
pdfDoc.updateDocProperties(pdf_title="Atlas Map",
                           pdf_author="Esri",
                           pdf_subject="Map Book",
                           pdf_keywords="Atlas; Map Books",
                           pdf_open_view="USE_THUMBS",
                           pdf_layout="SINGLE_PAGE")
pdfDoc.saveAndClose()
del pdfDoc
```

PrintMap (arcpy.mapping)

Top

Summary

Prints a specific data frame or a map document (`.mxd`) layout to a printer or file

Discussion

`PrintMap` provides the ability to print a specific data frame or a map document layout to a system printer or a print file. If a printer name is not provided, **`PrintMap`** will use the printer that is saved with the map document or will use the default system printer if the map document does not have a printer saved.

An easy way to determine the printers that are available to the local computer is to use the ListPrinterNames function.

If you want to print using **`ArcPress`**, you must set up the printer properties and save the printer with the map document.

Note:

Driver based printing is not supported on ArcGIS for Server. However, non-driver based printing is supported in web applications. For more information, see Printing in web applications.

Syntax

PrintMap (map_document, {printer_name}, {data_frame}, {out_print_file}, {image_quality})

Parameter	Explanation	Data Type
map_document	A variable that references a MapDocument object.	MapDocument
printer_name	A string that represents the name of a printer on the local computer. (The default value is None)	String
data_frame	A variable that references a DataFrame object. (The default value is PAGE_LAYOUT)	DataFrame
out_print_file	A path that includes the name of an output print file. The format created is dependent on the printer. If you are using a postscript printer, the format will be postscript, and it is recommended that a `.ps` extension be provided; if you are using a Windows printer, use a `.prn` extension. (The default value is None)	String
image_quality	A string that defines output image quality, the draw resolution of map layers that draw as rasters. Using the default value of "None" will cause the function to use the image quality saved in the map document. • BEST —An output image quality resample ratio of 1. • BETTER —An output image quality resample ratio of 2. • NORMAL —An output image quality resample ratio of 3. • FASTER —An output image quality resample ratio of 4. • FASTEST —An output image quality resample ratio of 5. (The default value is None)	String

Code Sample

PrintMap example 1

The following script prints a map using the default printer options.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
arcpy.mapping.PrintMap(mxd)
```

PrintMap example 2

The following script prints the first data frame within a map document using a specified printer name.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd)[0]
arcpy.mapping.PrintMap(mxd, r"\\olyfile\OLYCanon", df)
```

PublishMSDToServer (arcpy.mapping)

Top

Summary

Legacy:

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. Please use the Upload Service Definition Geoprocessing tool instead.

Publishes an existing map service definition (MSD) file to a designated ArcGIS for Server.

Discussion

Starting at ArcGIS 10.1, Map Server Definition (`.msd`) files have been replaced with Service Definition Draft (`.sddraft`) and Service Definition (`.sd`) files. See the following help topics for more information: What to expect when migrating to ArcGIS 10.2 for Server and Migration to ArcGIS 10.2 for Server.

Automating the publishing of a Map Document to a GIS Server using ArcPy is a four-part process. The first step is to run the CreateMapSDDraft function. The output created from CreateMapSDDraft is a Service Definition Draft (`.sddraft`) file. A Service Definition Draft is the combination of a Map Document, information about the server, and a set of service properties. The output Service Definition Draft file can then be analysed for suitability and potential performance issues using the AnalyzeForSD function. The Service Definition Draft can then be converted to a fully consolidated Service Definition (`.sd`) file using the Stage Service Geoprocessing tool. Staging compiles all the necessary information needed to successfully publish the GIS resource. If you have chosen to copy data to the server, the data will be added when the Service Definition Draft is staged. Finally, the Service Definition file can be uploaded and published as a GIS service to a specified GIS server using the Upload Service Definition Geoprocessing tool. This step takes the Service Definition file, copies it onto the server, extracts required information, and publishes the GIS resource. For more information, see overview of the Publishing toolset.

Syntax

PublishMSDToServer (msd_path, connection_url_or_name, server, service_name, {folder_name}, {service_capabilities}, {connection_username}, {connection_password}, {connection_domain})

Parameter	Explanation	Data Type
msd_path	A string that represents the path and name of an existing MXD document you want to serve.	String
connection_url_or_name	A string that represents the URL of the ArcGIS for Server to which you want to publish the MSD.	String
server	A string that represents the ArcGIS for Server host name to which you want to publish the MSD.	String
service_name	A string that represents the name of the service. This is the name people will see and use to identify the service. The name can only contain alphanumeric characters and underscores. No spaces or special characters are allowed. The name cannot be more than 120 characters in length.	String
folder_name	A string that represents a folder name to which you want to publish the MSD. If the folder does not currently exist, it will be created. The default folder is the server root level.	String
service_capabilities [service_capabilities,...]	A list of strings that represents additional capabilities in addition to the map service capability. • MAPPING —The default ArcGIS for Server capability • KML —Keyhole Markup Language • WCS —Web Coverage Service • WFS —Web Feature Service • WMS —Web Map Service (The default value is MAPPING)	String
connection_username	A string that represents a user name used to connect to the ArcGIS for Server. To publish a map service, this user name should be a member of the ArcGIS for Server admin group. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_password	A string that represents a password used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String
connection_domain	A string that represents a domain name used to connect to the ArcGIS for Server. This variable is only necessary when connecting to a UNIX/Linux ArcGIS for Server. (The default value is None)	String

RemoveLayer (arcpy.mapping)

Top

Summary

Provides the ability to remove a layer within a data frame in a map document (`.mxd`).

Discussion

`RemoveLayer` will remove a single layer or group layer from a specific data frame. If there is more than one layer that meets the criteria, then only the first layer will be removed unless the script iterates through each layer in a returned list.

Syntax

RemoveLayer (data_frame, remove_layer)

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be removed.	DataFrame
remove_layer	A reference to a Layer object representing the layer to be removed.	Layer

Code Sample

RemoveLayer example:

The following script will remove all layers with the name `Rivers` from a map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    for lyr in arcpy.mapping.ListLayers(mxd, "", df):
        if lyr.name.lower() == "rivers":
            arcpy.mapping.RemoveLayer(df, lyr)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

RemoveLayer example 2

The following script will remove a layer from a data frame called `County Maps` based on it's name and data source because other layers exist within the data frame with the same name but different source.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
for lyr in arcpy.mapping.ListLayers(mxd, "", df):
    if lyr.name.lower() == "lakes":
        if lyr.dataSource == r"C:\Project\Data\Data.mdb\NE_Lakes":
            arcpy.mapping.RemoveLayer(df, lyr)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

RemoveTableView (arcpy.mapping)

Top

Summary

Provides the ability to remove a table within a data frame in a map document (`.mxd`).

Discussion

`RemoveTableView` will remove a single table from a specific data frame in a map document.

Syntax

RemoveTableView (data_frame, remove_table)

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be removed.	DataFrame
remove_table	A reference to a Layer object representing the layer to be removed.	TableView

Code Sample

RemoveTableView example

The following script will remove all tables with the name `Accidents` from a map document.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    for tbl in arcpy.mapping.ListTableViews(mxd, "", df):
        if tbl.name.lower() == "accidents":
            arcpy.mapping.RemoveTableView(df, tbl)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

TableView (arcpy.mapping)

Top

Summary

Enables you to reference a table in a workspace as a **`TableView`** object so that it can be added to a map document.

Discussion

For a more complete discussion, see the TableView_Class help topic.

Syntax

TableView (table_view_data_source)

Parameter	Explanation	Data Type
table_view_data_source	A string that includes the full workspace path, including the name of the table.	String

Return Value

Data Type	Explanation
TableView	The TableView_Class object provides access to basic table properties.

UpdateLayer (arcpy.mapping)

Top

Summary

Provides the ability to update all layer properties or just the symbology for a layer in a map document (`.mxd`) by extracting the information from a source layer.

Discussion

The arcpy.mapping API only provides access to a limited number of layer properties that can be directly modified but all properties found in the **Layer Properties** dialog box can be modified using the **`UpdateLayer`** function. The information is extracted from a source layer and applied to the layer in a map document that needs to be updated. The **`source_layer`** can either be a layer (`.lyr`) file or a layer within a map document. **`UpdateLayer`** is a robust function because it can be used in several different ways to produce different results.

One option is to update only a layer's symbology. In this case, the **`update_layer`** and **`source_layer`** must have similar geometry types. Depending on the renderer (for example, unique value using a particular attribute), the attribute definitions also need to be the same. Updating symbology only is the default behavior.

Another option is to update ALL layer properties, including symbology (`symbology_only=False`). For example, you may want to update the field aliases, selection symbology, query definitions, and so on, for a layer that exists in many map documents. An easy way of doing this is to use ArcMap to modify the layer with the appropriate properties and then save the layer out to a layer (`.lyr`) file. The layer file can then be used as a **`source_layer`** to update all the properties for a given layer. Use care when updating all properties and make sure they are not properties that you don't want overwritten.

`UpdateLayer` is also capable of switching out completely unrelated layers. For example, you could replace a polygon layer with a raster layer or a group layer. This capability is only possible if the **`symbology_only`** value is set to `False`; otherwise, you might have conflicts with attributes and/or geometry types.

UpdateLayer, when not restricted to symbology, is essentially calling the RemoveLayer and AddLayer functions.

If you want to update the properties for a layer within a layer file, you must modify the properties of the layer in a map document first and then save the changes back to a layer file. See the Layer object's **`save`** or **`saveACopy`**methods.

If you are only interested in updating a layer's time properties, see UpdateLayerTime.

Syntax

UpdateLayer (data_frame, update_layer, source_layer, {symbology_only})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be updated.	DataFrame
update_layer	A Layer object representing an existing layer that will be updated.	Layer
source_layer	A reference to a Layer object that contains the information to be applied to the update_layer.	Layer
symbology_only	A Boolean that determines whether or not to update only the layer's symbology, or all other properties as well. If set to `True`, only the layer's symbology will be updated. (The default value is True)	Boolean

Code Sample

UpdateLayer example 1

The following script will update a layer's symbology only using a layer file. The layer is called `Rivers` and is in a data frame called `County Maps`.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
updateLayer = arcpy.mapping.ListLayers(mxd, "Rivers", df)[0]
sourceLayer = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.UpdateLayer(df, updateLayer, sourceLayer, True)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, sourceLayer
```

UpdateLayer example 2

The following script will completely replace a layer called `Rivers` with a group layer from another map document which displays different river feature classes at different scales.

```
import arcpy

#Reference layer in secondary map document
mxd2 = arcpy.mapping.MapDocument(r"C:\Project\ProjectTemplate.mxd")
df2 = arcpy.mapping.ListDataFrames(mxd2, "Layers")[0]
sourceLayer = arcpy.mapping.ListLayers(mxd2, "Rivers Group Layer",
df2)[0]

#Update layer in primary map document
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
updateLayer = arcpy.mapping.ListLayers(mxd, "Rivers", df)[0]
arcpy.mapping.UpdateLayer(df, updateLayer, sourceLayer, False)

#Save to a new map document and clear variable references
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, mxd2
```

UpdateLayerTime (arcpy.mapping)

Top

Summary

Provides the ability to update a layer's time properties for a layer in a map document (.mxd) by extracting time properties from a source layer.

Discussion

The UpdateLayer function has the ability to only update a layer's symbology properties or update ALL layer properties, including time properties.
The **UpdateLayerTime** function allows you to update only the time properties of a layer and therefore won't overwrite other layer properties that you don't want to change.

The source_layer contains the time properies that you want to apply. It can either be a layer file on disk or a reference to another layer in a map document.

If you want to update the properties for a layer within a layer file, you must modify the properties of the layer in a map document first and then save the changes back to a layer file. See the Layer object's **save** or **saveACopy**methods and the code example below.

Syntax

UpdateLayerTime (data_frame, update_layer, source_layer)

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be updated.	DataFrame
update_layer	A Layer object representing an existing layer that will be updated.	Layer
source_layer	A reference to a Layer object that contains the information to be applied to the update_layer.	Layer

Code Sample

UpdateLayerTime example 1

The following script will update a layer's time properties using a layer file. The layer called temperature is not time enabled. The time properties of a time-enabled layer file will be applied to the temperature layer.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Temperature.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "World")[0]
lyr = arcpy.mapping.ListLayers(mxd, "temperature", df)[0]
lyrFile =
arcpy.mapping.Layer(r"C:\Project\Data\Time\LayerWithTimeProperties.lyr
")
arcpy.mapping.UpdateLayerTime(df, lyr, lyrFile)

# Save changes to a new MXD
mxd.saveACopy(r"C:\Project\Temperature2.mxd")
# Clean up variables
del mxd, df, lyr, lyrFile
```

UpdateLayerTime example 2

The following script is similar to the one above but saves the changes back out to a layer file.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Temperature.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "World")[0]
lyr = arcpy.mapping.ListLayers(mxd, "temperature", df)[0]
lyrFile =
arcpy.mapping.Layer(r"C:\Project\Data\Time\LayerWithTimeProperties.lyr
") #orginally authored in ArcMap
arcpy.mapping.UpdateLayerTime(df, lyr, lyrFile)

# Save changes to a new Layer file
lyr.saveACopy(r"C:\Project\TemperatureWithTime.lyr")
# Clean up variables
del mxd, df, lyr, lyrFile
```

ArcPy.Mapping Parameter Constants

-with required and {optional} parameters

{add_position}—Used by AddLayer and AddLayerToGroup.

- "AUTO_ARRANGE"—Default
- "BOTTOM"
- "TOP"

{colorspace}—Used by DataDrivenPages.exportToPDF, ExportToAI, ExportToEPS, and ExportToPDF.

- "CMYK"
- "RGB"—Default

{color_mode}—Used by ExportToBMP, ExportToGIF, ExportToJPEG, ExportToPNG, and ExportToTIFF.

- "1-BIT_MONOCHROME_MASK"
- "1-BIT_MONOCHROME_THRESHOLD"
- "24-BIT_TRUE_COLOR"—Default for all but ExportToGIF
- "8-BIT_GRAYSCALE"—Default for ExportToGIF
- "8-BIT_PALETTE"

connection_type—Used by CreateGISServerConnectionFile.

- "ADMINISTER_GIS_SERVICES"
- "PUBLISH_GIS_SERVICES"
- "USE_GIS_SERVICES"

{dataset_option}—Used by ExportReport.

- "ALL"
- "DEFINITION_QUERY"
- "EXTENT"
- "SELECTED"
- "USE_RLF"—Default

displayUnits—A property of the DataFrame object.

- "Centimeters"
- "DecimalDegrees"
- "DecimalDegreesMinutes"
- "DecimalDegreesSeconds"
- "Decimeters"
- "Feet"
- "Inches"
- "Kilometers"
- "Meters"
- "MGRS"
- "Miles"
- "Millimeters"
- "NauticalMiles"
- "Points"
- "Unknown"
- "USNationalGrid"
- "UTM"
- "Yards"

{element_type}—Used by ListLayoutElements.

- "DATAFRAME_ELEMENT"
- "GRAPHIC_ELEMENT"
- "LEGEND_ELEMENT"
- "MAPSURROUND_ELEMENT"
- "PICTURE_ELEMENT"
- "TEXT_ELEMENT"

{encryption}—Used by PDFDocument.updateDocSecurity.

- "AES_V1"
- "AES_V2"
- "RC4"—Default

{gif_compression}—Used by ExportToGIF.

- "LZW"
- "NONE"—Default
- "RLE"

{image_compression}—Used by DataDrivenPages.exportToPDF, ExportToEPS, and ExportToPDF.

- "ADAPTIVE"—Default
- "DEFLATE"
- "JPEG"
- "LZW"
- "NONE"
- "RLE"

{image_quality}—Used by DataDrivenPages.exportToPDF, ExportToAI, ExportToEMF, ExportToEPS, ExportToPDF, ExportToSVG, and PrintMap.

- "BEST"—Default
- "BETTER"
- "FASTER"
- "FASTEST"
- "NORMAL"

{insert_position}—Used by InsertLayer and MoveLayer.

- "AFTER"
- "BEFORE"—Default

{layers_attributes}—Used by DataDrivenPages.exportToPDF and ExportToPDF.

- "LAYERS_AND_ATTRIBUTES"
- "LAYERS_ONLY"—Default
- "NONE"

{layer_property}—Used by Layer.supports.

- "BRIGHTNESS"
- "CONTRAST"
- "CREDITS"
- "DATASETNAME"
- "DATASOURCE"
- "DEFINITIONQUERY"
- "DESCRIPTION"
- "LABELCLASSES"
- "LONGNAME"
- "MAXSCALE"
- "MINSCALE"
- "NAME"
- "SERVICEPROPERTIES"
- "SHOWLABELS"
- "SYMBOLOGY"
- "SYMBOLOGYTYPE"
- "TIME"
- "TRANSPARENCY"
- "VISIBLE"
- "WORKSPACEPATH"

{msd_anti_aliasing}—Used by ConvertToMSD.

- "BEST"
- "FAST"
- "FASTEST"
- "NONE"—Default
- "NORMAL"

{msd_text_anti_aliasing}—Used by ConvertToMSD.

- "FORCE"—Default
- "NONE"
- "NORMAL"

{multiple_files}—Used by PDFDocument.exportToPDF.

- "PDF_MULTIPLE_FILES_PAGE_INDEX"
- "PDF_MULTIPLE_FILES_PAGE_NAME"
- "PDF_SINGLE_PAGE"—Default

{page_range_type}—Used by DataDrivenPages.exportToPDF and DataDrivenPages.printPages.

- "ALL"—Default
- "CURRENT"
- "RANGE"
- "SELECTED"

Next Page

{pdf_layout}—Used by PDFDocument.updateDocProperties.

- "DONT_CARE"
- "ONE_COLUMN"
- "SINGLE_PAGE"—Default
- "TWO_COLUMN_LEFT"
- "TWO_COLUMN_RIGHT"
- "TWO_PAGE_LEFT"
- "TWO_PAGE_RIGHT"

{pdf_open_view}—Used by PDFDocument.updateDocProperties.

- "ATTACHMENT"
- "FULL_SCREEN"
- "LAYERS"
- "USE_BOOKMARKS"
- "USE_NONE"
- "USE_THUMBS"—Default
- "VIEWER_DEFAULT"

{permissions}—Used by PDFDocument.updateDocSecurity.

- "ALL"—Default
- "ALL_MASTER"
- "COPY"
- "DOC_ASSEMBLY"
- "EDIT"
- "EDIT_NOTES"
- "FILL_AND_SIGN"
- "HIGH_PRINT"
- "OPEN"
- "PRINT"
- "SECURE"

{picture_symbol}—Used by DataDrivenPages.exportToPDF, ExportToAI, ExportToEMF, ExportToEPS, ExportToPDF, and ExportToSVG.

- "RASTERIZE_BITMAP"—Default
- "RASTERIZE_PICTURE"
- "VECTORIZE_BITMAP"

{ps_lang_level}—Used by ExportToEPS.

- 2
- 3—Default

{record_set}—Used by ExportReport.

- "ALL"
- "SELECTED"
- "USE_RLF"—Default

{rle_compression}—Used by ExportToBMP.

- "NONE"—Default
- "RLE"

{save_username_password}—Used by CreateGISServerConnectionFile.

- "DO_NOT_SAVE_USERNAME"—Default
- "SAVE_USERNAME"—Default

{service_capabilities}—Used by PublishMSDToServer.

- "MAPPING"—Default
- "KML"
- "WCS"
- "WFS"
- "WMS"

{server_type}—Used by CreateGISServerConnectionFile.

- "ARCGIS_SERVER"—Default
- "SPATIAL_DATA_SERVER"

{server_type}—Used by CreateMapSDDraft.

- "ARCGIS_SERVER"—Default
- "FROM_CONNECTION_FILE"
- "MY_HOSTED_SERVICES"
- "SPATIAL_DATA_SERVER"

symbologyType—A property of the Layer object.

- "GRADUATED_COLORS"
- "GRADUATED_SYMBOLS"
- "OTHER"
- "RASTER_CLASSIFIED"
- "UNIQUE_VALUES"

{tiff_compression}—Used by ExportToTIFF.

- "DEFLATE"
- "JPEG"
- "LZW"
- "NONE"—Default
- "PACK_BITS"

timeWindowUnits—A property of DataFrameTime.

- "centuries"
- "decades"
- "days"
- "hours"
- "millseconds"
- "minutes"
- "months"
- "seconds"
- "unknown"
- "weeks"
- "years"

{version}—Used by Layer.saveACopy and MapDocument.saveACopy.

- "10.1"—Default
- "10.0"
- "8.3"
- "9.0/9.1"
- "9.2"
- "9.3"

{workspace_type}—Used by Layer.replaceDataSource, MapDocument.replaceWorkspacePaths, and TableView.replaceDataSource.

- "ACCESS_WORKSPACE"
- "ARCINFO_WORKSPACE"
- "CAD_WORKSPACE"
- "EXCEL_WORKSPACE"
- "FILEGDB_WORKSPACE"
- "NONE"
- "OLEDB_WORKSPACE"
- "PCCOVERAGE_WORKSPACE"
- "RASTER_WORKSPACE"
- "SDE_WORKSPACE"
- "SHAPEFILE_WORKSPACE"
- "TEXT_WORKSPACE"
- "TIN_WORKSPACE"
- "VPF_WORKSPACE"

Previous Page

ArcPy.Time Classes and Functions

Time Classes

Class	Description
EsriTimeDelta_class	The **`EsriTimeDelta`** class represents a duration, the difference between two dates or times.
TimeZoneInfo_class	The **`TimeZoneInfo`** class can be used to assign a time zone to a Python datetime object.

Time Functions

Class	Description
ListTimeZones	Lists valid Time Zone IDs.

EsriTimeDelta (arcpy.time)

Top

Summary

The **EsriTimeDelta** class represents a duration, the difference between two dates or times.

Discussion

The **EsriTimeDelta** class is an alternative to the core Python `datetime.timedelta` and uses internal Esri time units for intervals that can't be handled by the core Python timedelta object (such as months, weeks, and so on).

The **timeStepInterval** property from the **LayerTime** and **DataFrameTime** classes return **EsriTimeDelta** objects.

Legacy:

Prior to the 10.1 release, the **timeStepInterval** property from the **DataFrameTime** class returned core Python `datetime.timedelta` objects.

Syntax

EsriTimeDelta (interval, units)

Parameter	Explanation	Data Type
interval	The interval of the **EsriTimeDelta**.	Double
units	The units of the **EsriTimeDelta**. Valid units are milliseconds, seconds, minutes, hours, days, weeks, months, years, decades, and centuries.	String

Properties

Property	Explanation	Data Type
interval (Read Only)	The interval of the EsriTimeDelta.	Double
units (Read Only)	The units of the EsriTimeDelta.	String

Code Sample

EsriTimeDelta example 1

The following script accesses the time-step interval property of a time-enabled layer in a map document. The time-step interval is an EsriTimeDelta object. The script then prints EsriTimeDelta properties.

```
import arcpy
mxd = arcpy.mapping.MapDocument('C:/Project/Temperature.mxd')
df = arcpy.mapping.ListDataFrames(mxd, 'USA')[0]
lyr = arcpy.mapping.ListLayers(mxd,'temperature', df)[0]
tsi = lyr.time.timeStepInterval
print "Time Step Interval:", tsi
print "EsriTimeDelta interval:", tsi.interval
print "EsriTimeDelta units:", tsi.units
```

EsriTimeDelta example 2

The following script uses the EsriTimeDelta object to iterate through twelve months starting from the current date.

```
import arcpy, datetime

time = datetime.datetime.now()

for delta in range(1, 13):
    next_date = time + arcpy.time.EsriTimeDelta(1 * delta, "months")
    print next_date
```

EsriTimeDelta example 3

The following script shows how the EsriTimeDelta handles leap years. The script adds a month to January 31, 2008 and returns February 29, 2008.

```
import arcpy, datetime

time = datetime.datetime(2008, 1, 31)

for delta in range(0, 12):
    next_date = time + arcpy.time.EsriTimeDelta(delta, "months")
    print next_date
    delta = delta + 1
```

TimeZoneInfo (arcpy.time)

Top

Summary

The TimeZoneInfo class can be used to read or assign a time zone to a Python datetime object.

Discussion

Native datetime objects are not time zone aware. By assigning a time zone to a datetime object, time zone-related operations can be performed. For example, you can use the time zone associated with a time value and convert it to another time zone.

Syntax

TimeZoneInfo (time_zone_id)

Parameter	Explanation	Data Type
time_zone_id	A valid time zone ID. A list of available time zone IDs can be obtained from the ListTimeZones function.	String

Method Overview

Method	Explanation
tzname (dt)	Returns the time zone name corresponding to the Python datetime object, dt, as a string.

Methods

tzname (dt)

Parameter	Explanation	Data Type
dt	A reference to a Python datetime object. (The default value is None)	DateTime

Return Value

Data Type	Explanation
String	The time zone name corresponding to the datetime object, dt.

Returns the time zone name corresponding to the datetime object, dt, as a string.

Code Sample

TimeZoneInfo example 1

The following script applies a 'Pacific Standard Time' time zone to a Python datetime object. It then loops through each month to demonstrate how the time zone name will change to 'Pacific Daylight Time' during the summer in observance of Daylight Savings Time.

```
import arcpy
import datetime

tzinfo = arcpy.time.TimeZoneInfo('Pacific Standard Time')

time = datetime.datetime(2011, 1, 1, tzinfo=tzinfo)

for delta in range(1, 13):
    next_date = time + arcpy.time.EsriTimeDelta(1 * delta, "months")
    print next_date, tzinfo.tzname(next_date)
```

TimeZoneInfo example 2

The following script demonstrates how to convert a datetime value in Pacific Standard Time to Eastern Standard Time.

```
import arcpy
import datetime

from_tzinfo = arcpy.time.TimeZoneInfo('Pacific Standard Time')
target_tzInfo = arcpy.time.TimeZoneInfo('Eastern Standard Time')
from_time = datetime.datetime.now(from_tzinfo)
print "target_time =", str(from_time.astimezone(target_tzInfo))
```

ListTimeZones (arcpy.time)

Top

Summary

Lists valid Time Zone IDs.

Discussion

List available Time Zone IDs for use in the TimeZoneInfo class.

Syntax

ListTimeZones ()

Return Value

Data Type	Explanation
String	A list of Time Zone IDs.

Code Sample

ListTimeZones example 1

The following script will list all available Time Zone IDs.

```
import arcpy
arcpy.time.ListTimeZones()
```