ArcPy Function Index

Function name	Category
AcceptConnections	Geodatabase administration
AddDataStoreItem	Data store
AddError	Messages and error handling
AddFieldDelimiters	Fields
AddIDMessage	Messages and error handling
AddMessage	Messages and error handling
AddReturnMessage	Messages and error handling
AddToolbox	Tools and toolboxes
AddWarning	Messages and error handling
AlterAliasName	General
AsShape	Geometry
CheckExtension	Licensing and installation
CheckInExtension	Licensing and installation
CheckOutExtension	Licensing and installation
CheckProduct	Licensing and installation
ClearEnvironment	Environments and settings
Command	General
CopyParameter	Getting and setting parameters
CreateObject	General
CreateGPSDDraft	Publishing
CreateImageSDDraft	Publishing
CreateRandomValueGenerator	General
CreateScratchName	General data functions
CreateUniqueName	General data functions
Describe	Describing data
DisconnectUser	Geodatabase administration
Exists	General data functions
FromWKB	Geometry
FromWKT	Geometry
GetArgumentCount	Getting and setting parameters
GetIDMessage	Messages and error handling
GetInstallInfo	Licensing and installation
SetLogHistory	Log history
GetMaxSeverity	Messages and error handling
GetMessage	Messages and error handling
GetMessageCount	Messages and error handling
GetMessages	Messages and error handling
GetParameter	Getting and setting parameters
GetParameterAsText	Getting and setting parameters
GetParameterCount	Getting and setting parameters
GetParameterInfo	Getting and setting parameters
GetParameterValue	Getting and setting parameters
GetReturnCode	Messages and error handling
GetSeverity	Messages and error handling
GetSeverityLevel	Messages and error handling
GetSystemEnvironment	Environments and settings
ImportToolbox	Tools and toolboxes
InsertCursor	Cursors
IsSynchronous	Tools and toolboxes
ListDatasets	Listing data
ListDataStoreItems	Data store
ListEnvironments	Environments and settings
ListFeatureClasses	Listing data
ListFields	Listing data
ListFiles	Listing data
ListIndexes	Listing data
ListInstallations	Licensing and installation
ListPrinterNames	General
ListRasters	Listing data
ListSpatialReferences	Spatial references and transformations
ListTables	Listing data
ListToolboxes	Tools and toolboxes
ListTools	Tools and toolboxes
ListTransformations	Spatial references and transformations
ListUsers	Geodatabase administration
ListVersions	Listing data
ListWorkspaces	Listing data
LoadSettings	Environments and settings
NumpyArrayToRaster	Raster
ParseFieldName	Fields
ParseTableName	General data functions
ProductInfo	Licensing and installation
RasterToNumPyArray	Raster
RefreshActiveView	General
RefreshCatalog	General
RefreshTOC	General
RemoveDataStoreItem	Data store
RemoveToolbox	Tools and toolboxes
ResetEnvironments	Environments and settings
ResetProgressor	Progress dialog
SaveSettings	Environments and settings
SearchCursor	Cursors
SetLogHistory	Log history
SetParameter	Getting and setting parameters
SetParameterAsText	Getting and setting parameters
SetProduct	Licensing and installation
SetProgressor	Progress dialog
SetProgressorLabel	Progress dialog
SetProgressorPosition	Progress dialog
SetSeverityLevel	Messages and error handling
TestSchemaLock	General data functions
UpdateCursor	Cursors
Usage	General
ValidateDataStoreItem	Data store
ValidateFieldName	Fields
ValidateTableName	General data functions

AcceptConnections (arcpy)

Top

Summary

Allows an administrator to enable or disable the ability of nonadministrative users to make connections to an enterprise geodatabase.

Discussion

The **AcceptConnections** function is used by an administrative user to temporarily block connections to an Enterprise geodatabase. This function is used to complement the Connections tab on an Enterprise geodatabase properties page found in ArcGIS for Desktop.

- The **AcceptConnections** function must utilize an administrative connection to the database.
- If this function is attempted to be run by a nonadministrative user the function will fail.

Syntax

AcceptConnections (sde_workspace, accept_connections)

Parameter	Explanation	Data Type
sde_workspace	The Enterprise geodatabase that will have its connection property altered. The connection properties specified in the Enterprise Geodatabase must be the geodatabase administrator.	String
accept_connections	Boolean value indicating if the geodatabase will accept connections (True) or will not accept connections (False).	Boolean

Code Sample

AcceptConnections example 1

The following example demonstrates how to prevent connections to a geodatabase.

```
import arcpy

arcpy.AcceptConnections("Database Connections/admin.sde", False)
```

AcceptConnections example 2

The following example demonstrates how to prevent connections to a geodatabase, disconnect active connections, then run a geodatabase compress.

```
import arcpy

# Set Admin workspace variable
admin_workspace = "Database Connections/admin.sde"

# Block connections
arcpy.AcceptConnections(admin_workspace, False)

# Disconnect users
arcpy.DisconnectUser(admin_workspace, 'ALL')

# Reconcile/Post using default parameters.
arcpy.ReconcileVersions_management(admin_workspace, 'ALL_VERSIONS',
                                   'sde.DEFAULT', with_post='POST')

# Compress the geodatabase
arcpy.Compress_management(admin_workspace)

# Allow connections.
arcpy.AcceptConnections(admin_workspace, True)
```

Related Topics

DisconnectUser
ListUsers

AddDataStoreItem (arcpy)

Top

Summary

Registers a folder or database with an ArcGIS Server site. See About registering your data with the server to learn more about when and why you should register your data.

Discussion

See About registering your data with the server to learn more about when and why you should register your data with ArcGIS Server.

Syntax

AddDataStoreItem (connection_file, datastore_type, connection_name, server_path, {client_path}, {hostname})

Parameter	Explanation	Data Type
connection_file	An ArcGIS Server connection file (.ags) representing the server with which you want to register the data. If you've made a connection in ArcCatalog you can use the connection file found in your user profile directory. Alternatively, you can create a connection file from scratch using the functionCreateGISServerConnectionFile.	String
datastore_type	The type of data being registered. • DATABASE —The data resides in an enterprise database. • FOLDER —The data is file-based.	String
connection_name	A name for this folder or database that publishers or administrators will see when they view the server properties.	String
server_path	The path or connection to the data as seen by the server. If you are registering a **DATABASE**, this is either the path to a database connection file (.sde) or a string containing the database connection parameters. See Database connections in ArcGIS Desktop to learn how to obtain this file or string. If you are registering a **FOLDER**, this is the path to the folder.	String
client_path	The path or connection to the data as seen by the publisher's machine, if different from the information used by the server. In some cases the publisher and the server may be referencing physically distinct databases or folders. When you provide the publisher path and the server path, ArcGIS Server automatically corrects the paths at publish time when your map documents and other resources are transferred to the server. If you are registering a **DATABASE**, provide either the path to a database connection file (.sde) or a string containing the database connection parameters. See Database connections in ArcGIS Desktop to learn how to obtain this file or string. If you are registering a **FOLDER**, provide the path to the folder. If you are registering ArcGIS Server's Managed Database, do not provide a path; instead, provide the string `managed` for this parameter. ArcGIS Server's Managed Database is an enterprise geodatabase you designate where data can be copied at publish time if a user attempts to publish a feature service from an unregistered data location. See Copying data to the server automatically when publishing to learn more.	
hostname	The name of the publisher or client machine that will use this registered folder or database. If left blank, the name of the machine running the script will be used.	String

Return Value

Data Type	Explanation
String	If successful, returns the string "Success".

Code Sample

Register a folder used by both the server and publisher

Registers a local folder C:\temp with ArcGIS Server. Assumes that a connection to the server has been created in the ***Catalog*** window of ArcMap and renamed MyConnection.

```
import arcpy

conn = "GIS Servers/MyConnection.ags"
path = "c:/temp"

arcpy.AddDataStoreItem(conn, "FOLDER", "My local data folder", path,
path)
```

Register a folder that differs between the server and publisher

Registers a shared folder \\MYSERVER\mydata\Washington with the server, with the local folder C:\mydata\Washington being used by the publisher.

```
import arcpy

conn = "c:/connections/MYSERVER.ags"

arcpy.AddDataStoreItem(conn, "FOLDER", "Washington",
                       "//MYSERVER/mydata/Washington",
                       "c:/mydata/Washington", "MYPUBLISHER")
```

Continued on next page.

Register a database used by both the server and publisher

Registers an enterprise database `wilma` used by both the server and the publisher machines. Uses an .sde connection file, created when you add a database connection in the ***Catalog*** window of ArcMap.

```
import arcpy

server_conn = "c:/connections/MYSERVER.ags"
db_conn = "c:/connections/Connection to wilma.sde"

arcpy.AddDataStoreItem(server_conn, "DATABASE", "Wilma", db_conn,
db_conn)
```

Register a database that differs between the server and publisher

Registers an enterprise database `wilma` with the server, with the database `pebbles` being used by the publisher.

```
import arcpy

server_conn = "c:/connections/MYSERVER.ags"
db_conn_serv = "c:/connections/Connection to wilma.sde"
db_conn_pub = "c:/connections/Connection to pebbles.sde"

arcpy.AddDataStoreItem(
    server_conn, "DATABASE", "WilmaAndPebbles", db_conn_serv,
db_conn_pub)
```

Register a database as ArcGIS Server's Managed Database

Registers an enterprise database `wilma` as ArcGIS Server's Managed Database. If a publisher attempts to publish a feature service from an unregistered data location, the data will be copied here.

```
import arcpy

server_conn = "c:/connections/MYSERVER.ags"
db_conn_serv = "c:/connections/Connection to wilma.sde"

arcpy.AddDataStoreItem(
    server_conn, "DATABASE", "WilmaManaged", db_conn_serv, "managed")
```

Register a database using a connection string

Registers an enterprise database `serverX` with the server using a database connection string.

```
import arcpy

server_conn = "c:/connections/MYSERVER.ags"

db_conn_string = u"PASSWORD=pwdX;SERVER=serverX;" + \

u"INSTANCE=sde:sqlserver:serverX;DBCLIENT=sqlserver;" + \
                u"DB_CONNECTION_PROPERTIES=serverX;" + \
                u"DATABASE=sde;USER=userX;AUTHENTICATION_MODE=DBMS"

arcpy.AddDataStoreItem(
    server_conn, "DATABASE", "ServerX", db_conn_string,
db_conn_string)
```

Related Topics

ListDataStoreItems
RemoveDataStoreItem
ValidateDataStoreItem

AddError (arcpy)

Top

Summary

Creates a geoprocessing tool error message (Severity=2) that can be accessed by any of the GetMessages functions.

Syntax

AddError (message)

Parameter	Explanation	Data Type
message	The message to add.	String

Code Sample

AddError example

Add custom geoprocessing error message.

```
import arcpy

fc = arcpy.GetParameterAsText(0)

# Get the count from GetCount's Result object
feature_count = int(arcpy.GetCount_management(fc).getOutput(0))

if feature_count == 0:
    arcpy.AddError("{0} has no features.".format(fc))
else:
    arcpy.AddMessage("{0} has {1} features.".format(fc,
feature_count))
```

Related Topics

AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

AddFieldDelimiters (arcpy)

Top

Summary

Adds field delimiters to a field name to allow for use in SQL expressions.

Syntax

AddFieldDelimiters (datasource, field)

Parameter	Explanation	Data Type
datasource	The field delimiters are based on the data source used.	String
field	The field name to which delimiters will be added. The field does not have to currently exist.	String

Return Value

Data Type	Explanation
String	Returns a delimited field name.

Code Sample

AddFieldDelimiters example

```
import arcpy

field_name = arcpy.GetParameterAsText(0)
arcpy.env.workspace = arcpy.GetParameterAsText(1)
in_features = arcpy.GetParameterAsText(2)
out_feat_class = arcpy.GetParameterAsText(3)
state value = arcpy.GetParameterAsText(4)

# AddFieldDelimiters will return a field name with the proper
#  field delimiters for the workspace specified.
#
sql_exp = """{0} = {1}""".format(
    arcpy.AddFieldDelimiters('c:/data', field name),
    field_name)

# Use delimited field for Select tool SQL expression
#
arcpy.Select_analysis(in_features, out_feat_class, sql_exp)
```

Related Topics

Specifying a query in Python

AddIDMessage (arcpy)

Top

Summary

Allows you to use system messages with a script tool. A list of messages and IDs that can be used are provided under Understanding geoprocessing tool errors and warnings.

Discussion

Geoprocessing errors and warnings are returned from geoprocessing tools with a six-digit code and a text message. Every error and warning has a corresponding description page in the desktop help system. This page contains both a detailed description of the error and possible solutions for the error. In tool dialog boxes, the ***Python*** window, and the ***Results*** window, the ID code is a link that, when clicked, takes you to a description page.

Syntax

AddIDMessage (message_type, message_ID, {add_argument1}, {add_argument2})

Parameter	Explanation	Data Type
message_type	The message type defines whether the message will be an error, warning, or informative. Valid message types are: • ERROR —Adds an error message to the tool messages. • INFORMATIVE —Adds an informative message to the tool messages. • WARNING —Adds a warning message to the tool messages.	String
message_ID	The message ID allows you to reference existing messages for your scripting errors and warnings.	Integer
add_argument1	Depending on which message ID is used, an argument may be necessary to complete the message. Common examples include dataset or field names. Datatype can be string, integer, or double.	Object
add_argument2	Depending on which message ID is used, an argument may be necessary to complete the message. Common examples include dataset or field names. Datatype can be string, integer, or double.	Object

Code Sample

AddIDMessage example

Add a message to a Python script tool.

```
class overwriteError(Exception):
    pass

import arcpy

in_feature_class = arcpy.GetParameterAsText(0)
out_feature_class = arcpy.GetParameterAsText(1)

try:
    # If the output feature class already exists, raise an error
    if arcpy.Exists(in_feature_class):
        # Raise a custom exception
        raise overwriteError(out_feature_class)
    else:
        arcpy.CopyFeatures_management(in_feature_class,
out_feature_class)

except overwriteError as err:
    # Use message ID 12, and provide the output feature class
    #    to complete the message.
    arcpy.AddIDMessage("Error", 12, str(err))
```

Related Topics

AddError
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

AddMessage (arcpy)

Top

Summary

Creates a geoprocessing informative message (Severity=0) that can be accessed with any of the GetMessages functions.

Syntax

AddMessage (message)

Parameter	Explanation	Data Type
message	The message to add.	String

Code Sample

AddMessage example

Add custom informative message to the Python script tool.

```
import arcpy

fc = arcpy.GetParameterAsText(0)

# Get the count from GetCount's Result object
feature_count = int(arcpy.GetCount_management(fc).getOutput(0))

if feature_count == 0:
    arcpy.AddError("{0} has no features.".format(fc))
else:
    arcpy.AddMessage("{0} has {1} features.".format(fc,
feature_count))
```

Related Topics

AddError
AddIDMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

AddReturnMessage (arcpy)

Top

Summary

Sets the return message of a script tool as an output message by index.

Discussion

There are times when you may want to return all messages from a tool you've called, regardless of message severity. Using the index parameter, **AddReturnMessage** will return a message from the last tool executed. The severity of the message (warning, error, and so on, is preserved).

Geoprocessing error numbers shown in the progress dialog box are hyperlinks to a help page that further describes the error. To enable hyperlinks for errors in your script, use the **AddReturnMessage** function instead of the **AddError** function, as follows:

```
import arcpy
try:
    result = arcpy.GetCount_management("c:/data/rivers.shp")

except:
    # Return Geoprocessing tool specific errors
    #
    for msg in range(0, arcpy.GetMessageCount()):
        if arcpy.GetSeverity(msg) == 2:
            arcpy.AddReturnMessage(msg)
```

Syntax

AddReturnMessage (index)

Parameter	Explanation	Data Type
index	The message index.	Integer

Code Sample

AddReturnMessage example

Returns all messages from the last tool executed as script tool output messages.

```
import arcpy

# Set current workspace
arcpy.env.workspace = "c:/data/base.gdb"

arcpy.Buffer_analysis("roads", "roads_buffer_1000", "1000 feet")

# Return the resulting messages as script tool output messages
for i in xrange(0, arcpy.GetMessageCount()):
    arcpy.AddReturnMessage(i)
```

Related Topics

AddError
AddIDMessage
AddMessage
AddWarning
GetMessage
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

AddToolbox (arcpy)

Top

Summary

Imports the specified toolbox into ArcPy, allowing for access to the toolbox's associated tools.

Note:

Equivalent to the **ImportToolbox** function.

Discussion

While any of the core ArcGIS toolboxes are accessible by default in a script, your own custom or third-party toolboxes must be added using **ImportToolbox** to use them in a script.

Other toolboxes may be found in any number of different folders or geodatabases, and may have many different origins; they may be toolboxes you have personally created, or are toolboxes created internally by your organization, or are toolboxes you have downloaded from sites like the Geoprocessing Resource Center. In any case, these toolboxes need to be imported into ArcPy in a one-step process before they can be used as tools in Python.

Server toolboxes can also be added using a semicolon delimiter.

Server	Syntax
Internet ArcGIS for Server	URL;servicename;{username};{password}

Learn more about using a geoprocessing service in Python

Syntax

AddToolbox (input_file, {module_name})

Parameter	Explanation	Data Type
input_file	The geoprocessing toolbox to be added to the ArcPy site package.	String
module_name	If the toolbox does not have an alias, the **module_name** is required. When a tool is accessed through the ArcPy site package, the toolbox alias where the tool is contained is a required suffix (arcpy.<toolname>_<alias>). Since ArcPy depends on toolbox aliases to access and execute the correct tool, aliases are extremely important when importing custom toolboxes. A good practice is to always define a custom toolbox's alias. However, if the toolbox alias is not defined, a temporary alias can be set as the second parameter.	String

Return Value

Data Type	Explanation
Module	Returns the imported module. If needed, tool names can be accessed from the module's **__all__** property.

Code Sample

AddToolbox example

Add the specified toolbox.

```
import arcpy

# Import custom toolbox
arcpy.AddToolbox("c:/tools/My_Analysis_Tools.tbx")

try:
    # Run tool in the custom toolbox.  The tool is identified by
    #  the tool name and the toolbox alias.
    arcpy.GetPoints_myanalysis("c:/data/forest.shp")
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

ImportToolbox

Using tools in Python

AddWarning (arcpy)

Top

Summary

Creates a geoprocessing warning message (Severity=1) that can be accessed by any of the GetMessages functions.

Syntax

AddWarning (message)

Parameter	Explanation	Data Type
message	The message to add.	String

Code Sample

AddWarning example

Add custom warning message to a script tool.

```
import arcpy

fc = arcpy.GetParameterAsText(0)

# Get the count from GetCount's Result object
feature_count = int(arcpy.GetCount_management(fc).getOutput(0))

if feature_count == 0:
    arcpy.AddError("{0} has no features.".format(fc))
else:
    arcpy.AddMessage("{0} has {1} features.".format(fc,
feature_count))
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
GetMessage
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

AlterAliasName (arcpy)

Top

Summary

Updates the alias name for a table or feature class.

Syntax

AlterAliasName (table, alias)

Parameter	Explanation	Data Type
table	Input table or feature class.	String
alias	The new alias name.	String

Code Sample

AlterAliasName example

Updates the alias name for a newly created table.

```
import arcpy

arcpy.CreateTable management('c:/city/Boston.gdb', 'SnowReport')
arcpy.AlterAliasName('c:/city/Boston.gdb/SnowReport', 'Snow
Report')
```

AsShape (arcpy)

Top

Summary

Converts Esri JSON or GeoJSON to ArcPy geometry objects. GeoJSON is a geospatial data interchange format for encoding geographic data structures.

Syntax

AsShape (geojson_struct, {esri_json})

Parameter	Explanation	Data Type
geojson_struct	The **geojson_struct** includes **type** and **coordinates**. The following strings are included for **type**: **Point**, **LineString**, **Polygon**, **MultiPoint**, and **MultiLineString**.	Dictionary
esri_json	Sets whether the input JSON is evaluated as Esri JSON or GeoJSON. If True, the input is evaluated as Esri JSON. (The default value is False)	Boolean

Return Value

Data Type	Explanation
Geometry	**AsShape** returns a geometry object (PointGeometry, Multipoint, Polyline, or Polygon) based on the input GeoJSON or esriJSON object.

```
import arcpy
geojson_point = {"type": "Point", "coordinates": [5.0, 5.0]}
point = arcpy.AsShape(geojson_point)
```

Code Sample

AsShape example 1

Create a PointGeometry object using a GeoJSON object.

```
import arcpy

geojson_point = {
    "type": "Point",
    "coordinates": [5.0, 5.0]}
point = arcpy.AsShape(geojson_point)
```

AsShape example 2

Create a PointGeometry object using an Esri JSON object.

```
import arcpy

esri_json = {
    "x": -122.65,
    "y": 45.53,
    "spatialReference": {
        "wkid": 4326}}
# Set the second parameter to True to use an esri JSON
point = arcpy.AsShape(esri_json, True)
```

AsShape example 3

Create a Multipoint object using a GeoJSON object.

```
import arcpy

geojson_multipoint = {
    "type": "MultiPoint",
    "coordinates": [[5.0, 4.0], [8.0, 7.0]]}
multipoint = arcpy.AsShape(geojson_multipoint)
```

AsShape example 4

Create a Multipoint object using an Esri JSON object.

```
import arcpy

esri_json = {
    "points" : [
        [-97.06138, 32.837],
        [-97.06133, 32.836],
        [-97.06124, 32.834],
        [-97.06127, 32.832]],
    "spatialReference" : {"wkid" : 4326}}
# Set the second parameter to True to use an esri JSON
multipoint = arcpy.AsShape(esri_json, True)
```

Continued on next page.

AsShape example 5

Create a Polyline object using a GeoJSON object.

```
import arcpy

geojson linestring = {
    "type": "LineString",
    "coordinates": [[5.0, 4.0], [8.0, 7.0]]}
polyline = arcpy.AsShape(geojson_linestring)
```

AsShape example 6

Create a Polyline object using an Esri JSON object.

```
import arcpy

esri_json = {
    "paths" : [
        [[-97.08, 32.8], [-97.05, 32.6], [-97.06, 32.7],
         [-97.07, 32.6]],
        [[-97.4, 32.5], [-97.2, 32.75]]],
    "spatialReference" : {"wkid" : 4326}}
# Set the second parameter to True to use an esri JSON
polyline = arcpy.AsShape(esri_json, True)
```

AsShape example 7

Create a multipart Polyline object using a GeoJSON object.

```
import arcpy

geojson_multilinestring = {
    "type": "MultiLineString",
    "coordinates": [
        [[5.0, 4.0], [8.0, 7.0]],
        [[4.0, 5.0], [7.0, 8.0]]]}
polyline = arcpy.AsShape(geojson_multilinestring)
```

AsShape example 8

Create a Polygon object using a GeoJSON object.

```
import arcpy

geojson_polygon = {
    "type": "Polygon",
    "coordinates": [
        [[10.0, 0.0], [20.0, 0.0], [20.0, 10.0], [10.0, 10.0],
         [10.0, 0.0]]]}
polygon = arcpy.AsShape(geojson_polygon)
```

AsShape example 9

Create a Polygon with a hole object using a GeoJSON object.

```
import arcpy

geojson_polygon = {
    "type": "Polygon",
    "coordinates": [
        [[10.0, 0.0], [20.0, 0.0], [20.0, 10.0], [10.0, 10.0],
         [10.0, 0.0]],
        [[12.0, 2.0], [18.0, 2.0], [18.0,  8.0], [12.0,  8.0],
         [12.0, 2.0]]]}
polygon = arcpy.AsShape(geojson_polygon)
```

Related Topics

FromWKB
FromWKT

CheckExtension (arcpy)

Top

Summary

Checks to see if a license is available to be checked out for a specific type of extension.
Once the extension license has been retrieved by the script, tools using that extension can be used. Once a script is finished with an extension's tools, the CheckInExtension function should be used to return the license to the License Manager so other applications can use it. All checked-out extension licenses and set product licenses are returned to the License Manager when a script completes.

Syntax

CheckExtension (extension_code)

Parameter	Explanation	Data Type
extension_code	Keyword for the extension product that is being checked. • 3D —3D Analyst • Schematics —ArcGIS Schematics • ArcScan —ArcScan • Business —Business Analyst • DataInteroperability —Data Interoperability • GeoStats —Geostatistical Analyst • JTX — Workflow Manager • Network —Network Analyst • Aeronautical —Esri Aeronautical Solution • Defense —Esri Defense Solution • Foundation —Esri Production Mapping • Datareviewer —ArcGIS Data Reviewer • Nautical —Esri Nautical Solution • NauticalB —Esri Bathymetry • Spatial —Spatial Analyst • StreetMap —StreetMap • Tracking —Tracking Licensing and extensions	String

Return Value

Data Type	Explanation
String	There are four possible returned values for CheckExtension: • Available —The requested license is available to be set. • Unavailable —The requested license is unavailable to be set. • NotLicensed —The requested license is not valid. • Failed —A system failure occurred during request.

Code Sample

CheckExtension example

Check for availability of 3D Analyst extension before checking it out.

```
import arcpy

class LicenseError(Exception):
    pass

try:
    if arcpy.CheckExtension("3D") == "Available":
        arcpy.CheckOutExtension("3D")
    else:
        # raise a custom exception
        raise LicenseError

    arcpy.env.workspace = "c:/GrosMorne"
    arcpy.HillShade_3d("WesternBrook", "wbrook_hill", 300)
    arcpy.Aspect_3d("WesternBrook", "wbrook_aspect")
    arcpy.CheckInExtension("3D")

except LicenseError:
    print("3D Analyst license is unavailable")
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

CheckInExtension
CheckOutExtension
Accessing licenses and extensions in Python
An overview of the extensions of ArcGIS

CheckInExtension (arcpy)

Top

Summary

Returns the license to the License Manager so other applications can use it. Once the extension license has been retrieved by the script, tools using that extension can be used. Once a script is finished with an extension's tools, the CheckInExtension function should be used to return the license to the License Manager so other applications can use it. All checked-out extension licenses and set product licenses are returned to the License Manager when a script completes.

Syntax

CheckInExtension (extension_code)

Parameter	Explanation	Data Type
extension_code	Keyword for the extension product that is being checked. • 3D —3D Analyst • Schematics —ArcGIS Schematics • ArcScan —ArcScan • Business —Business Analyst • DataInteroperability —Data Interoperability • GeoStats —Geostatistical Analyst • JTX — Workflow Manager • Network —Network Analyst • Aeronautical —Esri Aeronautical Solution • Defense —Esri Defense Solution • Foundation —Esri Production Mapping • Datareviewer —ArcGIS Data Reviewer • Nautical —Esri Nautical Solution • NauticalB —Esri Bathymetry • Spatial —Spatial Analyst • StreetMap —StreetMap • Tracking —Tracking Licensing and extensions	String

Return Value

Data Type	Explanation
String	There are three possible returned values for CheckInExtension: • NotInitialized —No desktop license has been set. • Failed —A system failure occurred during the request. • CheckedIn — The license has been returned successfully.

Code Sample

CheckInExtension example

Return 3D extension license to License manager.

```
import arcpy

class LicenseError(Exception):
    pass

try:
    if arcpy.CheckExtension("3D") == "Available":
        arcpy.CheckOutExtension("3D")
    else:
        # raise a custom exception
        raise LicenseError

    arcpy.env.workspace = "c:/GrosMorne"
    arcpy.HillShade_3d("WesternBrook", "wbrook_hill", 300)
    arcpy.Aspect_3d("WesternBrook", "wbrook_aspect")
    arcpy.CheckInExtension("3D")

except LicenseError:
    print("3D Analyst license is unavailable")
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

CheckExtension
CheckOutExtension
Accessing licenses and extensions in Python
An overview of the extensions of ArcGIS

CheckOutExtension (arcpy)

Top

Summary

Retrieves the license from the License Manager.
Once the extension license has been retrieved by the script, tools using that extension can be used. Once a script is finished with an extension's tools, the CheckInExtension function should be used to return the license to the License Manager so other applications can use it. All checked-out extension licenses and set product licenses are returned to the License Manager when a script completes.

Discussion

Tip:

The setting of the product and extensions is only necessary within stand-alone scripts. If you are running tools from the Python window or using script tools, the product is already set from within the application, and the active extensions are based on the Extensions dialog box.

Syntax

CheckOutExtension (extension_code)

Parameter	Explanation	Data Type
extension_code	Keyword for the extension product that is being checked. • 3D —3D Analyst • Schematics —ArcGIS Schematics • ArcScan —ArcScan • Business —Business Analyst • DataInteroperability —Data Interoperability • GeoStats —Geostatistical Analyst • JTX — Workflow Manager • Network —Network Analyst • Aeronautical —Esri Aeronautical Solution • Defense —Esri Defense Solution • Foundation —Esri Production Mapping • Datareviewer —ArcGIS Data Reviewer • Nautical —Esri Nautical Solution • Nauticalb —Esri Bathymetry • Spatial —Spatial Analyst • StreetMap —StreetMap • Tracking —Tracking Licensing and extensions	String

Return Value

Data Type	Explanation
String	There are three possible returned values for CheckOutExtension: • NotInitialized —No desktop license has been set. • Unavailable —The requested license is unavailable to be set. • CheckedOut —The license has been set successfully.

Code Sample

CheckOutExtension example

Check out 3D extension for use by tools.

```
import arcpy

class LicenseError(Exception):
    pass

try:
    if arcpy.CheckExtension("3D") == "Available":
        arcpy.CheckOutExtension("3D")
    else:
        # raise a custom exception
        raise LicenseError

    arcpy.env.workspace = "c:/GrosMorne"
    arcpy.HillShade_3d("WesternBrook", "wbrook_hill", 300)
    arcpy.Aspect_3d("WesternBrook", "wbrook_aspect")
    arcpy.CheckInExtension("3D")

except LicenseError:
    print("3D Analyst license is unavailable")
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

CheckExtension
CheckInExtension
Accessing licenses and extensions in Python
An overview of the extensions of ArcGIS

CheckProduct (arcpy)

Top

Summary

Checks to see if the requested license is available.

Syntax

CheckProduct (product)

Parameter	Explanation	Data Type
product	Product code for the product being checked. • arcview —ArcGIS for Desktop Basic product code • arceditor —ArcGIS for Desktop Standard product code • arcinfo —ArcGIS for Desktop Advanced product code • engine —Engine Runtime product code • enginegeodb —Engine Geodatabase Update product code • arcserver — Server product code	String

Return Value

Data Type	Explanation
String	There are five possible return values for CheckProduct: • AlreadyInitalized — License has already been set in the script. • Available — The requested license is available to be set. • Unavailable — The requested license is unavailable to be set. • NotLicensed — The requested license is not valid. • Failed — A system failure occurred during the request.

Code Sample

CheckProduct example

Check to see if an ArcGIS for Desktop Advanced license is available.

```
import sys
import arcpy

arcpy.env.workspace = "c:/data/world.gdb"

if arcpy.CheckProduct("ArcInfo") == "Available":
    arcpy.PolygonToLine_management("Lakes", "LakeLines")
else:
    msg = 'ArcGIS for Desktop Advanced license not available'
    print(msg)
    sys.exit(msg)
```

Related Topics

SetProduct
ProductInfo
Accessing licenses and extensions in Python

ClearEnvironment (arcpy)

Top

Summary

Resets a specific environment setting to its default.

Syntax

ClearEnvironment (environment_name)

Parameter	Explanation	Data Type
environment_name	The name of the environment setting that will be reset to its default setting.	String

Code Sample

ClearEnvironment example

Sets the specified environment setting back to its default.

```
import arcpy

arcpy.env.workspace = "c:/data/world.gdb"

# prints c:/data/world.gdb
print(arcpy.env.workspace)

arcpy.ClearEnvironment("workspace")

# prints None
print(arcpy.env.workspace)
```

Related Topics

Using environment settings in Python
GetSystemEnvironment
ListEnvironments
ResetEnvironments

Command (arcpy)

Top

Summary

Executes a geoprocessing tool as a single string.

Discussion

Legacy:

At ArcGIS 10, the Python window was introduced to replace the Command Line window. The Command function uses the syntax as used by the original Command Line, that is, parameters separated by spaces, unlike Python which separates parameters with commas.

Syntax

Command (command_line)

Parameter	Explanation	Data Type
command_line	The double-quoted string representing a command line command that is to be executed.	String

Return Value

Data Type	Explanation
String	The geoprocessing tool messages, separated by a newline ('\n'). If the tool fails, only the error messages are returned.

Code Sample

Command example

Execute double-quoted command line string.

```
import arcpy

# Set current workspace and define command line command.
arcpy.env.workspace = "c:/data/florida.gdb"
command_string = "Clip_analysis Runways DadeCounty DadeRunways"

# Execute command line string
arcpy.Command(command_string)
```

Related Topics

Using tools in Python

CopyParameter (arcpy)

Top

Summary

Copies the specified parameter by index to another parameter in the script tool. The specified parameters must be of the same data type.

Syntax

CopyParameter (from_param, to_param)

Parameter	Explanation	Data Type
from_param	The index position of the parameter to be copied.	Integer
to_param	The index position of the parameter that will be copied to.	Integer

Code Sample

CopyParameter example

Copy input parameter to output parameter.

```
import arcpy

# Copy the script tool's specified input parameter object
#  to the script tool's specified output parameter.
arcpy.CopyParameter(0, 1)
```

Related Topics

GetArgumentCount
GetParameter
GetParameterAsText
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

CreateObject (arcpy)

Top

Summary

Creates geoprocessing objects. The extra arguments can be used to specify additional requirements for the object creation such as the number of columns in the ValueTable object.

Discussion

Note:

Instead of using CreateObject, it is simpler and more direct to use the equivalent ArcPy class. For instance, instead of arcpy.CreateObject("array"), use arcpy.Array().

ArcSDESQLExecute
Array
Extent
FeatureSet
Field
FieldInfo
FieldMap
FieldMappings
Geometry
NetCDFFileProperties
Point
RecordSet
Result
SpatialReference
ValueTable

Syntax

CreateObject (name, {options})

Parameter	Explanation	Data Type
name	Name of the object to be created ArcSDESQLExecute, Array, Extent, FeatureSet, Field, FieldInfo, FieldMap, FieldMappings, Geometry, NetCDFFileProperties, Point, RecordSet, Result, SpatialReference, ValueTable.	String
options	Optional argument(s) depend on the object being created.	Object

Return Value

Data Type	Explanation
Object	The object returned depends on type of object specified in the first parameter.

Code Sample

CreateObject example

Use value table to hold feature class names and ranks for the Union tool.

```
import arcpy

# Set the workspace. List all of the feature classes in the dataset
arcpy.env.workspace = "c:/data/landbase.gdb/wetlands"
fcs = arcpy.ListFeatureClasses()

# Create the value table for the Analysis Union tool with 2 columns
vtab = arcpy.CreateObject("valuetable", 2)

# Iterate through the list of feature classes
for fc in fcs:
    # Update the value table with a rank of 2 for each record,
except
    #   for BigBog
    if fc.lower() != "bigbog":
        vtab.addRow(fc + " 2")
    else:
        vtab.addRow(fc + " 1")

# Union the wetlands feature classes with the land use feature
class
# to create a single feature class with all of the wetlands and
land
# use data
vtab.addRow("c:/data/landbase.gdb/land_use 2")
arcpy.Union_analysis(vtab, "c:/data/landbase.gdb/wetlands_use")
```

Related Topics

Using classes in Python

CreateGPSDDraft (arcpy)

Top

Summary

The function converts **Result** objects and result files (.rlt) into Service Definition Draft (.sddraft) files.

Note:

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

Discussion

CreateGPSDDraft is the first step to automating the publishing of a geoprocessing result to a GIS Server using ArcPy. The output created from the **CreateGPSDDraft** is a Service Definition Draft (.sddraft) file. A Service Definition Draft is the combination of a result file or **Result** object, information about the server, and a set of service properties. A **Result** object can be created in a Python script by setting a variable to a tool execution, for example, the following buffer result gets saved to a variable called result:

```
import arcpy
result = arcpy.Buffer_analysis("inPts", "output.shp", "100 Meters")
```

Result files can be created by right clicking a result in the Results window and choosing **Save As**.

Information about the server includes the server connection, 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 geoprocessing and additional capabilities of the service such as Web Processing Services (WPS). The capabilities are not exposed as a parameter. If you need to modify the value (or any nonexposed parameter), you need to publish the sddraft first and modify the draft by editing the .sddraft using XML libraries such as xml.dom.minidom. Please refer to the example of modifying an sddraft for the usage of the library. Although the example is from a map service draft, you can use the same library and method for the GP service draft since it is an XML file.

The function returns a Python dictionary containing errors and other potential issues that you should address prior to creating your Service Definition file.

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 the Upload 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 An overview of the Publishing toolset.

Syntax

CreateGPSDDraft (result, out_sddraft, service_name, {server_type}, {connection_file_path}, {copy_data_to_server}, {folder_name}, {summary}, {tags}, {executionType}, {resultMapServer}, {showMessages}, {maximumRecords}, {minInstances}, {maxInstances}, {maxUsageTime}, {maxWaitTime}, {maxIdleTime})

Parameter	Explanation	Data Type

result
[result,...]
	A reference to one or multiple Result objects or result files (.rlt) on disk. Multiple results must be supplied in a list format. The following example demonstrates multiple results as input to the CreateGPSDDraft function.
	`import arcpy`
	`r1 = arcpy.Buffer_analysis("inPts", "output.shp", "100 Meters")`
	`r2 = arcpy.GetCount_management("FireStations")`
	`arcpy.CreateGPSDDraft([r1, r2], "output.sddraft", "myservice")`
		Result

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 chooseFROM_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 (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).	String
copy_data_to_server	A Boolean that indicates whether the data referenced in the **result** 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	Boolean

	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 **result** is registered with the server, then **copy_data_to_server**will always be `False`. Conversely, if the data in the **result** is not registered with the server, then **copy_data_to_server** will always be `True`. (The default value is False)	
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. 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. 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
executionType	Asynchronous and synchronous define how the client (the application using the task) interacts with the server and gets the result from the task. When a service is set to synchronous, the client waits for the task to finish. Typically, a synchronous task executes quickly—five seconds or less. An asynchronous task typically takes longer to execute, and the client must periodically ask the server if the task has finished and, if it has finished, get the result. A web application using an asynchronous task must have logic implemented to check the status of a task and handle the result once execution is finished. ArcGIS Desktop clients handle both execution types natively. (The default value is Asynchronous)	String
resultMapServer	When publishing a geoprocessing service, you can choose to view the result of all tasks with the service as a map (in addition to other results of your task). The map is created on the server using a Map Service for transport back to the client as an image (a `.jpeg`, for example). The symbology, labeling, transparency, and all other properties of the returned map are the same as the settings of your output layer. Remember, if you are creating result layers within the Python scripting environment (outside ArcMap), default symbologies will be used. To maintain control over symbology you will need to pre-create layer files with rich symbology and use them to modify the output symbology of your task. When you choose this option, a map service is automatically created on the server with the same name as your geoprocessing service. (The default value is False)	Boolean
showMessages	A string setting the message level for the geoprocessing service. The following is a list of valid message levels the service will return to the client. • None —No geoprocessing messages are returned to the client, only whether the execution was successful or failed. • Error —Only tool messages that produce an error are returned to the client. • Warning —All tool error and warning messages are returned to the client. • Info —All tool messages from execution are returned to the client. (The default value is None)	String
maximumRecords	The maximum number of results the service can return to a client. Setting this value to a large number means your GIS server can handle sending a lot of individual records or features to the client. If you don't want to return any features, set this value to 0 (zero). Typically, you set this value to zero only when you enable **View result with a map service**. (The default value is 1000)	Integer
minInstances	An integer value representing the minimum number of instances a service will start and make available for use. For heavily used services you may want to increase this value. (The default value is 1)	Integer
maxInstances	An integer value representing the maximum number of instances a service can start and make available for use. For heavily used services you may need to increase this value. Ensure the server has adequate hardware to support the maximum number of instances you will allow. (The default value is 2)	Integer
maxUsageTime	The maximum time, in seconds, that a service can be used. You may need to increase the default of 600 seconds (10 minutes) for long-running geoprocessing tasks. Alternatively, you may need to reduce this time to ensure a client will not abuse your services. (The default value is 600)	Integer
maxWaitTime	The maximum time, in seconds, that a client will wait to connect with an instance before timing out. When all instances are busy processing requests, subsequent requests are queued. If this time-out elapses before an instance becomes available, the task will fail. The default is 60 seconds (1 minute). (The default value is 60)	Integer
maxIdleTime	The maximum time, in seconds, that an instance will continue to be active before pool shrinking occurs. Any instances above the minimum number of instances that have not been used will be shut down once the idle maximum time value has elapsed. (The default value is 1800)	Integer

Return Value

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

Code Sample

CreateGPSDDraft example 1

The following script demonstrates the complete publishing of a geoprocessing service through Python scripting. Automating the publishing of geoprocessing services uses a combination of ArcPy functions and geoprocessing tools in the Publishing toolset. The workflow begins by executing a geoprocessing task that you want to publish. After you have successfully run the geoprocessing task, and have saved a result file, use the ArcPy function CreateGPSDDraft to create a service definition draft. Note that the Item Description, Summary, and Tags for the input geoprocessing result 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 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 completeservice 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 geoprocessing service.

```
import arcpy

result = "c:/gis/gp/Analysis.rlt"
connectionPath = "c:/gis/conections/myServer.ags"
sddraft = "c:/gis/gp/drafts/AnalysisDraft.sddraft"
sd = "c:/gis/gp/sd/AnalysisDraft.sd"
serviceName = "AnalysisService"

# Create service definition draft
arcpy.CreateGPSDDraft(
    result, sddraft, serviceName, server_type="ARCGIS_SERVER",
    connection_file_path=connectionPath, copy_data_to_server=True,
    folder_name=None, summary="Analysis Service", tags="gp",
    executionType="Synchronous", resultMapServer=False,
    showMessages="INFO", maximumRecords=5000, minInstances=2,
    maxInstances=3, maxUsageTime=100, maxWaitTime=10,
    maxIdleTime=180)

# Analyze the service definition draft
analyzeMessages = arcpy.mapping.AnalyzeForSD(sddraft)

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

Continued on next page.

CreateGPSDDraft example 2

This example demonstrates the creation and publishing of the extract data workflow and ensures the data is registered with the server. The Extract Data task takes input layers and allows the user to extract an area of interest, convert to another output type and receive a zip file back as output. The Extract Data tool only uses layers as input. This example creates multiple layers using the Make Feature Layer tool, runs the tool, creates a service definition draft and uploads the service definition to be used in service creation. Any layer that was created in the Python session of the workflow will be considered as possible input in the final service and available as a choice selection.

```
import os
import arcpy

connPath = "c:/gis/conections/myServer.ags"
sddraft = "c:/gis/gp/drafts/ExtractionDraft.sddraft"
sd = "c:/gis/gp/sd/AnalysisDraft.sd"
serviceName = "DataExtractor"
arcpy.env.workspace = "c:/gis/citydata"
aoi = "c:/gis/citydata/extract.shp"

# Create layers which will be available as input
arcpy.MakeFeatureLayer_management('Wiarton.gdb/places/Cityhall',
                                  'CityHall')
arcpy.MakeFeatureLayer_management('Wiarton.gdb/places/Airport',
                                  'Airport')
arcpy.MakeFeatureLayer_management('Wiarton.gdb/places/FireStations',
                                  'FireStations')

# Run the extract data task and assign it to the 'result' variable
# only the cityhall layer was used as input, but the airport and
# firestation layers will be used in the service creation
result = arcpy.ExtractDataTask_server(
    "CityHall", aoi, "File Geodatabase - GDB - .gdb", "ESRI GRID -
GRID",
    os.path.join(arcpy.env.scratchFolder, "output.zip"))

# Make sure the folder is registered with the server, if not, add
# it to the datastore
if arcpy.env.workspace not in [
        i[2] for i in arcpy.ListDataStoreItems(connPath, 'FOLDER')]:
    # both the client and server paths are the same
    dsStatus = arcpy.AddDataStoreItem(
        connPath, "FOLDER", "CityData", arcpy.env.workspace,
        arcpy.env.workspace)
    print("Data store : {0}".format(dsStatus))

# Create service definition draft
arcpy.CreateGPSDDraft(
    result, sddraft, serviceName, server_type="ARCGIS_SERVER",
    connection_file_path=connPath, copy_data_to_server=False,
    folder_name=None, summary="Extraction Service",
    tags="extract data, clip")

# Analyze the service definition draft
analyzeMessages = arcpy.mapping.AnalyzeForSD(sddraft)
```

```
# Stage and upload the service if the sddraft analysis did not
```

```
# contain errors
if analyzeMessages['errors'] == {}:
    # Execute StageService
    arcpy.StageService_server(sddraft, sd)
    # Execute UploadServiceDefinition
    upStatus = arcpy.UploadServiceDefinition_server(sd, connPath)
    print("Completed upload")
else:
    # If the sddraft analysis contained errors, display them
    print(analyzeMessages['errors'])
```

Related Topics

AnalyzeForSD
CreateImageSDDraft
CreateMapSDDraft

CreateImageSDDraft (arcpy)

Top

Summary

The CreateImageSDDraft function is the first step to automating the publishing of a mosaic dataset or raster dataset as an Image Service using ArcPy. The output created from the **CreateImageSDDraft** is a Service Definition Draft (.sddraft) file, which is a combination of a mosaic dataset in the geodatabase or a raster dataset, information about the server, and a set of service properties. This service definition draft can be staged as service definition then uploaded to a specified ArcGIS server as an image service. 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).

Note:

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

Syntax

CreateImageSDDraft (raster_or_mosaic_layer, out_sddraft, service_name, {server_type}, {connection_file_path}, {copy_data_to_server}, {folder_name}, {summary}, {tags})

Parameter	Explanation	Data Type
raster_or_mosaic_layer	The raster layer or mosaic layer that you want to publish.	String
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 chooseFROM_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. (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`). (The default value is None)	String
copy_data_to_server	A Boolean that indicates whether the source data referenced by the mosaic dataset, the mosaic dataset itself, or the raster dataset published as an image service 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 workspace that contains the source data referenced by the mosaic dataset—the mosaic dataset itself or raster dataset registered with the server—then **copy_data_to_server** will always be**False**. Conversely, if the workspace that contains the source data referenced by the mosaic dataset—the mosaic dataset or raster dataset is not registered with the server—then **copy_data_to_server** will always be **True**. • False –The data will not be copied to the server. This is the default. • True –The data will be copied 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. Use this parameter to override the user interface summary or to provide a summary if one does not exist. (The default value is None)	String
tags	A string that represents the Item Description Tags. Use this parameter to override the user interface tags or to provide tags if they do not exist. (The default value is None)	String

Code Sample

CreateImageSDDraft example 1

Create an image service definition draft file.

```
import arcpy

ws = "C:/workspace"
mdpath = os.path.join(ws, "fgdb.gdb/mdDEM")
con = os.path.join(ws, "myserver_6080 (publisher).ags")
service = 'dem_service'
sddraft = os.path.join(ws, service + '.sddraft')

arcpy.CreateImageSDDraft(mdpath, sddraft, service, 'ARCGIS_SERVER',
                         con, True, None, "Publish las MD",
                         "las,image service")
```

CreateImageSDDraft example 2

Publish an image service from a mosaic dataset.

```
# It is recommended that you set the default mosaic dataset properly
before
# publishing. A connection to ArcGIS Server must be established in
the
# Catalog window of ArcMap before running this script

import arcpy
import os
import sys

# Define local variables:

# The folder for service definition draft and service definition
files
MyWorkspace = r"\\myserver\ArcPyPublishing"
Name = "OrthoImageService"
InputData = r"\\myserver\ArcPyPublishing\fgdb.gdb\ortho_images"
Sddraft = os.path.join(MyWorkspace, Name + ".sddraft")
Sd = os.path.join(MyWorkspace, Name + ".sd")
con = os.path.join(MyWorkspace, "arcgis on myserver_6080
(admin).ags")

# Create service definition draft
try:
    print("Creating SD draft")
    arcpy.CreateImageSDDraft(InputData, Sddraft, Name,
'ARCGIS_SERVER', con,
                             False, None, "Ortho Images",
                             "ortho images,image service")
except Exception as err:
    print(err[0] + "\n\n")
    sys.exit("Failed to create SD draft")

# Analyze the service definition draft
analysis = arcpy.mapping.AnalyzeForSD(Sddraft)
print("The following was returned during analysis of the image
service:")
for key in analysis.keys():

    print("---{}---".format(key.upper()))

    for ((message, code), layerlist) in analysis[key].iteritems():
        print("    {} (CODE {})".format(message, code))
        print("       applies to: {}".format(
            " ".join([layer.name for layer in layerlist])))

# Stage and upload the service if the sddraft analysis did not
contain errors
if analysis['errors'] == {}:
    try:
        print("Adding data path to data store to avoid copying data
to server")
        arcpy.AddDataStoreItem(con, "FOLDER", "Images", MyWorkspace,
                               MyWorkspace)

        print "Staging service to create service definition"
        arcpy.StageService_server(Sddraft, Sd)

        print "Uploading the service definition and publishing image
service"
        arcpy.UploadServiceDefinition_server(Sd, con)

        print "Service successfully published"
    except arcpy.ExecuteError:
        print(arcpy.GetMessages() + "\n\n")
        sys.exit("Failed to stage and upload service")

    except Exception as err:
        print(err[0] + "\n\n")
        sys.exit("Failed to stage and upload service")
else:
    print("Service was not published because of errors found during
analysis.")
    print(analysis['errors'])
```

Related Topics

AnalyzeForSD
CreateGPSDDraft
CreateMapSDDraft

CreateRandomValueGenerator (arcpy)

Top

Summary

Creates a new random number generator.

Syntax

CreateRandomValueGenerator (seed, distribution)

Parameter	Explanation	Data Type
seed	Initializes the random number generator.	Integer
distribution	The random generation algorithm. • ACM599 —ACM collected algorithm 599 • MERSENNE_TWISTER —Mersenne Twister mt19937 • STANDARD_C —Standard C Rand (The default value is ACM599)	String

Return Value

Data Type	Explanation
Object	The RandomNumberGenerator object.

Code Sample

CreateRandomValueGenerator example

Create and initialize random number generator object.

```
import arcpy

# CreateRandomValueGenerator takes 2 arguments, seed and distribution
# method. The distribution method options are ACM599,
# MERSENNE_TWISTER, and STANDARD_C.
#
# The gen variable is a randomNumberGenerator object that is assigned
# to the randomGenerator environments setting.
arcpy.env.randomGenerator = arcpy.CreateRandomValueGenerator(20,
"STANDARD_C")

# Calculate a random number using the ArcGIS.Rand() function
result = arcpy.CalculateValue_management("arcgis.rand('normal 0.0
10.0')")

# Print the returned value from the Result object
print(float(result.getOutput(0)))
```

Related Topics

RandomNumberGenerator
Calculate Value
Calculate Field
Create Random Points
Create Random Raster
Random Number Generator (Environment setting)

CreateScratchName (arcpy)

Top

Summary

Creates a unique scratch path name for the specified data type. If no workspace is given the current workspace is used.

Discussion

The scratch name will include a prefix of

Syntax

CreateScratchName ({prefix}, {suffix}, {data_type}, {workspace})

Parameter	Explanation	Data Type
prefix	The prefix that is added to the scratchname. By default, a prefix of xx is used. (The default value is xx)	String
suffix	The suffix added to the scratchname. This can be an empty double-quoted string.	String
data_type	The data type which will be used to create the scratchname. Valid datatypes are: • Coverage —Only valid Coverage names are returned. • Dataset —Only valid Dataset names are returned. • FeatureClass —Only valid FeatureClass names are returned. • FeatureDataset —Only valid FeatureDataset names are returned. • Folder —Only valid Folder names are returned. • Geodataset —Only valid Geodataset names are returned. • GeometricNetwork —Only valid Geometric Network names are returned. • ArcInfoTable —Only valid ArcInfo Table names are returned. • NetworkDataset —Only valid Network Dataset names are returned. • RasterBand —Only valid Raster Band names are returned. • RasterCatalog —Only valid Raster Catalog names are returned. • RasterDataset —Only valid Raster Dataset names are returned. • Shapefile —Only valid Shapefile names are returned. • Terrain —Only valid Terrain names are returned. • Workspace —Only valid Workspace scratchnames are returned.	String
workspace	The workspace used to determine the scratch name to be created. If not specified, the current workspace is used.	String

Return Value

Data Type	Explanation
String	The unique scratch path name.

Code Sample

CreateScratchName example

Create a unique scratch name for the derived output of the Buffer tool. This scratch name is then used as input to the Clip tool.

```
import arcpy

# Set workspace
#
arcpy.env.workspace = "C:/Data/Municipal.gdb"

# Create a scratch name for the Buffer tool output.
#   The scratch name created will be include 'temp0.shp',
#   If temp0.shp already exists, the number will be incremented
#   until the name is unique in the workspace.
#
scratch_name = arcpy.CreateScratchName("temp",
                                       data_type="Shapefile",

workspace=arcpy.env.scratchFolder)

# Execute Buffer tool, using scratch name for output
#
arcpy.Buffer_analysis("Roads", scratch_name, "1000 feet")

# Execute Clip tool, using scratch name for input
#
arcpy.Clip_analysis(scratch_name, "CityBoundary", "CityRoads")

# Delete scratch dataset
arcpy.Delete_management(scratch_name)
```

Related Topics

CreateUniqueName

CreateUniqueName (arcpy)

Top

Summary

Creates a unique name in the specified workspace by appending a number to the input name. This number is increased until the name is unique. If no workspace is specified, the current workspace is used.

Syntax

CreateUniqueName (base_name, {workspace})

Parameter	Explanation	Data Type
base_name	The base name used to create the unique name.	String
workspace	The workspace used for creation of the unique name.	String

Return Value

Data Type	Explanation
String	The unique name with a number appended to make it unique in the workspace. The number starts at 0 and is incremented until it is unique.

Code Sample

CreateUniqueName example

Creates a unique name for use with the Buffer and Clip tools.

```
import arcpy

# Set workspace
arcpy.env.workspace = "c:/data"

# Create a unique name for the Buffer tool's derived output.
unique_name = arcpy.CreateUniqueName("temp.shp")

# Use unique name for Buffer Tool output dataset name
arcpy.Buffer_analysis("roads.shp", unique_name, "100 feet")

# Clip output from Buffer tool with County Boundary to obtain
buffered roads
#  in county.
arcpy.Clip_analysis(unique_name, "county.shp", "clipped_roads.shp")
```

Related Topics

CreateScratchName

Describe (arcpy)

Top

Summary

The **Describe** function returns a Describe object, with multiple properties, such as data type, fields, indexes, and many others. Its properties are dynamic, meaning that depending on what data type is described, different describe properties will be available for use.

Describe properties are organized into a series of property groups. Any particular dataset will acquire the properties of at least one of these groups. For instance, if describing a geodatabase feature class, you could access properties from the GDB FeatureClass,FeatureClass, Table, and Dataset property groups. All data, regardless of the data type, will always acquire the generic Describe Object properties.

Discussion

Many data types include properties from other property groups. For instance, if describing a geodatabase feature class, you could access properties from the GDB FeatureClass, FeatureClass, Table, and Dataset property groups.

Note:

In some cases the object returned by **Describe** will not have all of the properties that are documented for it. For example, the describe object for a Layer in ArcMap's table of contents will not have the layer property set. That property only exists if you describe a `.lyr` file.

If you try to access a property that a **Describe** object does not have, it will either throw an error or return an empty value (None, 0 or -1, or empty string). If you are uncertain of a particular property, you can use Python's `hasattr()` function to check.

Describe Object Properties
ArcInfo Workstation Item Properties
ArcInfo Workstation Table Properties
CAD Drawing Dataset Properties
CAD FeatureClass Properties
Cadastral Fabric Properties
Coverage FeatureClass Properties
Coverage Properties
Dataset Properties
dBASE Table Properties
Editor Tracking Properties
FeatureClass Properties
File Properties

Continued on next page.

Folder Properties
GDB FeatureClass Properties
GDB Table Properties
Geometric Network Properties
LAS Dataset Properties
Layer Properties
Map Document Properties
Mosaic Dataset Properties
Network Analyst Layer Properties
Network Dataset Properties
Prj File Properties
Raster Band Properties
Raster Catalog Properties
Raster Dataset Properties
RecordSet and FeatureSet Properties
RelationshipClass Properties
RepresentationClass Properties
Schematic Dataset Properties
Schematic Diagram Properties
Schematic Folder Properties
SDC FeatureClass Properties
Shapefile FeatureClass Properties
Table Properties
TableView Properties
Text File Properties
Tin Properties
Tool Properties
Toolbox Properties
Topology Properties
VPF Coverage Properties
VPF FeatureClass Properties
VPF Table Properties
Workspace Properties

Syntax

Describe (value)

Parameter	Explanation	Data Type
value	The specified data element or geoprocessing object to describe.	String

Return Value

Data Type	Explanation
Describe	Returns an object with properties detailing the data element described. Some of the returned object's properties will contain literal values or objects.

Code Sample

Describe properties example (stand-alone script)

The following stand-alone script displays some layer and describe object properties from a layer set by a script parameter. The parameter may get set to either a .lyr file or to a layer in ArcMap.

```
import arcpy

# Get the layer as a parameter and describe it.
#
# The layer could be a layer in ArcMap (like "some_layer")
# Or, it could be a .lyr file (like "C:/data/some.lyr")
#
layerString = arcpy.GetParameterAsText(0)
desc = arcpy.Describe(layerString)

# Print selected layer and describe object properties
#
print "Name:", desc.name
if hasattr(desc, "layer"):
    print "Layer name:", desc.layer.name
    print "Layer data source:", desc.layer.catalogPath
    print ".lyr file:", desc.catalogPath
else:
    print "Layer name:", desc.name
    print "Layer data source:", desc.catalogPath

if desc.fidSet != '':
    print "Number of selected features:",
str(len(desc.fidSet.split(";")))
```

DisconnectUser (arcpy)

Top

Summary

Allows an administrator to disconnect users who are currently connected to an Enterprise geodatabase.

Discussion

The **DisconnectUser** function is used by an administrative user to disconnect users from an Enterprise geodatabase. This function is used to complement the Connected users dialog box found in ArcGIS for Desktop.

- The **DisconnectUser** function must utilize an administrative connection to the database.
- If this function is attempted to be run by a nonadministrative user the function will fail.
- Selecting all users will disconnect all users other than the administrator connection used to execute the function or database.

Syntax

DisconnectUser (sde_workspace, {users})

Parameter	Explanation	Data Type
sde_workspace	The Enterprise geodatabase containing the users to be disconnected. The connection properties specified in the Enterprise Geodatabase must have administrative rights that allow the user to disconnect other connections.	String
users [users,...]	Specifies which users will be disconnected from the geodatabase. • sde_id —The ID value returned from the ListUsers function or the Connections tab in the Geodatabase Administration dialog. This can be passed to the function as an individual sde_id or a Python list containing multiple sde_ids. • ALL —Keyword specifying that all connected users should be disconnected. **Note:** **DisconnectUser** will not disconnect the user who is executing the function.	Integer

Code Sample

DisconnectUser example 1

The following example demonstrates how to disconnect all users from a geodatabase.

```
import arcpy

arcpy.DisconnectUser("Database Connections/admin@sde.sde", "ALL")
```

DisconnectUser example 2

The following example demonstrates how to run the command to disconnect a single user. In this example, the SDE ID is extracted based on the name of the user that the admin would like to disconnect from the list returned from the ListUsers function

```
import arcpy

admin_workspace = "Database Connections/tenone@sde.sde"
arcpy.env.workspace = admin_workspace
user_name = "GDB"

# Look through the users in the connected user list and get the SDE
ID.
# Use the SDE ID to disconnect the user that matches the username
variable
users = arcpy.ListUsers() # The environment is set, no workspace is
needed.
for item in users:
    if item.Name == user_name:
        arcpy.DisconnectUser(admin_workspace, item.ID)
```

DisconnectUser example 3

The following example demonstrates how to run the DisconnectUser command to disconnect multiple users.

```
import arcpy

# Set the admistrative workspace connection
admin_workspace = "Database Connections/tenone@sde.sde"

# Create a list of users to disconnect.
user_names = ["TRAVIS", "DEBBIE", "PHIL"]

# Get a list of connected users
connected_users = arcpy.ListUsers(admin_workspace)

# Loop through the list of connected users and execute DisconnectUser
# if the user name is in the userNamesList created earlier:
for user in connected_users:
    if user.Name in user_names:
        print('Disconnecting {0}'.format(user.Name))
        arcpy.DisconnectUser(admin_workspace, user.ID)
```

Related Topics

ListUsers

Exists (arcpy)

Top

Summary

Determines the existence of the specified data object. Tests for the existence of feature classes, tables, datasets, shapefiles, workspaces, layers, and files in the current workspace. The function returns a Boolean indicating if the element exists.

Syntax

Exists (dataset)

Parameter	Explanation	Data Type
dataset	The name, path, or both of a feature class, table, dataset, layer, shapefile, workspace, or file to be checked for existence.	String

Return Value

Data Type	Explanation
Boolean	A Boolean value of True will be returned if the specified element exists.

Code Sample

Exists example

Check for existence of specified data object.

```
import arcpy

# Set the current workspace
#
arcpy.env.workspace = "c:/base/data.gdb"

# Check for existence of data before deleting
#
if arcpy.Exists("roadbuffer"):
    arcpy.Delete_management("roadbuffer")
```

Related Topics

Checking for the existence of data

FromWKB (arcpy)

Top

Summary

Create a new Geometry object from a well-known binary (WKB) string stored in a Python **bytearray**.

Syntax

FromWKB (byte_array)

Parameter	Explanation	Data Type
byte_array	A WKB string stored in a Python **bytearray**.	Bytearray

Return Value

Data Type	Explanation
Geometry	**FromWKB** returns a geometry object (PointGeometry, Multipoint, Polyline, or Polygon) based on the input bytearray.

Code Sample

FromWKB example

This example illustrates the use of converting a WKB to a geometry object.

```
import arcpy

fc = "c:/base/gdb.gdb/counties"

rows = arcpy.da.SearchCursor(fc, ['SHAPE@'])
studyarea = rows.next()[0]

# Create geometry to WKB
polyWKB = studyarea.WKB

# Convert WKB back to a geometry
polyGeom = arcpy.FromWKB(polyWKB)
```

Related Topics

FromWKT
AsShape

FromWKT (arcpy)

Top

Summary

Create a new Geometry object from a well-known text (WKT) string.

Syntax

FromWKT (wkt_string, {spatial_reference})

Parameter	Explanation	Data Type
wkt_string	A WKT string.	String
spatial_reference	The spatial reference of the geometry. It can be specified with either a SpatialReference object or string equivalent.	SpatialReference

Return Value

Data Type	Explanation
Geometry	**`FromWKT`** returns a geometry object (PointGeometry, Multipoint, Polyline, or Polygon) based on the input WKT string.

Related Topics

FromWKB

AsShape

GetArgumentCount (arcpy)

Top

Summary

Returns the number of arguments passed to the script.

Syntax

GetArgumentCount ()

Return Value

Data Type	Explanation
Integer	The number of arguments passed to the script.

Code Sample

GetArgumentCount example

Check for required number of arguments to run Clip tool and handle optional argument.

```
import sys
import arcpy

# Set workspace
arcpy.env.workspace = "c:/data/airport.gdb"

# Set local variables
in_features = arcpy.GetParameterAsText(0)
clip_features = arcpy.GetParameterAsText(1)
out_feature_class = arcpy.GetParameterAsText(2)
xy_tolerance = arcpy.GetParameterAsText(3)

# Check for required number of arguments
if arcpy.GetArgumentCount() < 3:
    print("3 arguments required for Clip_analysis tool")

# Execute Clip tool
try:
    arcpy.Clip_analysis(in_features, clip_features,
                        out_feature_class, xy_tolerance)
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

CopyParameter
GetParameter
GetParameterAsText
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

GetIDMessage (arcpy)

Top

Summary

Get the string of the error or warning ID message.

Discussion

GetIDMessage allows you to access ArcGIS tool message codes for use in Python. These message codes are found are documented in Tool errors and warnings.

Syntax

GetIDMessage (message_ID)

Parameter	Explanation	Data Type
message_ID	The geoprocessing message ID.	Integer

Return Value

Data Type	Explanation
String	The message string associated with the message ID.

Code Sample

GetIDMessage example

Access a message string using GetIDMessage and add to the tool messages using AddMessage.

```
import arcpy

message = arcpy.GetIDMessage(84001)

# The returned value should be:
#   u'Reading data....
arcpy.AddMessage(message)
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

GetInstallInfo (arcpy)

Top

Summary

The **GetInstallInfo** function returns a Python dictionary that contains information on the installation type properties.

Syntax

GetInstallInfo ()

Return Value

Data Type	Explanation
Dictionary	The function returns a Python dictionary containing the properties of the installation.

Keys	Values
SourceDir	Source directory
InstallDate	Date of installation
SPBuild	Service pack build (or N/A)
ProductName	Product installed (Desktop, Server, Engine)
BuildNumber	The build number
InstallType	The installation type (or N/A)
Version	The product version
SPNumber	Service pack build number (or N/A)
Installer	Account installed by
InstallDir	Installation location
InstallTime	Time of installation

GetInstallInfo's dictionary object keys

Code Sample

GetInstallInfo example

Return installation information.

```
import arcpy

# Use the dictionary iteritems to iterate through
#   the key/value pairs from GetInstallInfo
d = arcpy.GetInstallInfo()
for key, value in d.iteritems():
    # Print a formatted string of the install key and its value
    #
    print("{:<13} : {}".format(key, value))
```

GetInstallInfo example 2

Return the product version.

```
import arcpy

print(arcpy.GetInstallInfo()['Version'])
```

Related Topics

ListInstallations

Accessing licenses and extensions in Python

SetLogHistory (arcpy)

Top

Summary

For script tools and stand-alone scripts (scripts run outside of an ArcGIS application), you can enable or disable history logging using the **SetLogHistory** function.

The history log file is an Extensible Markup Language (XML) file that contains information about each geoprocessing operation. The information contained in the log file is essentially the same as that found in the Results window.

Syntax

SetLogHistory (log_history)

Parameter	Explanation	Data Type
log_history	True, to enable geoprocessing logging history and False, to disable.	Boolean

Code Sample

SetLogHistory example

Turn off geoprocessing log history.

```
import arcpy
arcpy.SetLogHistory(False)
```

Related Topics

GetLogHistory

Viewing tool execution history

GetMaxSeverity (arcpy)

Top

Summary

Gets the maximum severity returned from the last executed tool.

Syntax

GetMaxSeverity ()

Return Value

Data Type	Explanation
Integer	The returned severity. • 0 —no errors/warnings raised. • 1 —warning raised. • 2 —error raised.

Code Sample

GetMaxSeverity example

Displays a custom message based on maximum severity of the last executed command.

```
import arcpy

# Set current workspace
arcpy.env.workspace = "c:/data/mydata.gdb"

try:
    arcpy.Clip_analysis("Roads", "County", "Roads_Clip")
except arcpy.ExecuteError:
    pass

severity = arcpy.GetMaxSeverity()

if severity == 2:
    # If the tool returned an error
    print("Error occurred \n{0}".format(arcpy.GetMessages(2)))
elif severity == 1:
    # If the tool returned no errors, but returned a warning
    print("Warning raised \n{0}".format(arcpy.GetMessages(1)))
else:
    # If the tool did not return an error or a warning
    print(arcpy.GetMessages())
```

Related Topics

GetSeverity

GetMessage (arcpy)

Top

Summary

Returns a geoprocessing tool message by its index position.

Syntax

GetMessage (index)

Parameter	Explanation	Data Type
index	The message to retrieve.	Integer

Return Value

Data Type	Explanation
String	The geoprocessing tool message.

Code Sample

GetMessage example

Returns specified geoprocessing messages.

```
import arcpy

fc = arcpy.GetParameterAsText(0)

arcpy.GetCount_management(fc)

# Print the first and last message returned by the last
#  tool executed (GetCount)
message_count = arcpy.GetMessageCount()
print(arcpy.GetMessage(0))
print(arcpy.GetMessage(message_count - 1))
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessageCount
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

GetMessageCount (arcpy)

Top

Summary

Returns a numeric count of all the returned messages from the last executed command.

Syntax

GetMessageCount ()

Return Value

Data Type	Explanation
Integer	The count of returned messages from the last executed command.

Code Sample

GetMessageCount example

Returns the first and last geoprocessing messages.

```
import arcpy

fc = arcpy.GetParameterAsText(0)
arcpy.GetCount_management(fc)

# Print the first and last geoprocessing tool messages
message_count = arcpy.GetMessageCount()
print(arcpy.GetMessage(0))
print(arcpy.GetMessage(message_count - 1))
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessages
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

GetMessages (arcpy)

Top

Summary

Returns the geoprocessing messages from a tool by specified severity level.

Syntax

GetMessages ({severity})

Parameter	Explanation	Data Type
severity	The severity level of messages to return. • 0 –messages returned. • 1 –warning messages returned. • 2 –error messages returned. Not specifying a severity will return all types of messages. (The default value is 0)	Integer

Return Value

Data Type	Explanation
String	The geoprocessing tool messages, separated by a newline ('\n').

Code Sample

GetMessages example

Returns the geoprocessing messages.

```
import arcpy

fc = arcpy.GetParameterAsText(0)
arcpy.GetCount_management(fc)

# Print all of the geoprocessing messages returned by the
#  last tool (GetCount)
print(arcpy.GetMessages())
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
GetReturnCode
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

GetParameter (arcpy)

Top

Summary

From the parameter list, select the desired parameter by its index value. The parameter is returned as an object.

Discussion

To use the parameter as a text string instead, see GetParameterAsText.

Syntax

GetParameter (index)

Parameter	Explanation	Data Type
index	Selects the specified parameter, by its index, from the parameter list.	Integer

Return Value

Data Type	Explanation
Object	Object is returned from specified parameter.

Code Sample

GetParameter example

Get script tool parameter as object.

```
import arcpy

# Get the spatial reference from the tool dialog.
spatial_ref = arcpy.GetParameter(0)

# Display the Spatial Reference properties
arcpy.AddMessage("Name is: {0}".format(spatial_ref.name))
arcpy.AddMessage("Type is: {0}".format(spatial_ref.type))
arcpy.AddMessage("Factory code is:
{0}".format(spatial_ref.factoryCode))
```

Related Topics

CopyParameter
GetArgumentCount
GetParameterAsText
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

GetParameterAsText (arcpy)

Top

Summary

Gets the specified parameter as a text string by its index position from the list of parameters.

Discussion

Any value regardless of the parameter data type will be returned as a string; to use the parameter as an ArcPy or Python object instead, see GetParameter.

Syntax

GetParameterAsText (index)

Parameter	Explanation	Data Type
index	The numeric position of the parameter in the parameter list.	Integer

Return Value

Data Type	Explanation
String	The value of the specified parameter, returned as a string.

Code Sample

GetParameterAsText example

Get specified parameter as text string.

```
import os
import arcpy

# Set the input workspace, get the feature class name to copy
#  and the output location.
arcpy.env.workspace = arcpy.GetParameterAsText(0)
in_featureclass = arcpy.GetParameterAsText(1)
out workspace = arcpy.GetParameterAsText(2)

out_featureclass = os.path.join(out_workspace,
                                os.path.basename(in_featureclass))

# Copy feature class to output location
arcpy.CopyFeatures_management(in_featureclass, out_featureclass)
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

GetParameterCount (arcpy)

Top

Summary

Returns a count of the parameter values for the specified tool. If the tool is contained in a custom toolbox, use ImportToolbox to access the custom tool.

Syntax

GetParameterCount (tool_name)

Parameter	Explanation	Data Type
tool_name	The name of the tool for which the number of parameters will be returned.	String

Return Value

Data Type	Explanation
Integer	The number of parameters for the specified tool.

Code Sample

GetParameterCount example

Returns the number of tool parameters.

```
import arcpy

# Returns 7
print(arcpy.GetParameterCount("Buffer_analysis"))
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterAsText
GetParameterInfo
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

GetParameterInfo (arcpy)

Top

Summary

Returns a list of parameter objects for a given tool and is commonly used in a script tool's ToolValidator class.

Syntax

GetParameterInfo (tool_name)

Parameter	Explanation	Data Type
tool_name	The tool name. Including the toolbox alias will help to resolve any conflicts with duplicate tool names. **Note:** When the GetParameterInfo function is used as part of a script tool's ToolValidator class, the tool_name argument is optional.	String

Return Value

Data Type	Explanation
Parameter	Returns a list of parameter objects.

Code Sample

GetParameterInfo example 1

Display some parameter object properties for the specified tool.

```
import arcpy

# Load tool parameter objects into list.
params = arcpy.GetParameterInfo("HotSpots")

for param in params:
    print("Name: {}, Type: {}, Value: {}".format(
        param.name, param.parameterType, param.value))
```

GetParameterInfo example 2

Setting symbology for a script tool's output dataset.

```
import os
import arcpy

# Set data variables for Clip tool.
in_features = arcpy.GetParameterAsText(0)
clip_features = arcpy.GetParameterAsText(1)
out_feature_class = arcpy.GetParameterAsText(2)

# Execute Clip tool
output = arcpy.Clip_analysis(in_features, clip_features,
                             out_feature_class)[0]
```

```
# Get parameter objects

params = arcpy.GetParameterInfo()

# Use describe on result object and get shape type.
desc = arcpy.Describe(output)

# Set symbology property for out_feature_class parameter
# Layer files are located in same folder as the .py file
lyr_location = os.path.dirname(__file__)

if desc.shapeType == "Polygon":
    params[2].symbology = os.path.join(lyr_location, "Polygon.lyr")
elif desc.shapeType == "Polyline":
    params[2].symbology = os.path.join(lyr_location, "Polyline.lyr")
else:
    params[2].symbology = os.path.join(lyr_location, "Point.lyr")
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterAsText
GetParameterCount
GetParameterValue
SetParameter
SetParameterAsText
Setting script tool parameters

GetParameterValue (arcpy)

Top

Summary

For a specified tool name, returns the default value of the desired parameter.

Syntax

GetParameterValue (tool_name, index)

Parameter	Explanation	Data Type
tool_name	The tool name for which the parameter default value will be returned.	String
index	Index position of the parameter in the specified tool's parameter list.	Integer

Return Value

Data Type	Explanation
String	Returns the default value of the specified parameter for the tool.

Code Sample

GetParameterValue example

Returns the default value for specified tool parameter.

```
import arcpy

# Returns 'POLYGON'
print(arcpy.GetParameterValue("CreateFeatureClass_management", 2))
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterAsText
GetParameterCount
GetParameterInfo
SetParameter
SetParameterAsText
Setting script tool parameters

GetReturnCode (arcpy)

Top

Summary

Return the message error code by index.
If the message for the specified index is a warning or informative message the function will return a 0; if the message is an error the function will return a value other than 0.

Syntax

GetReturnCode (index)

Parameter	Explanation	Data Type
index	The specified position of the message in the returned list of messages, warnings, or errors.	Integer

Return Value

Data Type	Explanation
Integer	The return code of the message at the specified index position.

Code Sample

GetReturnCode example

Returns the severity code for the specified message.

```
import arcpy

arcpy.env.workspace = "c:/census/data.gdb"
arcpy.Erase_analysis("housing", "income", "low_income")

# Return the return code of the message in index
# position 3 (4th message)
print(arcpy.GetReturnCode(3))
```

Related Topics

AddError
AddIDMessage
AddMessage
AddReturnMessage
AddWarning
GetMessage
GetMessageCount
GetMessages
Writing messages in script tools
Understanding message types and severity
Understanding messages in script tools

GetSeverity (arcpy)

Top

Summary

Gets the severity code (0, 1, 2) of the specified message by index.

Syntax

GetSeverity (index)

Parameter	Explanation	Data Type
index	Numeric index position of the message in the stack.	Integer

Return Value

Data Type	Explanation
Integer	Severity code of the message. • 0 —message • 1 —warning • 2 —error

Code Sample

GetSeverity example

Returns the severity code and message for each geoprocessing tool message.

```
import arcpy

in_featureclass = arcpy.GetParameterAsText(0)
out_featureclass = arcpy.GetParameterAsText(1)

# Run the CopyFeatures tool. If it fails, print out the
# severity and message for each message.
try:
    arcpy.CopyFeatures_management(in_featureclass, out_featureclass)
except arcpy.ExecuteError:
    for i in xrange(0, arcpy.GetMessageCount()):
        print('{0}: {1}'.format(arcpy.GetSeverity(i),
                                arcpy.GetMessage(i)))
```

Related Topics

GetMaxSeverity

GetSeverityLevel (arcpy)

Top

Summary

Returns the severity level. The severity level is used to control how geoprocessing tools throw exceptions.

Syntax

GetSeverityLevel ()

Return Value

Data Type	Explanation
Integer	The severity level. • 0 —A tool will not throw an exception, even if the tool produces an error or warning. • 1 —If a tool produces a warning or an error, it will throw an exception. • 2 —If a tool produces an error, it will throw an exception. This is the default.

Code Sample

GetSeverityLevel example

Use SetSeverityLevel to force tool to throw an exception when a tool warning is encountered.

```
import arcpy

fc_1 = 'c:/resources/resources.gdb/boundary'
fc_2 = 'c:/resources/resources.gdb/boundary2'

# Set the severity level to 1 (tool warnings will throw an exception)
arcpy.SetSeverityLevel(1)
print("Severity is set to : {0}".format(arcpy.GetSeverityLevel()))

try:
    # FeatureCompare returns warning messages when a miscompare is
    #  found.  This normally would not cause an exception, however,
by
    #  setting the severity level to 1, all tool warnings will also
    #  return an exception.
    arcpy.FeatureCompare_management(fc_1, fc_2, "OBJECTID")

except arcpy.ExecuteWarning:
    print(arcpy.GetMessages(1))
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

SetSeverityLevel

GetSystemEnvironment (arcpy)

Top

Summary

Gets the specified system environment variable value, such as "TEMP".

Discussion

When using GetSystemEnvironment to retrieve the "TEMP" environment variable's value, GetSystemEnvironment will cycle through "TEMP", "TMP" and "MW_TMPDIR" environment variables and return the first value it finds.

Syntax

GetSystemEnvironment (environment)

Parameter	Explanation	Data Type
environment	The name of the system environment variable.	String

Return Value

Data Type	Explanation
String	Returns the value of the specified system environment variable as a string.

Code Sample

GetSystemEnvironment example

Return the specified system environment variable value.

```
import arcpy

# Set the scratchWorkspace environment to the value returned
# from the system environment variable TEMP
arcpy.env.scratchWorkspace = arcpy.GetSystemEnvironment("TEMP")
```

Related Topics

ListEnvironments

ImportToolbox (arcpy)

Top

Summary

Imports the specified toolbox into ArcPy, allowing for access to the toolbox's associated tools.

Discussion

While any of the core ArcGIS toolboxes are accessible by default in a script, your own custom or third-party toolboxes must be added using **ImportToolbox** to use them in a script.

Other toolboxes may be found in any number of different folders or geodatabases, and may have many different origins; they may be toolboxes you have personally created, or are toolboxes created internally by your organization, or are toolboxes you have downloaded from sites like the Geoprocessing Resource Center. In any case, these toolboxes need to be imported into ArcPy in a one-step process before they can be used as tools in Python.

Server toolboxes can also be added using a semicolon delimiter.

Server	Syntax
Internet ArcGIS for Server	URL;servicename;{username};{password}

Learn more about using a geoprocessing service in Python

Syntax

ImportToolbox (input_file, {module_name})

Parameter	Explanation	Data Type
input_file	The geoprocessing toolbox to be added to the ArcPy site package.	String
module_name	If the toolbox does not have an alias, the **module_name** is required. When a tool is accessed through the ArcPy site package, the toolbox alias where the tool is contained is a required suffix (arcpy.<toolname>_<alias>). Since ArcPy depends on toolbox aliases to access and execute the correct tool, aliases are extremely important when importing custom toolboxes. A good practice is to always define a custom toolbox's alias. However, if the toolbox alias is not defined, a temporary alias can be set as the second parameter.	String

Return Value

Data Type	Explanation
Module	Returns the imported module. If needed, tool names can be accessed from the module's **__all__** property.

Code Sample

ImportToolbox example

Import geoprocessing toolbox for use in ArcPy.

```
import arcpy

# Import custom toolbox
arcpy.ImportToolbox("c:/tools/My_Analysis_Tools.tbx")

try:
    # Run tool in the custom toolbox.  The tool is identified by
    #  the tool name and the toolbox alias.
    arcpy.GetPoints_myanalysis("c:/data/forest.shp")
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

Using tools in Python

InsertCursor (arcpy)

Top

Summary

Inserts rows into a feature class, shapefile, or table. The **InsertCursor** returns an enumeration object that hands out row objects.

Discussion

New row objects can be obtained using the **newRow** method on the enumeration object into which rows are to be inserted. Each call to **insertRow** on the cursor creates a new row in the table whose initial values are set to the values in the input row.

Syntax

InsertCursor (dataset, {spatial_reference})

Parameter	Explanation	Data Type
dataset	The table, feature class, or shapefile into which rows will be inserted.	String
spatial_reference	Coordinates are specified in the **spatial_reference** provided and converted on the fly to the coordinate system of the dataset.	SpatialReference

Return Value

Data Type	Explanation
Cursor	Returns a **Cursor** object against the specified feature class, shapefile, or table.

Code Sample

InsertCursor example

Inserts 25 new rows into a table.

```
import arcpy

# Create insert cursor for table
rows = arcpy.InsertCursor("c:/base/data.gdb/roads_lut")

# Create 25 new rows. Set the initial row ID and distance values
for x in xrange(1, 26):
    row = rows.newRow()
    row.setValue("rowid", x)
    row.setValue("distance", 100)
    rows.insertRow(row)

# Delete cursor and row objects to remove locks on the data
del row
del rows
```

Related Topics

Accessing data using cursors
UpdateCursor
SearchCursor

IsSynchronous (arcpy)

Top

Summary

Determines if a tool is running synchronous or asynchronous. When a tool is synchronous, the results are automatically returned, but no other action may be taken until the tool has completed. All non-server tools are synchronous. Server tools may beasynchronous, meaning that once the tool has been submitted to the server, other functionality can be run without waiting, and the results must be explicitly requested from the server.

Syntax

IsSynchronous (tool_name)

Parameter	Explanation	Data Type
tool_name	The name of the tool to determine if it is synchronous.	String

Return Value

Data Type	Explanation
Boolean	A return Boolean value of True indicates the tool is synchronous.

Code Sample

IsSynchronous example

Determine if a server tool is running synchronous.

```
import time
import arcpy

# Add server toolbox from a local ArcGIS for Server
arcpy.ImportToolbox("pondermatic;buffertools")

# Create and load a recordset object for the tool's input
record set = arcpy.RecordSet()
record_set.load("c:/temp/lines.shp")

# Run the server tool
results = arcpy.BufferLines_mytools(record_set, "100")

# If the tool is asynchronous, wait until the task is finished
(status = 4)
if not arcpy.IsSynchronous("BufferLines"):
    while results.status < 4:
        time.sleep(0.1)

# Get output from task and export to a feature class on disk
result = results.getOutput(0)
result.save("c:/temp/bufferlines.shp")
```

Related Topics

ListToolboxes
ListTools
Listing tools, toolboxes, and environment settings

ListDatasets (arcpy)

Top

Summary

Lists all of the datasets in a workspace. Search conditions can be specified for the dataset name and dataset type to limit the list that is returned.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListDatasets ({wild_card}, {feature_type})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String
feature_type	The feature type to limit the results returned by the wildcard argument. Valid dataset types are: • Coverage —Only coverages. • Feature —Coverage or geodatabase dataset, depending on the workspace. • GeometricNetwork —Only geometric network datasets. • Mosaic —Only mosaic datasets. • Network —Only network datasets. • ParcelFabric —Only parcel fabric datasets. • Raster —Only raster datasets. • RasterCatalog —Only raster catalog datasets. • Schematic —Only schematic datasets. • Terrain —Only terrain datasets. • Tin —Only TIN datasets. • Topology —Only topology datasets. • All —All datasets in the workspace. This is the default value. (The default value is All)	String

Return Value

Data Type	Explanation
String	A list containing dataset names returned from the function, limited by the wildcard and feature type arguments.

Code Sample

ListDatasets example

List Feature Dataset names that start with C.

```
import arcpy

arcpy.env.workspace = "c:/data"

# Print to the Interactive window all the feature datasets in the
#   workspace that start with the letter C.
datasets = arcpy.ListDatasets("C*", "Feature")

for dataset in datasets:
    print(dataset)
```

Related Topics

Create lists of data

ListDataStoreItems (arcpy)

Top

Summary

Returns a list of the folders or databases registered with an ArcGIS Server site.

Discussion

See About registering your data with the server to learn more about when and why you should register your data with ArcGIS Server.

Syntax

ListDataStoreItems (connection_file, datastore_type)

Parameter	Explanation	Data Type
connection_file	An ArcGIS Server connection file (.ags) for the server whose registered databases or folders you want to list. If you've made a connection in the***Catalog*** window of ArcMap, you can use the connection file found in your user profile directory. Alternatively, you can create a connection file from scratch using the function CreateGISServerConnectionFile.	String
datastore_type	The type of data that you want to list. • DATABASE —Enterprise databases registered with the server will be listed. • FOLDER —File-based data sources registered with the server will be listed.	String

Return Value

Data Type	Explanation
String	Returns the registered folders or databases as a list of lists of strings in the format [store_name, server_data, publisher_data, type]. • store_name—The alias of the folder or database as it is registered with the ArcGIS Server site. • server_data—When listing folders, the path to the folder as seen by the server. When listing databases, the connection properties as seen by the server. • publisher_data—When listing folders, the path to the folder as seen by the publisher's machine. When listing databases, the connection properties as seen by the publisher's machine. • type—If the publisher's machine and the server read the data out of the same physical location, this is `shared`. If the publisher and server read the data out of different physical locations, this is`replicated`. If the data location is registered as ArcGIS Server's Managed Database, this is `managed`.

Code Sample

ListDataStoreItems example

Prints all folders registered with the ArcGIS Server site.

```
import arcpy

print("Registered FOLDER items are:")

for item in arcpy.ListDataStoreItems("GIS Servers/MyConnection.ags",
"FOLDER"):
    print("Name: {}".format(item[0]))
    print("Server's path: {}".format(item[1]))
    print("Publisher's path: {}".format(item[2]))
    if item[3] == "managed":
        print("This is ArcGIS Server's Managed Database")
```

Related Topics

AddDataStoreItem
RemoveDataStoreItem
ValidateDataStoreItem

ListEnvironments (arcpy)

Top

Summary

The ListEnvironments function returns a list of geoprocessing environment names.

Syntax

ListEnvironments ({wild_card})

Parameter	Explanation	Data Type

wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.
	`import arcpy`
	
	`# A wild_card of "*workspace" will return a list including the`
	`#   workspace and scratchWorkspace environment names`
	`arcpy.ListEnvironments("*workspace")`
		String

Return Value

Data Type	Explanation
String	The list returned from the function containing the geoprocessing environment variable names, optionally limited by a wild card filter.

Code Sample

ListEnvironments example

Lists the environment variable name and current value.

```
import arcpy

environments = arcpy.ListEnvironments()

# Sort the environment names
environments.sort()

for environment in environments:
    # Format and print each environment and its current setting.
    # (The environments are accessed by key from arcpy.env.)
    print("{0:<30}: {1}".format(environment, arcpy.env[environment]))
```

Related Topics

Using environment settings in Python
ClearEnvironment
GetSystemEnvironment
ResetEnvironments
Listing tools, toolboxes, and environment settings

ListFeatureClasses (arcpy)

Top

Summary

Lists the feature classes in the workspace, limited by name, feature type, and optional feature dataset.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListFeatureClasses ({wild_card}, {feature_type}, {feature_dataset})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String
feature_type	The feature type to limit the results returned by the wild card argument. Valid feature types are: • Annotation —Only annotation feature classes are returned. • Arc —Only arc (or line) feature classes are returned. • Dimension —Only dimension feature classes are returned. • Edge —Only edge feature classes are returned. • Junction —Only junction feature classes are returned. • Label — Only label feature classes are returned. • Line —Only line (or arc) feature classes are returned. • Multipatch —Only multipatch feature classes are returned. • Node —Only node feature classes are returned. • Point —Only point feature classes are returned. • Polygon —Only polygon feature classes are returned. • Polyline —Only line (or arc) feature classes are returned. • Region —Only region feature classes are returned. • Route —Only route feature classes are returned. • Tic —Only tic feature classes are returned. • All — All datasets in the workspace. This is the default value. (The default value is All)	String
feature_dataset	Limits the feature classes returned to the feature dataset, if specified. If blank, only stand-alone feature classes will be returned in the workspace.	String

Return Value

Data Type	Explanation
String	The list containing feature class names is returned from the function, limited by the optional wild card, feature type, and feature dataset arguments.

Code Sample

ListFeatureClasses example

Copy shapefiles to a geodatabase.

```
import os
import arcpy

# Set the workspace for the ListFeatureClass function
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of
#  shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
    arcpy.CopyFeatures_management(
        fc, os.path.join("c:/base/output.gdb",
                         os.path.splitext(fc)[0]))
```

Related Topics

Create lists of data

ListFields (arcpy

Top

Summary

Lists the fields in a feature class, shapefile, or table in a specified dataset. The returned list can be limited with search criteria for name and field type and will contain field objects.

Syntax

ListFields (dataset, {wild_card}, {field_type})

Parameter	Explanation	Data Type
dataset	The specified feature class or table whose fields will be returned.	String
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned. (The default value is None)	String
field_type	The specified field type to be returned. Valid field types are: • All — All field types are returned. This is the default. • BLOB —Only field types of BLOB are returned. • Date —Only field types of Date are returned. • Double —Only field types of Double are returned. • Geometry —Only field types of Geometry are returned. • GlobalID —Only field types of GlobalID are returned. • GUID —Only field types of GUID are returned. • Integer —Only field types of Integer are returned. • OID —Only field types of OID are returned. • Raster —Only field types of Raster are returned. • Single —Only field types of Single are returned. • SmallInteger —Only field types of SmallInteger are returned. • String —Only field types of String are returned. (The default value is All)	String

Return Value

Data Type	Explanation
Field	A list containing Field objects is returned.

Code Sample

ListFields example

List field properties.

```
import arcpy

# For each field in the Hospitals feature class, print
#  the field name, type, and length.
fields = arcpy.ListFields("c:/data/municipal.gdb/hospitals")

for field in fields:
    print("{0} is a type of {1} with a length of {2}"
          .format(field.name, field.type, field.length))
```

ListFields example 2

Generate a list of field names.

```
import arcpy

featureclass = "c:/data/municipal.gdb/hospitals"
field_names = [f.name for f in arcpy.ListFields(featureclass)]
```

Related Topics

Create lists of data
Using fields and indexes
Field

ListFiles (arcpy)

Top

Summary

Returns a list of files in the current workspace based on a query string. Specifying search conditions can be used to limit the results.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListFiles ({wild_card})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String

Return Value

Data Type	Explanation
String	A list of files.

Code Sample

ListFiles example

Copy a list of CSV files to dBASE format.

```
import os
import arcpy

arcpy.env.workspace = "c:/temp"

# Copy each file with a .csv extension to a dBASE file
for csv_file in arcpy.ListFiles("*.csv"):
    # Use splitext to set the output table name
    dbase_file = os.path.splitext(csv_file)[0] + ".dbf"
    arcpy.CopyRows_management(csv_file, dbase_file)
```

Related Topics

Create lists of data

ListIndexes (arcpy)

Top

Summary

Lists the indexes in a feature class, shapefile, or table in a specified dataset. The list returned can be limited with search criteria for index name and will contain index objects.

Syntax

ListIndexes (dataset, {wild_card})

Parameter	Explanation	Data Type
dataset	The specified feature class or table whose indexes will be returned.	String
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String

Return Value

Data Type	Explanation
Index	A list containing Index objects is returned.

Code Sample

ListIndexes example

List index properties.

```
import arcpy

featureclass = "c:/data/roads.shp"

# Get list of indexes for roads.shp and print properties
indexes = arcpy.ListIndexes(featureclass)
for index in indexes:
    print("Name        : {0}".format(index.name))
    print("IsAscending : {0}".format(index.isAscending))
    print("IsUnique    : {0}".format(index.isUnique))
```

Related Topics

Create lists of data
Using fields and indexes
Index

ListInstallations (arcpy)

Top

Summary

The ListInstallations function returns a Python List of the installation types (server, desktop, and engine).

Syntax

ListInstallations ()

Return Value

Data Type	Explanation
String	The Python List containing installation names returned from the function.

Code Sample

ListInstallations example

Return a list of installation types on the computer.

```
import arcpy

for install in arcpy.ListInstallations():
    print(install)
```

Related Topics

GetInstallInfo

Accessing licenses and extensions in Python

ListPrinterNames (arcpy)

Top

Summary

Returns a list of available printer names.

Discussion

`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	Returns a list containing the printer names available to the script.

Code Sample

ListPrinterNames example

Returns a list of printer names available to session.

```
import arcpy

# Print available printers
printers = arcpy.ListPrinterNames()
for printer in printers:
    print(printer)
```

ListRasters (arcpy)

Top

Summary

Returns a list of the rasters in the workspace, limited by name and raster type.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListRasters ({wild_card}, {raster_type})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String
raster_type	The raster type to limit the results returned by the wild card argument. Valid raster types are: • BMP —Bitmap graphic raster dataset format. • GIF —Graphic Interchange Format for raster datasets. • IMG — ERDAS IMAGINE raster data format. • JP2 —JPEG 2000 raster dataset format. • JPG —Joint Photographics Experts Group raster dataset format. • PNG — Portable Network Graphics raster dataset format. • TIF —Tagged Image File for raster datasets. • GRID — Grid data format. • All —All supported raster types are returned. This is the default. (The default value is All)	String

Return Value

Data Type	Explanation
String	The list returned from the function containing raster names in the workspace, limited by the optional wild card and raster type.

Code Sample

ListRasters example

List Grid raster names in workspace.

```
import arcpy

# Set the current workspace
arcpy.env.workspace = "c:/data/DEMS"

# Get and print a list of GRIDs from the workspace
rasters = arcpy.ListRasters("*", "GRID")
for raster in rasters:
    print(raster)
```

Related Topics

Create lists of data

ListSpatialReferences (arcpy)

Top

Summary

Returns a Python list of available spatial reference names for use as an argument to **arcpy.SpatialReference**.

Syntax

ListSpatialReferences ({wild_card}, {spatial_reference_type})

Parameter	Explanation	Data Type
wild_card	Limit the spatial references listed by a simple wildcard check. The check is not case sensitive. For example, `arcpy.ListSpatialReferences("*Eckert*")` would list **Eckert I**, **Eckert II**, and so forth.	String
spatial_reference_type	Limit the spatial references listed by type. • GCS —List only Geographic Coordinate Systems. • PCS —List only Projected Coordinate Systems. • ALL —List both Projected and Geographic Coordinate Systems. This is the default. (The default value is All)	String

Return Value

Data Type	Explanation
String	A Python list of spatial references that match the wildcard and spatial reference type. Each item in the list includes qualifying information, separated with forward slashes, to help limit your search or better understand the purpose of the spatial reference. For example, `u'Projected Coordinate Systems/World/Sinusoidal (world)'` might be in the list. You can see from the path that this spatial reference is Sinusoidal, is a projected coordinate system, and is intended to be used at a global extent. Here is another example: `u'Projected Coordinate Systems/UTM/South America/Corrego Alegre UTM Zone 25S'`. This is a UTM spatial reference for a UTM zone in South America.

Code Sample

ListSpatialReferences example 1

List all geographic spatial references.

```
import arcpy

# Get the list of spatial references and print it.
srs = arcpy.ListSpatialReferences(spatial_reference_type="GCS")
for sr_name in srs:
    print sr_name
```

ListSpatialReferences example 2

Print the central meridians and names of UTM zones in New Zealand.

```
import arcpy

# Get the list of spatial references
srs = arcpy.ListSpatialReferences("*utm/new zealand*")

# Create a SpatialReference object for each one and print the
# central meridian
for sr_string in srs:
    sr_object = arcpy.SpatialReference(sr_string)
    print "{0.centralMeridian}   {0.name}".format(sr_object)
```

Related Topics

ListTransformations

ListTables (arcpy)

Top

Summary

Lists the tables in the workspace, limited by name and table type.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListTables ({wild_card}, {table_type})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String
table_type	The table type to limit the results returned by the wild card argument. Valid table types are: • dBASE —Only tables of type dBASE are returned. • INFO —Only stand-alone INFO tables are returned. • ALL —All stand-alone tables, including geodatabase tables, are returned. This is the default. (The default value is All)	String

Return Value

Data Type	Explanation
String	The list returned from the function containing table names in the workspace, limited by the optional wild card and table type.

Code Sample

ListTables example

List all table names in workspace.

```
import arcpy

# Set the current workspace
arcpy.env.workspace = "c:/data/mydata.gdb"

# Get and print a list of tables
tables = arcpy.ListTables()
for table in tables:
    print(table)
```

Related Topics

Create lists of data

ListToolboxes (arcpy)

Top

Summary

Lists the geoprocessing toolboxes, limited by name.

Syntax

ListToolboxes ({wild_card})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String

Return Value

Data Type	Explanation
String	The list returned from the function containing geoprocessing toolbox names, limited by the optional wild card.

Code Sample

ListToolboxes example

Lists specified toolboxes.

```
import arcpy

# Get and print a list of all toolboxes.
toolboxes = arcpy.ListToolboxes()

for toolbox in toolboxes:
    print(toolbox)
```

Related Topics

IsSynchronous
ListTools
Listing tools, toolboxes, and environment settings

ListTools (arcpy)

Top

Summary

Lists the geoprocessing tools, limited by name.

Syntax

ListTools ({wild_card})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String

Return Value

Data Type	Explanation
String	The list returned from the function containing geoprocessing tool names, limited by the optional wild card.

Code Sample

ListTools example

Lists all tools in the specified toolbox.

```
import arcpy

# Create a list of tools in the Analysis toolbox
tools = arcpy.ListTools("*_analysis")

# Loop through the list and print each tool's usage.
for tool in tools:
    print(arcpy.Usage(tool))
```

Related Topics

IsSynchronous
ListToolboxes
Listing tools, toolboxes, and environment settings

ListTransformations (arcpy)

Top

Summary

Returns a list of valid transformation methods for converting data from one spatial reference to another. An extent can be used to narrow the list of valid transformation methods for a specific geographic area.

Syntax

ListTransformations (from_sr, to_sr, {extent})

Parameter	Explanation	Data Type
from_sr	The starting geographic coordinate system. Can be specified with a **SpatialReference** object, the name of the spatial reference, or a path to a projection file (.prj).	SpatialReference
to_sr	The final geographic coordinate system. Can be specified with a **SpatialReference** object, the name of the spatial reference, or a path to a projection file (.prj).	SpatialReference
extent	Only transformations that span the entire extent will be returned. The extent needs to be specified in coordinates from the **in_sr**. When working with data, the extent on a **Describe** object can be used.	Extent

Return Value

Data Type	Explanation
String	A list of valid transformation methods.

Code Sample

ListTransformations example

Use ListTransformations to identify valid transformations for projecting from one coordinate system to another.

```
import arcpy

from_sr = arcpy.SpatialReference('WGS 1984')
to_sr = arcpy.SpatialReference('NAD 1927 StatePlane California VI
FIPS 0406')

extent = arcpy.Extent(-178.217598182, 18.9217863640001,
                      -66.969270909, 71.4062354550001)
transformations = arcpy.ListTransformations(from_sr, to_sr, extent)
```

Related Topics

Choosing an appropriate transformation
ListSpatialReferences

ListUsers (arcpy)

Top

Summary

Returns a list of named tuples containing information for users who are connected to an enterprise geodatabase.

Discussion

The ListUsers function is used by an administrative user to identify users who are currently connected to an enterprise geodatabase.

- The **`ListUsers`** function must utilize an administrative connection to the database.
- This function will fail if attempted by a nonadministrative user.

Syntax

ListUsers (sde_workspace)

Parameter	Explanation	Data Type
sde_workspace	An enterprise geodatabase (sde connection file). The connection properties specified in the enterprise geodatabase must have administrative rights that allow the user to disconnect other connections.	String

Return Value

Data Type	Explanation
tuple	The ListUsers function returns a list of named tuples. The named tuples returned each have the following five items:

Name	Datatype	Description
ClientName	String	Name of the client machine from where the connection is being made.
ConnectionTime	Datetime	The time when the user made the initial connection to the geodatabase.
ID	Integer	The SDE connection identifier. This is the value that can be used in the **`DisconnectUser`** function to identify the user to disconnect.
IsDirectConnection	Boolean	True if the connection is a direct connection (two-tier). False if the connection is an application server connection (three-tier).
Name	String	The name of the user who is connected to the database.

Code Sample

ListUsers example 1

```
import arcpy

arcpy.ListUsers("Database Connections/admin.sde")
```

ListUsers example 2

The following example demonstrates how to print a list of connected users along with their connection time.

```
import arcpy

users = arcpy.ListUsers("Database Connections/admin.sde")
for user in users:
    print("Username: {0}, Connected at: {1}".format(
        user.Name, user.ConnectionTime))
```

ListUsers example 3

The following example demonstrates how to generate a new list of only SDE IDs from the list returned by ListUsers.

```
import arcpy

# Set the admistrative workspace connection
arcpy.env.workspace = "Database Connections/tenone@sde.sde"

# Create a list of users
'''
NOTE: When the arcpy.env.workspace environment is set, a workspace
does not need to be provided to the function.
'''
users = arcpy.ListUsers()

# Create a list of SDE ID's.
# Use a list comprehension to get the ID values in a new list.
id_users = [user.ID for user in users]
print(id_users)
```

Related Topics

AcceptConnections
DisconnectUser

ListVersions (arcpy)

Top

Summary

Lists the versions the connected user has permission to use.

Discussion

You can specify the path to an ArcSDE connection file as an argument to the function or you can set the workspace environment to the ArcSDE Connection file and call the **ListVersions** function without any arguments. Only those versions the connected user has permissions to use will be included in the list returned by the function.

Note:

The **arcpy.ListVersions** function should not be confused with the **arcpy.da.ListVersions** function which is used to return a list of Version objects.

Syntax

ListVersions (sde_workspace)

Parameter	Explanation	Data Type
sde_workspace	An ArcSDE geodatabase workspace.	String

Return Value

Data Type	Explanation
String	The returned from the function containing version names in the ArcSDE geodatabase the connected user has permissions to use.

Code Sample

ListVersions example

Gets the list of versions the user has permissions to use and prints them.

```
import arcpy

database = "Database Connections/toolboxDEFAULTVersion.sde"
versions = arcpy.ListVersions(database)

# Print the versions available to the user
for version in versions:
    print(version)
```

Related Topics

Create lists of data

ListWorkspaces (arcpy)

Top

Summary

Lists all of the workspaces within the set workspace. Search conditions can be specified for the workspace name and workspace type to limit the list that is returned.

Discussion

The workspace environment must be set first before using several of the List functions,
including ListDatasets, ListFeatureClasses, ListFiles, ListRasters, ListTables, and ListWorkspaces.

Syntax

ListWorkspaces ({wild_card}, {workspace_type})

Parameter	Explanation	Data Type
wild_card	The wild card limits the results returned. If no wild card is specified, all values are returned.	String
workspace_type	The workspace type to limit the results returned by the wild card argument. There are six possible workspace types: • Access —Only personal geodatabases will be selected. • Coverage —Only coverage workspaces will be selected. • FileGDB —Only file geodatabases will be selected. • Folder —Only shapefile workspaces will be selected. • SDE —Only ArcSDE databases will be selected. • All —All workspaces will be selected. This is the default. (The default value is All)	String

Return Value

Data Type	Explanation
String	The list containing workspace names is returned from the function, limited by the wild card and workspace type arguments.

Code Sample

ListWorkspaces example

Compact all File Geodatabases in workspace.

```
import arcpy

arcpy.env.workspace = "c:/data"

# List all file geodatabases in the current workspace
workspaces = arcpy.ListWorkspaces("*", "FileGDB")

for workspace in workspaces:
    # Compact each geodatabase
    arcpy.Compact_management(workspace)
```

Related Topics

Create lists of data

LoadSettings (arcpy)

Top

Summary

Loads environment settings from an environment settings file (text stored in an Extensible Markup Language [XML] schema). See also SaveSettings on how to save environment settings.

Syntax

LoadSettings (file_name)

Parameter	Explanation	Data Type
file_name	An existing XML file that contains environment settings.	String

Code Sample

LoadSettings example

Load settings from environment settings XML file.

```
import arcpy

# Load previously saved environment settings
#
arcpy.LoadSettings("C:/Data/MyCustomSettings.xml")
```

Related Topics

SaveSettings

NumPyArrayToRaster (arcpy)

Top

Summary

Converts a NumPy array to a raster.

Discussion

The size and data type of the resulting raster dataset depends on the input array.
Having the x_cell_size and the y_cell_size arguments allows support for rectangular cells.
This function honors the following geoprocessing environment settings: Output Coordinate System, Extent, Snap Raster, Current Workspace, Scratch Workspace

Syntax

NumPyArrayToRaster (in_array, {lower_left_corner}, {x_cell_size}, {y_cell_size}, {value_to_nodata})

Parameter	Explanation	Data Type
in_array	The NumPy array to convert to a raster.	NumPyArray
lower_left_corner	The lower left corner of the output raster to position the NumPy array. The X and Y values are in map units. (The default value is 0.0)	Point
x_cell_size	The cell size in the x direction specified in map units. The input can be a specified cell size (type: double) or an input raster. When a dataset is input for the x_cell_size, the x cell size of the dataset is used for the x cell size for the output raster. If only the x_cell_size is identified and not the y_cell_size, a square cell will result with the specified size. If neither **x_cell_size** or y_cell_size are specified, a default of 1.0 will be used for both the x and y cell size. (The default value is 1.0)	Double
y_cell_size	The cell size in y direction specified in map units. The input can be a specified cell size (type: double) or an input raster. When a dataset is input for the y_cell_size the y cell size of the dataset is used for the y cell size for the output raster. If only the y_cell_size is identified and not the x_cell_size, a square cell will result with the specified size. If neither x_cell_size or y_cell_size are specified, a default of 1.0 will be used for both the x and y cell size. (The default value is 1.0)	Double
value_to_nodata	The value in the NumPy array to assign to NoData in the output raster. If no value is specified for value_to_nodata, there will not be any NoData values in the resulting raster.	Double

Return Value

Data Type	Explanation
Raster	The output raster.

Code Sample

NumPyToRaster example

A new raster is created from a randomly generated NumPy array.

```
import numpy
import arcpy

my_array = numpy.random.random_integers(0, 100, 2500)
my array.shape = (50, 50)
my_raster = arcpy.NumPyArrayToRaster(my_array)
my_raster.save("c:/output/fgdb.gdb/myRandomRaster")
```

Related Topics

RasterToNumPyArray
Working with NumPy in ArcGIS

ParseFieldName (arcpy)

Top

Summary

Parses a fully qualified field name into its components (database, owner name, table name, and field name) depending on the workspace. ParseFieldName returns a string containing the parsed table name, containing the database, owner, table, and field names separated by commas. The workspace must be a personal, file, or ArcSDE geodatabase.

Syntax

ParseFieldName (name, {workspace})

Parameter	Explanation	Data Type
name	The field name to be parsed.	String
workspace	Specifies the workspace for fully qualifying the field name. The workspace must be a personal, file, or ArcSDE geodatabase.	String

Return Value

Data Type	Explanation
String	Returns the field name parsed into its components (owner name, database name, table name, field name) separated by commas.

Code Sample

ParseFieldName example

Returns a fully qualified field name parsed into its components.

```
import arcpy

# Get the name of the input field and parse it.
#
field_name = arcpy.GetParameterAsText(0)
workspace = arcpy.GetParameterAsText(0)

# Create a list and populate it.
#
fullname = arcpy.ParseFieldName(field_name, workspace)
database, owner, featureclass = fullname.split(",")

# Qualify the name of the feature class that will be appended to and
set
#   the workspace using the administrator's connection.
#
arcpy.env.workspace = "Database Connections/Trans_admin.sde"
append_fc = arcpy.ValidateTableName("common", "roads")

try:
    if owner in ["ted", "laura"]:
        arcpy.CalculateField management(
            fullname, "AppendedBy", owner, "PYTHON_9.3")
            arcpy.Append_management(fullname, append_fc)
    else:
        arcpy.AddError("Unknown user of input feature class")
except arcpy.ExecuteError:
    arcpy.AddError(arcpy.GetMessages(2))
```

Related Topics

ParseTableName
ValidateFieldName
Validating table and field names in Python

ParseTableName (arcpy)

Top

Summary

Parses a table name into its components (database, owner, table) depending on the workspace. ParseTableName returns a string containing the parsed table name, with the database name, owner name, and table name separated by commas. This workspace must be a personal, file, or ArcSDE geodatabase.

Syntax

ParseTableName (name, {workspace})

Parameter	Explanation	Data Type
name	Specifies which table will be parsed.	String
workspace	Specifies the workspace for fully qualifying the table name. The workspace must be a personal, file, or ArcSDE geodatabase.	String

Return Value

Data Type	Explanation
String	Returns the table name parsed into its components (owner name, database name, table name) separated by commas.

Code Sample

ParseTableName example

Parse a table name into its components.

```
import arcpy

# Get the name of the input field and parse it.
#
field_name = arcpy.GetParameterAsText(0)
workspace = arcpy.GetParameterAsText(0)

# Create a list and populate it.
#
fullname = arcpy.ParseTableName(field_name, workspace)
database, owner, featureclass = fullname.split(",")

# Qualify the name of the feature class that will be appended to and
set
#   the workspace using the administrator's connection.
#
arcpy.env.workspace = "Database Connections/Trans admin.sde"
append_fc = arcpy.ValidateTableName("common", "roads")

try:
    if owner in ["ted", "laura"]:
        arcpy.CalculateField_management(
            fullname, "AppendedBy", owner, "PYTHON_9.3")
        arcpy.Append_management(fullname, append_fc)
    else:
        arcpy.AddError("Unknown user of input feature class")
except arcpy.ExecuteError:
    arcpy.AddError(arcpy.GetMessages(2))
```

Related Topics

ValidateTableName
ParseFieldName
Validating table and field names in Python

ProductInfo (arcpy)

Top

Summary

Returns the current product license.

Syntax

ProductInfo ()

Return Value

Data Type	Explanation
String	• NotInitialized —No license set • ArcView —ArcGIS for Desktop Basic product license set • ArcEditor —ArcGIS for Desktop Standard product license set • ArcInfo —ArcGIS for Desktop Advanced product license set • Engine —Engine runtime license set • EngineGeoDB —Engine Geodatabase Update license set • ArcServer —Server license set

Code Sample

ProductInfo example

Returns the current product license.

```
import arcview
import arcpy

print(arcpy.ProductInfo()) # prints ArcView
```

Related Topics

CheckProduct
SetProduct
Accessing licenses and extensions in Python

RasterToNumPyArray (arcpy)

Top

Summary

Converts a raster to a NumPy array.

Discussion

A Python NumPy array is designed to deal with large arrays. There are many existing Python functions that have been created to process NumPy arrays, the most noted being contained in the SciPy scientific computing package for Python. You may want to convert an ArcGIS raster to a NumPy array to

1. Implement one of the many existing Python functions that can be applied to a NumPy array (for example, run filters on the data, perform multidimensional analysis, or utilize optimization routines).
2. Develop a custom function by accessing the individual cells within the NumPy array (for example, to implement neighborhood notation, change individual cell values, or run accumulative operators on an entire raster).

If the array definition (the lower left corner and the number of rows and columns) exceeds the extent of the **in_raster**, the array values will be assigned NoData.

If the **lower_left_corner** does not coincide with the corner of a cell, it will automatically be snapped to the lower left of the nearest cell corner applying the same rules as the Snap Raster environment setting. This snapping action within the **RasterToNumPy**function is not to be confused with the Snap Raster environment setting; the function only uses the same interaction; see:

Learn more about how the Snap Raster environment works

Syntax

RasterToNumPyArray (in_raster, {lower_left_corner}, {ncols}, {nrows}, {nodata_to_value})

Parameter	Explanation	Data Type
in_raster	The input raster to convert to a NumPy array.	Raster
lower_left_corner	The lower left corner within the **in_raster** from which to extract the processing block to convert to an array. The x- and y-values are in map units. (The default value is origin of inRaster)	Point
ncols	The number of columns from the **lower_left_corner** in the **in_raster** to convert to the NumPy array. (The default value is number of columns in inRaster)	Integer
nrows	The number of rows from the **lower_left_corner** in the **in_raster** to convert to the NumPy array. (The default value is number of rows in inRaster)	Integer
nodata_to_value	The value to assign the **in_raster** NoData values in the resulting NumPy array. The data type depends on the type of the **in_raster**. If no value is specified, the NoData values in **in_raster** will be assigned the value associated with NoData in **in_raster**.	Variant

Return Value

Data Type	Explanation
NumPyArray	The output NumPy array.

Code Sample

RasterToNumPy example

A raster is converted to a NumPy array to calculate the percentage of the cell value in the entire raster row. A new raster is then created.

```
import arcpy
import numpy

my_array = arcpy.RasterToNumPyArray('C:/data/inRaster')
my_array_sum = my_array.sum(1)
my_array_sum.shape = (my_array.shape[0], 1)
my_array_perc = (my_array * 1.0) / my_array_sum
new_raster = arcpy.NumPyArrayToRaster(my_array_perc)
new_raster.save("C:/output/fgdb.gdb/PercentRaster")
```

Related Topics

Working with NumPy in ArcGIS

NumPyArrayToRaster

RefreshActiveView (arcpy)

Top

Summary

Refreshes the active view and table of contents of the current map document.

Discussion

RefreshActiveView is only needed if you want to see the active view of the current map document updated. **arcpy.mapping** export, save, and printing functions will generate the expected updated results without use of **RefreshActiveView**.

Syntax

RefreshActiveView ()

Code Sample

RefreshActiveView example

Refresh the current map to reflect **zoomToSelectedFeatures**.

```
import arcpy

mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd, "Layers")[0]
lyr = arcpy.mapping.ListLayers(mxd, "Cities", df)[0]

# Use the SelectLayerByAttribute tool to select New York and
#  zoom to the selection
arcpy.SelectLayerByAttribute_management(lyr, "NEW_SELECTION",
                                        "CITY_NAME = 'New York'")
df.zoomToSelectedFeatures()

# Export the map to a .jpg
arcpy.mapping.ExportToJPEG(mxd, "C:/data/NewYork.jpg")

# Clear the selection and refresh the active view
arcpy.SelectLayerByAttribute_management(lyr, "CLEAR_SELECTION")
arcpy.RefreshActiveView()

del mxd
```

Related Topics

RefreshCatalog

RefreshCatalog (arcpy)

Top

Summary

Forces a refresh of the Catalog window or Catalog tree.

Discussion

The Catalog window may not always display the latest state of all information. In these cases, it is useful to refresh your content. For instance, intermediate data created within a script tool, or using other python modules to create or move data will not be automatically shown in the Catalog window.

Syntax

RefreshCatalog (dataset)

Parameter	Explanation	Data Type
dataset	Data element to be refreshed.	String

Code Sample

RefreshCatalog example

Forces a refresh after using shutil.copytree to copy a directory.

```
import shutil
import arcpy

input_folder = "c:/data/hydrology"
target_folder = "c:/test/hydro_backup"

# Copy a directory tree to a backup location
shutil.copytree(input_folder, target_folder)

# Refresh the Catalog window for the new directory
arcpy.RefreshCatalog(target_folder)
```

Related Topics

RefreshActiveView

RefreshTOC (arcpy)

Top

Summary

Refreshes the table of contents.

Discussion

If Python is used to modify the table of contents within the current map document (e.g., change a layer name), the map will not automatically update with the changes. **RefreshTOC** is only needed if you want to see the table of contents updated.

Syntax

RefreshTOC ()

Code Sample

RefreshTOC example

A simple script run from the Python window that demonstrates how to update the Table of Contents and force a refresh to make the change visible.

```
import arcpy

# Create a MapDocument object from the current map
mxd = arcpy.mapping.MapDocument("CURRENT")

# Update the layer name of the first layer in the Table of Contents
arcpy.mapping.ListLayers(mxd)[0].name = "New Layer Name"

# Refresh the Table of Contents to reflect the change
arcpy.RefreshTOC()

del mxd
```

RemoveDataStoreItem (arcpy)

Top

Summary

Unregisters a folder or database from an ArcGIS Server site.

Syntax

RemoveDataStoreItem (connection_file, datastore_type, connection_name)

Parameter	Explanation	Data Type
connection_file	An ArcGIS Server connection file (.ags) for the server whose database or folder is being unregistered. If you've made a connection in ArcCatalog, you can use the connection file found in your user profile directory. Alternatively, you can create a connection file from scratch using the functionCreateGISServerConnectionFile.	String
datastore_type	The type of data being unregistered. • DATABASE —The data resides in an enterprise database. • FOLDER —The data is file-based.	String
connection_name	The name of the folder or database being unregistered, as it is currently registered with the ArcGIS Server site.	String

Code Sample

RemoveDataStoreItem example

Unregisters a folder from ArcGIS Server that was using the alias "My local data folder".

```
import arcpy

arcpy.RemoveDataStoreItem("GIS Servers/MyConnection.ags", "FOLDER",
                          "My local data folder")
```

Related Topics

AddDataStoreItem
ListDataStoreItems
ValidateDataStoreItem

RemoveToolbox (arcpy)

Top

Summary

Removes the specified toolbox, either by specifying its path or referencing its alias. Removes the specified toolbox from the current geoprocessing session. Server toolboxes can also be removed using a semicolon delimiter.

Discussion

Caution:

RemoveToolbox only removes tools from the geoprocessor object; tools are not directly removed from arcpy.

Syntax

RemoveToolbox (toolbox)

Parameter	Explanation	Data Type
toolbox	The name of the toolbox, including either path or alias, to be removed from the current geoprocessing session. The name/path or alias should be placed in a double-quoted string. Server toolboxes can be removed using a semicolon delimiter. • The name, including path, or alias, of the toolbox to be removed from the current geoprocessing session. Place the name/path, or alias, string inside double quotes. Server toolboxes can also be removed using a semicolon delimiter. ▪ Syntax for Internet ArcGIS for Server ○ *URL servername;{username};{password}* ▪ Syntax for Local ArcGIS for Server ○ *machinename;servername*. ▪ Syntax for Internet ArcGIS for Server ○ *URL;servername;{username};{password}* ▪ Syntax for Local ArcGIS for Server ○ *machinename;servername*	String

Code Sample

RemoveToolbox example

Removes the specified toolbox from access by current geoprocessing session.

```
import arcpy

# Remove a toolbox from session
arcpy.RemoveToolbox("c:/mytoolboxes/operations.tbx")
```

Related Topics

IsSynchronous
ListToolboxes
ListTools
Listing tools, toolboxes, and environment settings

ResetEnvironments (arcpy)

Top

Summary

Resets all environment settings to their default settings.

Syntax

ResetEnvironments ()

Code Sample

ResetEnvironments example

Resets all environment settings to their default values.

```
import arcpy

# Reset environment settings to default settings.
#
arcpy.ResetEnvironments()
```

Related Topics

Using environment settings in Python
ClearEnvironment
GetSystemEnvironment
ListEnvironments

ResetProgressor (arcpy)

Top

Summary

Resets the progressor back to its initial state.

Syntax

ResetProgressor ()

Code Sample

ResetProgressor example

Reset progress dialog box to initial state.

```
import os
import arcpy

# Allow overwriting of output
arcpy.env.overwriteOutput = True

# Set current workspace
arcpy.env.workspace = "c:/data"

# Get a list of shapefiles in folder
fcs = arcpy.ListFeatureClasses()

# Find the total count of shapefiles in list
fc_count = len(fcs)

# Set the progressor
arcpy.SetProgressor("step", "Copying shapefiles to geodatabase...",
                    0, fc_count, 1)

# Create a file gdb to contain new feature classes
arcpy.CreateFileGDB_management(arcpy.env.workspace, "fgdb.gdb")

# For each shapefile, copy to a file geodatabase
for shp in fcs:
    # Trim the '.shp' extension
    fc = os.path.splitext(shp)[0]

    # Update the progressor label for current shapefile
    arcpy.SetProgressorLabel("Loading {0}...".format(shp))

    # Copy the data
    arcpy.CopyFeatures_management(shp, os.path.join("fgdb.gdb", fc))

    # Update the progressor position
    arcpy.SetProgressorPosition()

arcpy.ResetProgressor()
```

Related Topics

SetProgressor
SetProgressorLabel
SetProgressorPosition
Controlling the progress dialog box

SaveSettings (arcpy)

Top

Summary

Saves environment settings to an environment settings file (text stored in an Extensible Markup Language [XML] schema). See also LoadSettings on how to load environment settings from an XML file.

Syntax

SaveSettings (file_name)

Parameter	Explanation	Data Type
file_name	The XML file to be created that will store the current environment settings.	String

Code Sample

SaveSettings example

Save environment settings to an XML file.

```
import arcpy

arcpy.env.workspace = "c:/data/mydata.gdb"
arcpy.env.cellSize = 28
arcpy.env.compression = "LZ77"

# Save environment settings to XML file
arcpy.SaveSettings("c:/data/mycustomsettings.xml")
```

Related Topics

LoadSettings

SearchCursor (arcpy)

Top

Summary

The **SearchCursor** function establishes a read-only cursor on a feature class or table. The **SearchCursor** can be used to iterate through row objects and extract field values. The search can optionally be limited by a where clause or by field, and optionally sorted.

Discussion

Search cursors are able to be iterated with a **for** loop or in a **while** loop using the cursor's **next** method to return the next row. When using the **next** method on a cursor to retrieve all rows in a table containing N rows, the script must make N calls to **next**. A call to **next** after the last row in the result set has been retrieved returns **None**, which is a Python data type that acts here as a placeholder.

Using **SearchCursor** with a **for** loop.

```
import arcpy

fc = "c:/data/base.gdb/roads"
field = "StreetName"
cursor = arcpy.SearchCursor(fc)
for row in cursor:
    print(row.getValue(field))
```

Using **SearchCursor** with a **while** loop.

```
import arcpy

fc = "c:/data/base.gdb/roads"
field = "StreetName"
cursor = arcpy.SearchCursor(fc)
row = cursor.next()
while row:
    print(row.getValue(field))
    row = cursor.next()
```

Syntax

SearchCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})

Parameter	Explanation	Data Type
dataset	The feature class, shapefile, or table containing the rows to be searched.	String
where_clause	An optional expression that limits the rows returned in the cursor. For more information on WHERE clauses and SQL statements, see About building an SQL expression.	
spatial_reference	When specified, features will be projected on the fly using the **spatial_reference** provided.	SpatialReference
fields	The fields to be included in the cursor. By default, all fields are included.	String
sort_fields	Fields used to sort the rows in the cursor. Ascending and descending order for each field is denoted by A and D.	String

Return Value

Data Type	Explanation
Cursor	A **Cursor** object that can hand out row objects.

Code Sample

SearchCursor example

List field contents for Counties.shp. Cursor sorted by State Name and Population.

```
import arcpy

# Open a searchcursor
#  Input: C:/Data/Counties.shp
#  Fields: NAME; STATE_NAME; POP2000
#  Sort fields: STATE_NAME A; POP2000 D
rows = arcpy.SearchCursor("c:/data/counties.shp",
                          fields="NAME; STATE NAME; POP2000",
                          sort_fields="STATE_NAME A; POP2000 D")

# Iterate through the rows in the cursor and print out the
# state name, county and population of each.
for row in rows:
    print("State: {0}, County: {1}, Population: {2}".format(
        row.getValue("STATE_NAME"),
        row.getValue("NAME"),
        row.getValue("POP2000")))
```

Related Topics

Accessing data using cursors
UpdateCursor
InsertCursor

SetLogHistory (arcpy)

Top

Summary

For script tools and stand-alone scripts (scripts run outside of an ArcGIS application), you can enable or disable history logging using the **SetLogHistory** function.

The history log file is an Extensible Markup Language (XML) file that contains information about each geoprocessing operation. The information contained in the log file is essentially the same as that found in the Results window.

Syntax

SetLogHistory (log_history)

Parameter	Explanation	Data Type
log_history	True, to enable geoprocessing logging history and False, to disable.	Boolean

Code Sample

SetLogHistory example

Turn off geoprocessing log history.

```
import arcpy
arcpy.SetLogHistory(False)
```

Related Topics

GetLogHistory

Viewing tool execution history

SetParameter (arcpy)

Top

Summary

Sets a specified parameter property by index using an object. This is used when passing objects from a script to a script tool. If you need to pass a text value to a script tool, use SetParameterAsText.

Syntax

SetParameter (index, value)

Parameter	Explanation	Data Type
index	The specified parameter's index position in the parameter list.	Integer
value	The object that will set the specified parameter's property.	Object

Code Sample

SetParameter example

Pass object to specified tool parameter.

```
import arcpy

# Get the input feature class name.
#
fc = arcpy.GetParameterAsText(0)

# Obtain the spatial reference object and return it to the tool.
SR = arcpy.Describe(fc).spatialReference
arcpy.SetParameter(1, SR)
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterAsText
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameterAsText
Setting script tool parameters

SetParameterAsText (arcpy)

Top

Summary

Sets a specified parameter property by index using a string value. This is used when passing values from a script to a script tool. If you need to pass an object, such as a spatial reference to a script tool, use SetParameter.

Syntax

SetParameterAsText (index, text)

Parameter	Explanation	Data Type
index	The specified parameter's index position in the parameter list.	Integer
text	The string value that will set the specified parameter's property.	String

Code Sample

SetParameterAsText example

Pass text string to specified tool parameter.

```
import arcpy

# Get the feature class from the tool.
fc = arcpy.GetParameterAsText(0)

# Determine the shape type of the feature class.
dscFC = arcpy.Describe(fc)

# Set tool output parameters based on shape type.
#
if dscFC.ShapeType.lower() == "polygon":
    arcpy.AddMessage("Feature Type is polygon")
    arcpy.SetParameterAsText(1, "true")  # Is polygon
    arcpy.SetParameterAsText(2, "false") # Is not line
    arcpy.SetParameterAsText(3, "false") # Is not point

elif dscFC.ShapeType.lower() == "polyline":
    arcpy.AddMessage("Feature Type is polyline")
    arcpy.SetParameterAsText(1, "false") # Is not polygon
    arcpy.SetParameterAsText(2, "true")  # Is line
    arcpy.SetParameterAsText(3, "false") # Is not point

elif dscFC.ShapeType.lower() == "point":
    arcpy.AddMessage("Feature Type is point")
    arcpy.SetParameterAsText(1, "false") # Is not polygon
    arcpy.SetParameterAsText(2, "false") # Is not line
    arcpy.SetParameterAsText(3, "true")  # Is point

else:
    arcpy.AddMessage("Unknown feature type")
    arcpy.SetParameterAsText(1, "false") # Is not polygon
    arcpy.SetParameterAsText(2, "false") # Is not line
    arcpy.SetParameterAsText(3, "false") # Is not point
```

Related Topics

CopyParameter
GetArgumentCount
GetParameter
GetParameterAsText
GetParameterCount
GetParameterInfo
GetParameterValue
SetParameter
Setting script tool parameters

SetProduct (arcpy)

Top

Summary

The **SetProduct** function defines the desktop license. **SetProduct** returns information on the license.

Legacy:

The product level should be set by importing the appropriate product module (**arcinfo**, **arceditor**, **arcview**, **arcserver**, **arcenginegeodb**, **or arcengine**) prior to importing arcpy. The **SetProduct** function is a legacy function and cannot set the product once arcpy has been imported. For scripts using the **arcgisscripting** module, the equivalent **SetProduct** method is still supported.

Discussion

Tip:

The setting of the product and extensions is only necessary within stand-alone scripts. If you are running tools from the Python window or using script tools, the product is already set from within the application, and the active extensions are based on the Extensions dialog box.

Syntax

SetProduct (product)

Parameter	Explanation	Data Type
product	Product code for the product being set. • arcview —ArcGIS for Desktop Basic product code • arceditor —ArcGIS for Desktop Standard product code • arcinfo —ArcGIS for Desktop Advanced product code • engine —Engine Runtime product code • enginegeodb —Engine Geodatabase Update product code • arcserver — Server product code	String

Return Value

Data Type	Explanation
String	The function returns the status on the product license. There are four possible values: • CheckedOut —License successfully set. • AlreadyInitialized —License has already been set. • NotLicensed —The license is not valid or available. • Failed —A system failure occurred during the set request.

Code Sample

SetProduct example

Sets an ArcGIS for Desktop Basic product license using the arcview module.

```
# Set the ArcGIS for Desktop Basic product by importing the arcview
module.
import arcview
import arcpy

arcpy.env.workspace = "c:/data/SanDiego.gdb"

arcpy.CreateRasterDataset_management(
    arcpy.env.workspace, "LandUse", "30", "8_BIT_UNSIGNED",
    "Freeways", 1)
```

Related Topics

CheckProduct
ProductInfo
Accessing licenses and extensions in Python

SetProgressor (arcpy)

Top

Summary

Establishes a progressor object which allows progress information to be passed to a progress dialog box. The appearance of the progress dialog box can be controlled by choosing either the default progressor or the step progressor.

Syntax

SetProgressor (type, {message}, {min_range}, {max_range}, {step_value})

Parameter	Explanation	Data Type
type	The progressor type (default or step). • default —The progressor moves back and forth continuously. • step —The progressor shows the percentage complete. (The default value is default)	String
message	The progressor label. The default is no label.	String
min_range	Starting value for progressor. Default is 0. (The default value is 0)	Integer
max_range	Ending value for progressor. Default is 100. (The default value is 100)	Integer
step_value	The progressor step interval for updating the progress bar. (The default value is 1)	Integer

Code Sample

SetProgressor example

Set progressor object for displaying progress in the progress dialog box.

```
import os
import arcpy

# Allow overwriting of output
arcpy.env.overwriteOutput = True

# Set current workspace
arcpy.env.workspace = "c:/data"

# Get a list of shapefiles in folder
fcs = arcpy.ListFeatureClasses()

# Find the total count of shapefiles in list
fc_count = len(fcs)

# Set the progressor
arcpy.SetProgressor("step", "Copying shapefiles to geodatabase...",
                    0, fc_count, 1)

# Create a file gdb to contain new feature classes
arcpy.CreateFileGDB_management(arcpy.env.workspace, "fgdb.gdb")

# For each shapefile, copy to a file geodatabase
for shp in fcs:

    # Trim the '.shp' extension

    fc = os.path.splitext(shp)[0]


    # Update the progressor label for current shapefile
    arcpy.SetProgressorLabel("Loading {0}...".format(shp))

    # Copy the data
    arcpy.CopyFeatures_management(shp, os.path.join("fgdb.gdb", fc))

    # Update the progressor position
    arcpy.SetProgressorPosition()

arcpy.ResetProgressor()
```

Related Topics

ResetProgressor
SetProgressorLabel
SetProgressorPosition
Controlling the progress dialog box

SetProgressorLabel (arcpy)

Top

Summary

Updates the progressor dialog box label.

Syntax

SetProgressorLabel (label)

Parameter	Explanation	Data Type
label	The label to be used on the progressor dialog box.	String

Code Sample

SetProgressorLabel example

Updates the progressor dialog box label.

```
import os
import arcpy

# Allow overwriting of output
arcpy.env.overwriteOutput = True

# Set current workspace
arcpy.env.workspace = "c:/data"

# Get a list of shapefiles in folder
fcs = arcpy.ListFeatureClasses()

# Find the total count of shapefiles in list
fc_count = len(fcs)

# Set the progressor
arcpy.SetProgressor("step", "Copying shapefiles to geodatabase...",
                    0, fc_count, 1)

# Create a file gdb to contain new feature classes
arcpy.CreateFileGDB_management(arcpy.env.workspace, "fgdb.gdb")

# For each shapefile, copy to a file geodatabase
for shp in fcs:
    # Trim the '.shp' extension
    fc = os.path.splitext(shp)[0]

    # Update the progressor label for current shapefile
    arcpy.SetProgressorLabel("Loading {0}...".format(shp))

    # Copy the data
    arcpy.CopyFeatures_management(shp, os.path.join("fgdb.gdb", fc))

    # Update the progressor position
    arcpy.SetProgressorPosition()

arcpy.ResetProgressor()
```

Related Topics

ResetProgressor
SetProgressor
SetProgressorPosition
Controlling the progress dialog box

SetProgressorPosition (arcpy)

Top

Summary

Updates the status bar in the progressor dialog box.

Syntax

SetProgressorPosition ({position})

Parameter	Explanation	Data Type
position	Sets the position of the status bar in the progressor dialog box.	Integer

Code Sample

SetProgressorPosition example

Updates the status bar position in the progressor dialog box.

```
import os
import arcpy

# Allow overwriting of output
arcpy.env.overwriteOutput = True

# Set current workspace
arcpy.env.workspace = "c:/data"

# Get a list of shapefiles in folder
fcs = arcpy.ListFeatureClasses()

# Find the total count of shapefiles in list
fc_count = len(fcs)

# Set the progressor
arcpy.SetProgressor("step", "Copying shapefiles to geodatabase...",
                    0, fc_count, 1)

# Create a file gdb to contain new feature classes
arcpy.CreateFileGDB_management(arcpy.env.workspace, "fgdb.gdb")

# For each shapefile, copy to a file geodatabase
for shp in fcs:
    # Trim the '.shp' extension
    fc = os.path.splitext(shp)[0]

    # Update the progressor label for current shapefile
    arcpy.SetProgressorLabel("Loading {0}...".format(shp))

    # Copy the data
    arcpy.CopyFeatures_management(shp, os.path.join("fgdb.gdb", fc))

    # Update the progressor position
    arcpy.SetProgressorPosition()

arcpy.ResetProgressor()
```

Related Topics

ResetProgressor
SetProgressor
SetProgressorLabel
Controlling the progress dialog box

SetSeverityLevel (arcpy)

Top

Summary

Used to control how geoprocessing tools throw exceptions.

Discussion

If **SetSeverityLevel** is not used, the default behavior is equivalent to setting the severity_level to 2; that is, tools will only throw an exception when the tool has an error.

Syntax

SetSeverityLevel (severity_level)

Parameter	Explanation	Data Type
severity_level	The severity level • 0 —A tool will not throw an exception, even if the tool produces an error or warning. • 1 —If a tool produces a warning or an error, it will throw an exception. • 2 —If a tool produces an error, it will throw an exception. This is the default.	Integer

Code Sample

SetSeverityLevel example

Use SetSeverityLevel to force tool to throw an exception when a tool warning is encountered.

```
import arcpy

fc1 = 'c:/resources/resources.gdb/boundary'
fc2 = 'c:/resources/resources.gdb/boundary2'

# Set the severity level to 1 (tool warnings will throw an exception)
arcpy.SetSeverityLevel(1)
print("Severity is set to : {0}".format(arcpy.GetSeverityLevel()))

try:
    # FeatureCompare returns warning messages when a miscompare is
    #  found.  This normally would not cause an exception, however,
by
    #  setting the severity level to 1, all tool warnings will also
    #  return an exception.
    arcpy.FeatureCompare_management(fc1, fc2, "OBJECTID")
except arcpy.ExecuteWarning:
    print(arcpy.GetMessages(1))
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

GetSeverityLevel

TestSchemaLock (arcpy)

Top

Summary

Tests if a schema lock can be acquired for a feature class, table, or feature dataset. Tools that alter schema will require a schema lock to be placed on the input data. The Add Field tool is an example of such a tool. If the tool requires a schema lock and is unable to aquire one at the time of execution, an appropriate error message is returned. Scripts that use such tools should test if a schema lock can be acquired on the input data. The TestSchemaLock function will not actually apply a schema lock on the input data, but will return a Boolean.

Syntax

TestSchemaLock (dataset)

Parameter	Explanation	Data Type
dataset	The input data to be tested if a schema lock can be applied.	String

Return Value

Data Type	Explanation
Boolean	Returns a Boolean indicating if a schema lock can be applied to the input dataset. The possible Boolean values are: • True —A schema lock can be applied to the dataset. • False —A schema lock cannot be obtained on the dataset.

Code Sample

TestSchemaLock example

Returns a Boolean True if exclusive lock can be applied to dataset.

```
import arcpy

data = arcpy.GetParameterAsText(0)

# Test if a schema lock can be applied, and if so, add a new field
#
if arcpy.TestSchemaLock(data):
    arcpy.AddField_management(data, "Flag", "LONG")
else:
    print("Unable to acquire the necessary schema lock to add the new
field")
```

Related Topics

Rules for working with schema locks

UpdateCursor (arcpy)

Top

Summary

The **UpdateCursor** function creates a cursor that lets you update or delete rows on the specified feature class, shapefile, or table. The cursor places a lock on the data that will remain until either the script completes or the update cursor object is deleted.

Discussion

Update cursors are able to be iterated with a **for** loop or in a **while** loop using the cursor's **next** method to return the next row. When using the **next** method on a cursor to retrieve all rows in a table containing N rows, the script must make N calls to **next**. A call to **next** after the last row in the result set has been retrieved returns **None**, which is a Python data type that acts here as a placeholder.

Using **UpdateCursor** with a **for** loop.

```
import arcpy

fc = "c:/data/base.gdb/roads"
field1 = "field1"
field2 = "field2"

cursor = arcpy.UpdateCursor(fc)
for row in cursor:
    # field2 will be equal to field1 multiplied by 3.0
    row.setValue(field2, row.getValue(field1) * 3.0)
    cursor.updateRow(row)
```

Using **UpdateCursor** with a **while** loop.

```
import arcpy

fc = "c:/data/base.gdb/roads"
field1 = "field1"
field2 = "field2"

cursor = arcpy.UpdateCursor(fc)
row = cursor.next()
while row:
    # field2 will be equal to field1 multiplied by 3.0
    row.setValue(field2, row.getValue(field1) * 3.0)
    cursor.updateRow(row)
    row = cursor.next()
```

Syntax

UpdateCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})

Parameter	Explanation	Data Type
dataset	The feature class, shapefile, or table containing the rows to be updated or deleted.	String
where_clause	An optional expression that limits the rows returned in the cursor. For more information on WHERE clauses and SQL statements, seeAbout building an SQL expression.	String
spatial_reference	Coordinates are specified in the **spatial_reference** provided and converted on the fly to the coordinate system of the dataset.	SpatialReference
fields [fields,...]	The fields to be included in the cursor. By default, all fields are included.	String
sort_fields	Fields used to sort the rows in the cursor. Ascending and descending order for each field is denoted by A and D.	String

Return Value

Data Type	Explanation
Cursor	A **Cursor** object that can hand out row objects.

Code Sample

UpdateCursor example

Update field values in feature class, based on another field's value.

```
import arcpy

# Create update cursor for feature class
rows = arcpy.UpdateCursor("c:/data/base.gdb/roads")

# Update the field used in buffer so the distance is based on the
# road type. Road type is either 1, 2, 3 or 4. Distance is in meters.
for row in rows:
    # Fields from the table can be dynamically accessed from the
    # row object. Here fields named BUFFER_DISTANCE and ROAD_TYPE
    # are used
    row.setValue("BUFFER_DISTANCE", row.getValue("ROAD_TYPE") * 100)
    rows.updateRow(row)

# Delete cursor and row objects to remove locks on the data
del row
del rows
```

Related Topics

Accessing data using cursors
SearchCursor
InsertCursor

Usage (arcpy)

Top

Summary

Returns the syntax for the specified tool or function.

Discussion

Complete help information for a tool, including complete parameter descriptions, can be extracted by using the __doc__ property.

```
import arcpy
print(arcpy.Buffer_analysis.__doc__)
```

Syntax

Usage (tool_name)

Parameter	Explanation	Data Type
tool_name	The tool name to display the syntax.	String

Return Value

Data Type	Explanation
String	Returns a string containing the specified tool's syntax.

Code Sample

Usage example

Print specified tool's syntax.

```
import arcpy

print(arcpy.Usage("Buffer_analysis"))
print(arcpy.Usage("MakeFeatureLayer_management"))
```

Related Topics

Using tools in Python

ValidateDataStoreItem (arcpy)

Top

Summary

Validates whether a folder or database has been successfully registered with an ArcGIS Server site.

Discussion

See About registering your data with the server to learn more about when and why you should register your data with ArcGIS Server.

Syntax

ValidateDataStoreItem (connection_file, datastore_type, connection_name)

Parameter	Explanation	Data Type
connection_file	An ArcGIS Server connection file (.ags) for the server whose registered database or folder is being validated. If you've made a connection in ArcCatalog, you can use the connection file found in your user profile directory. Alternatively, you can create a connection file from scratch using the function CreateGISServerConnectionFile.	String
datastore_type	The type of data being validated. • DATABASE —The data resides in an enterprise database. • FOLDER —The data is file-based.	String
connection_name	The name by which the folder or database being validated is registered with the ArcGIS Server site.	String

Return Value

Data Type	Explanation
String	

Code Sample

ValidateDataStoreItem example

Prints the validity of all folders and databases registered with an ArcGIS Server site.

```
import arcpy

conn = "GIS Servers/MyConnection.ags"
for store_type in ["FOLDER", "DATABASE"]:
    print("Validating data store items of type
{}".format(store_type))
    for i in arcpy.ListDataStoreItems(conn, store_type):
        validity = arcpy.ValidateDataStoreItem(conn, store_type,
i[0])
        print("The data item '{}' is {}".format(i[0], validity))
```

Related Topics

AddDataStoreItem
ListDataStoreItems
RemoveDataStoreItem

ValidateFieldName (arcpy)

Top

Summary

Takes a string (field name) and a workspace path and returns a valid field name based on name restrictions in the output geodatabase. All invalid characters in the input string will be replaced with an underscore (_). The field name restrictions depend on the specific database used (Structured Query Language [SQL] or Oracle).

Syntax

ValidateFieldName (name, {workspace})

Parameter	Explanation	Data Type
name	The field name to be validated. If the optional workspace is not specified, the field name is validated against the current workspace.	String
workspace	An optional specified workspace to validate the field name against. The workspace can be a file system or a personal, file, or ArcSDE geodatabase. If the workspace is not specified, the field name is validated using the current workspace environment. If the workspace environment has not been set, the field name is validated based on a folder workspace.	String

Return Value

Data Type	Explanation
String	Returns a string containing the valid field name, based on either the current or specified workspace.

Code Sample

ValidateFieldName example

Returns a valid field name based on the workspace.

```
import arcpy
import os


class ShapeError(Exception):
    pass

try:
    # Get the input feature class and make sure it contains polygons.
    #
    in_fc = arcpy.GetParameterAsText(0)
    if arcpy.Describe(input).shapeType != "Polygon":
        # Raise a custom exception
        raise ShapeError("Input does not contain polygons")

    # Get the new field name and validate it.
    #
    field_name = arcpy.GetParameterAsText(1)

    field_name = arcpy.ValidateFieldName(field_name,
os.path.dirname(in_fc))

    # Add the new field and calculate the value.
    #
    arcpy.AddField_management(in_fc, field_name, "DOUBLE")
    arcpy.CalculateField_management(in_fc,
                                    field_name,
                                    "!shape.area! / !shape.length!",
                                    "PYTHON_9.3")
except ShapeError as err:
    print(err[0])
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
```

Related Topics

ValidateTableName
ParseFieldName
Validating table and field names in Python

ValidateTableName (arcpy)

Top

Summary

Takes a table name and a workspace path and returns a valid table name for the workspace. An underscore "_" will replace any invalid character found in the table name and will honor the name restrictions for the workspace. The table name restrictions depend on the specific RDBMS used.

Syntax

ValidateTableName (name, {workspace})

Parameter	Explanation	Data Type
name	The table name to be validated.	String
workspace	The optional workspace against which to validate the table name. If the workspace is not specified, the table name is validated using the current workspace environment. If the workspace environment has not been set, the table name is validated based on a folder workspace.	String

Return Value

Data Type	Explanation
String	The valid table name for the workspace, based on name restrictions of the workspace.

Code Sample

ValidateTableName example

Returns a valid table name for the workspace.

```
import os
import arcpy

# Get the input and output workspaces
#
arcpy.env.workspace = arcpy.GetParameterAsText(0)
out_workspace = arcpy.GetParameterAsText(1)

# Get a list of input feature classes to be copied and copy
#  to new output location
#
for fc in arcpy.ListFeatureClasses():
    out_fc = arcpy.ValidateTableName(fc, out_workspace)
    arcpy.CopyFeatures_management(
        fc, os.path.join(out_workspace, out_fc))
```

Related Topics

ParseTableName
ValidateFieldName
Validating table and field names in Python

Describe object properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for all Describe objects.

Properties

Property	Explanation	Data Type
baseName (Read Only)	The file base name	String
catalogPath (Read Only)	The path of the data	String
children (Read Only)	A list of sub elements	Describe
childrenExpanded (Read Only)	Indicates whether the children have been expanded	Boolean
dataElementType (Read Only)	The element type of the element	String
dataType (Read Only)	The type of the element	String
extension (Read Only)	The file extension	String
file (Read Only)	The file name	String
fullPropsRetrieved (Read Only)	Indicates whether full properties have been retrieved	Boolean
metadataRetrieved (Read Only)	Indicates whether the metadata has been retrieved	Boolean
name (Read Only)	The user-assigned name for the element	String
path (Read Only)	The file path	String

Code Sample

Describe object properties example (stand-alone script)

Display some Describe object properties for a file geodatabase.

```
import arcpy

# Create a Describe object
#
desc = arcpy.Describe("C:/Data/chesapeake.gdb")

# Print some Describe Object properties
#
if hasattr(desc, "name"):
    print "Name:         " + desc.name
if hasattr(desc, "dataType"):
    print "DataType:     " + desc.dataType
if hasattr(desc, "catalogPath"):
    print "CatalogPath: " + desc.catalogPath

# Examine children and print their name and dataType
#
print "Children:"
for child in desc.children:
    print "\t%s = %s" % (child.name, child.dataType)
```

ArcInfo Workstation Item properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for ArcInfo WorkstationINFO Table Items. ArcInfo Workstation Items are accessed from the **itemSet** property of ArcInfo Workstation INFO Table Properties.
An ArcInfo Workstation Item returns a **dataType** of "ArcInfoItem".

Properties

Property	Explanation	Data Type
alternateName (Read Only)	The alternate name is another name you can use to refer to the item. It sometimes contains abbreviated names for items that otherwise have long descriptive names. Long item names often help for documentation purposes. Shorter names may be convenient for ad hoc usage.	String
isIndexed (Read Only)	True if the item is indexed. Indexed items speed up selection operations on large INFO files.	Boolean
isPseudo (Read Only)	True if the item is a pseudo item.	Boolean
isRedefined (Read Only)	True if it is a redefined item. Redefined items can be subsets of regular items or can span multiple regular items.	Boolean
itemType (Read Only)	The data type of the item. One of Binary, Character, Date, Floating, Integer, Number, and OID.	String
numberDecimals (Read Only)	The number of digits to the right of the decimal place. This is only for item types that hold decimal numbers.	Integer
outputWidth (Read Only)	The number of spaces used to display the item's values.	Integer
startPosition (Read Only)	The starting position of a redefined item.	Integer
width (Read Only)	The number of spaces (or bytes) used to store the item's values.	Integer

Code Sample

ArcInfo Workstation Item properties example (stand-alone script)

The following stand-alone script displays properties from all the ArcInfo Workstation Items in an ArcInfo Workstation Table.

```
import arcpy

# Create a list of Describe objects from the ArcInfo Table.
#
descList = arcpy.Describe("C:/data/crimefreq").itemSet

# Print properties about each item in the itemSet
#
for item in descList:
    print item.name
    print "%-22s %s" % ("  Alternate name:", item.alternateName)
    print "%-22s %s" % ("  Is indexed:", item.isIndexed)
    print "%-22s %s" % ("  Is pseudo:", item.isPseudo)
    print "%-22s %s" % ("  Is redefined:", item.isRedefined)
    print "%-22s %s" % ("  Item type:", item.itemType)
    print "%-22s %s" % ("  Number of decimals:", item.numberDecimals)
    print "%-22s %s" % ("  Output width:", item.outputWidth)
    print "%-22s %s" % ("  Start position:", item.startPosition)
    print "%-22s %s" % ("  Width:", item.width)
```

ArcInfo Workstation Table properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for ArcInfo Workstation INFO Tables. Table Properties and Dataset Properties are also supported.
An INFO Table returns a **`dataType`** of `"ArcInfoTable"`.

Properties

Property	Explanation	Data Type
itemSet (Read Only)	A Python list of items in the table. Each entry in the list is an ArcInfo Workstation Item Properties describe object, representing one item in the table.	Object

Code Sample

ArcInfo Workstation Table properties example (stand-alone script)

The following stand-alone script displays a Table property from an ArcInfo Workstation Table. It then gets a list of ArcInfo Workstation items and prints the name of each item.

```
import arcpy

# Create a Describe object from the ArcInfo Table.
#
desc = arcpy.Describe("C:/data/crimefreq")

# Print a Table property from the ArcInfo Table.
#
#print "HasOID:       " + desc.hasOID
print "%-11s %s" % ("HasOID:", desc.hasOID)


# Get the itemSet from the ArcInfo Table and
# print the name and item type of each item.
#
iSet = desc.itemSet
for item in iSet:
    print "%-12s %s" % ("\nName:", item.name)
    print "%-11s %s" % ("Item type:", item.itemType)
```

CAD Drawing Dataset properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for CAD Drawing Datasets. Dataset Properties are also supported.

A CAD Drawing Dataset returns a **`dataType`** of `"CadDrawingDataset"`.

Properties

Property	Explanation	Data Type
is2D (Read Only)	Indicates whether a CAD dataset is 2D	Boolean
is3D (Read Only)	Indicates whether a CAD dataset is 3D	Boolean
isAutoCAD (Read Only)	Indicates whether a CAD dataset is an AutoCAD file	Boolean
isDGN (Read Only)	Indicates whether a CAD dataset is a MicroStation file	Boolean

Code Sample

CAD Drawing Dataset properties example (stand-alone script)

The following stand-alone script displays properties for a CAD Drawing Dataset.

```
import arcpy

# Create a describe object
#
desc = arcpy.Describe("C:/data/arch.dgn")

# Print Cad Drawing Dataset properties
#
print "%-12s %s" % ("is2D:", desc.is2D)
print "%-12s %s" % ("is3D:", desc.is3D)
print "%-12s %s" % ("isAutoCAD:", desc.isAutoCAD)
print "%-12s %s" % ("isDGN:", desc.isDGN)
```

CAD FeatureClass properties (arcpy)

Top

Summary

The **`Describe`** function returns FeatureClass Properties, Table Properties and Dataset Properties for CAD Feature Classes.
A CAD Feature Class returns a **`dataType`** of `"FeatureClass"`.

Cadastral Fabric properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for Cadastral Fabrics. Dataset Properties are also supported.

Properties

Property	Explanation	Data Type
bufferDistanceForAdjustment (Read Only)	The distance used to generate a buffer around the job parcels. This buffer defines the adjustment area.	Double
compiledAccuracyCategory (Read Only)	The default accuracy category for compiled parcels in this cadastral fabric. Category values range from 1 to 7, inclusively.	Integer
defaultAccuracyCategory (Read Only)	The default accuracy category for the whole cadastral fabric. Category values range from 1 to 7, inclusively.	Integer
maximumShiftThreshold (Read Only)	Coordinate changes will be written if the shift is greater than this tolerance value.	Double
multiGenerationEditing (Read Only)	True if Cadastral Fabrics greater than one level below default can be edited.	Boolean
multiLevelReconcile (Read Only)	True if reconciling and posting with an ancestor more than one generation above the working version is allowed.	Boolean
pinAdjustmentBoundary (Read Only)	True if points on the adjustment area boundary should be pinned.	Boolean
pinAdjustmentPointsWithinBoundary (Read Only)	True if nonadjusted points within the adjustment area should be pinned.	Boolean
surrogateVersion (Read Only)	The name of the surrogate version if applicable. Indicates if the cadastral fabric is a surrogate version. If set, the cadastral fabrics can be edited in the surrogate version and its immediate children. If not set, fabric editing can only be performed in the default version and its immediate children. The surrogate default version functionality is only available to versions newer than version 1.	String
type (Read Only)	The cadastral fabric type. • 0 —Map type, which allows flexible point editing. • 1 —Survey type, which prevents flexible point editing.	Integer
version (Read Only)	The schema version of the cadastral fabric system tables. **Schema version** / **ArcGIS version** 1 / 9.3, 9.3.1 2 / 10.0 3 / 10.1	Integer
writeAdjustmentVectors (Read Only)	True if adjustment vectors should be written.	Boolean

Coverage FeatureClass properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Coverage Feature Classes. FeatureClass Properties, Table Properties and Dataset Properties are also supported.
A Coverage Feature Class returns a **dataType** of "CoverageFeatureClass".

Properties

Property	Explanation	Data Type
featureClassType (Read Only)	The feature class types. • Point • Arc • Polygon • Node • Tic • Annotation • Section • Route • Link • Region • Label • File	String
hasFAT (Read Only)	True if the coverage feature class has a Feature Attribute Table (FAT) and False if it does not.	Boolean
topology (Read Only)	Indicates the state of the coverage feature class topology. • NotApplicable —The topology is not supported by this feature class. • Preliminary —Topology is preliminary. • Exists —Topology exists. • Unknown —Topology status is unknown.	String

Code Sample

Coverage FeatureClass properties example (stand-alone script)

The following stand-alone script displays properties for a Coverage FeatureClass.

```
import arcpy

# Create describe object from a coverage feature class
#
desc = arcpy.Describe("C:/data/tongass1/polygon")

# Print coverage feature class properties
#
print "%-17s %s" % ("featureClassType:", desc.featureClassType)
print "%-17s %s" % ("hasFAT:", desc.hasFAT)
print "%-17s %s" % ("topology:", desc.topology)
```

Coverage properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Coverages. Dataset Properties are also supported.
A Coverage returns a **dataType** of "Coverage".

Properties

Property	Explanation	Data Type
tolerances (Read Only)	Tolerances is a property set that includes: • fuzzy • dangle • ticMatch • edit • nodeSnap • weed • grain • snap	Object

Code Sample

Coverage properties example (stand-alone script)

The following stand-alone script displays tolerance properties for a coverage.

```
import arcpy

# Create a describe object from a coverage
desc = arcpy.Describe("C:/data/tongass1")

# Get the tolerances property set from the describe object
tolProps = desc.tolerances

# Print all eight tolerance properties
print "Tolerances"
print "=========="
print "%-10s %s" % ("fuzzy:", tolProps.fuzzy)
print "%-10s %s" % ("dangle:", tolProps.dangle)
print "%-10s %s" % ("ticMatch:", tolProps.ticMatch)
print "%-10s %s" % ("edit:", tolProps.edit)
print "%-10s %s" % ("nodeSnap:", tolProps.nodeSnap)
print "%-10s %s" % ("weed:", tolProps.weed)
print "%-10s %s" % ("grain:", tolProps.grain)
print "%-10s %s" % ("snap:", tolProps.snap)
```

Dataset properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Datasets. Dataset properties are available in many types of Describe objects.

Properties

Property	Explanation	Data Type
canVersion (Read Only)	Indicates whether the dataset can be versioned	Boolean
datasetType (Read Only)	Returns the type of dataset being described • Any • Container • Geo • FeatureDataset • FeatureClass • PlanarGraph • GeometricNetwork • Topology • Text • Table • RelationshipClass • RasterDataset • RasterBand • TIN • CadDrawing • RasterCatalog • Toolbox • Tool • NetworkDataset • Terrain • RepresentationClass • CadastralFabric • SchematicDataset • Locator	String
DSID (Read Only)	The ID of the dataset	Integer
extent (Read Only)	The Extent object	Extent
isVersioned (Read Only)	Indicates whether the dataset is versioned	Boolean
MExtent (Read Only)	A space-delimited string (MMin MMax)	String
spatialReference (Read Only)	Returns the SpatialReference object for the dataset	SpatialReference
ZExtent (Read Only)	A space-delimited string (ZMin ZMax)	String

Code Sample

Dataset properties example (stand-alone script)

The following stand-alone script displays some dataset properties for a shapefile.

```
import arcpy

# Create a Describe object from the shapefile
#
desc = arcpy.Describe("C:/temp/xy.shp")

# Print dataset properties
#
print("Dataset Type: {0}".format(desc.datasetType))
print("Extent:\n  XMin: {0}, XMax: {1}, YMin: {2}, YMax: {3}".format(
    desc.extent.XMin, desc.extent.XMax, desc.extent.YMin,
desc.extent.YMax))
print("MExtent: {0}".format(desc.MExtent))
print("ZExtent: {0}".format(desc.ZExtent))

print("Spatial reference name:
{0}:".format(desc.spatialReference.name))
```

dBase Table properties (arcpy)

Top

Summary

The **`Describe`** function returns Table Properties and Dataset Properties for dBase Tables.

A dBase Table returns a **`dataType`** of `"DbaseTable"`.

Editor Tracking properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for datasets that have Editor Tracking enabled.
Editor Tracking can be enabled for a FeatureClass, Table, Mosaic Dataset, or Raster Catalog. The **dataType** returned is the **dataType** of the Feature Class, Table, Mosaic Dataset, or Raster Catalog.

Properties

Property	Explanation	Data Type
editorTrackingEnabled (Read Only)	True if editor tracking is enabled for the dataset.	Boolean
creatorFieldName (Read Only)	The name of the field that contains the user name of the person who created a feature, row, or raster.	String
createdAtFieldName (Read Only)	The name of the field that contains the date and time a feature, row, or raster was created.	String
editorFieldName (Read Only)	The name of the field that contains the user name of the person who most recently edited a feature, row, or raster.	String
editedAtFieldName (Read Only)	The name of the field that contains the date and time a feature, row, or raster was most recently edited.	String
isTimeInUTC (Read Only)	True if times stored in the CreatedAt field and EditedAt field are stored in UTC (Coordinated Universal Time). False if they are stored in database time.	Boolean

Code Sample

Editor Tracking Dataset properties example (stand-alone script)

The following stand-alone script displays how many features in a feature class were most recently edited by each user.

```
import arcpy

# Create a Describe object from the feature class
#
gdb_fc = "C:/data/ParcelBase.gdb/parcels_enabled"
desc = arcpy.Describe(gdb_fc)

# If the feature class has editor tracking enabled, then
#   list how many features were last edited by each user.
#
if desc.editorTrackingEnabled:
    #
    # Get the editorFieldName from the describe object
    whoField = desc.editorFieldName
    #
    # Use a cursor to search through all the features
    userDictionary = {}
    cur = arcpy.da.SearchCursor(gdb_fc, [whoField])
    for row in cur:
        featureEditedBy = row[0]

        if featureEditedBy in userDictionary:
            userDictionary[featureEditedBy] += 1
        else:
            userDictionary[featureEditedBy] = 1
    #
    # Print the results
    for user in userDictionary.keys():
        if user == None:
            print 'Last edited before editor tracking was enabled: '+ \
                  str(userDictionary[user])
        else:
            print "Last edited by " + user + ": " + str(userDictionary[user])
else:
    print 'Editor tracking not enabled for '+gdb_fc
```

FeatureClass properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Feature Classes. Table Properties and Dataset Properties are also supported. Editor Tracking Properties are supported if editor tracking has been enabled for this feature class.
A Feature Class returns a **dataType** of "FeatureClass".

Properties

Property	Explanation	Data Type
featureType (Read Only)	The feature type of the feature class. • Simple —Polygons, polylines, and points representing objects or places that have area such as water bodies; linear objects such as rivers; and localized positions, such as houses or sample sites. • SimpleJunction —Simple junction feature in a geometric network representing point objects, such as a fuse, service point, or telephone pole. • SimpleEdge —Simple edge feature in a geometric network representing polyline objects, such as primary or secondary overheads. • ComplexEdge —Complex edge feature in a geometric network representing polyline objects, such as primary overheads, which have midspan connectivity. Network resources flow through complex edge without interruption by midspan connectivity. • Annotation —Place or object names or identifiers, such as street names, hydrant ID numbers, land values, or elevation. • CoverageAnnotation —Place or object names or identifiers, such as street names, hydrant ID numbers, land values, or elevation. Not supported in geodatabases; only supported in coverage datasets. • Dimension —Measurements, such as distances, lengths, widths, and depths. • RasterCatalogItem —A raster dataset in a raster catalog that has information, such as footprints, names, metadata, and any other user-defined attributes.	String
hasM (Read Only)	Indicates if the geometry is m-value enabled.	Boolean
hasZ (Read Only)	Indicates if the geometry is z-value enabled.	Boolean
hasSpatialIndex (Read Only)	Indicates if the feature class has a spatial index.	Boolean
shapeFieldName (Read Only)	The name of the Shape field.	String
shapeType (Read Only)	The geometry shape type. • Polygon • Polyline • Point • MultiPoint • MultiPatch	String

Code Sample

FeatureClass properties example (stand-alone script)

The following stand-alone script displays some feature class properties.

```
import arcpy

# Create a Describe object from the feature class
#
desc = arcpy.Describe("C:/data/arch.dgn/Point")

# Print some feature class properties
#
print "Feature Type:  " + desc.featureType
print "Shape Type :   " + desc.shapeType
print "Spatial Index: " + str(desc.hasSpatialIndex)
```

File properties (arcpy)

Top

Summary

When used with the **`Describe`** function, a File returns a **`dataType`** of `"File"`.

Code Sample

File properties example (stand-alone script)

The following stand-alone script displays some Describe Object properties for a file.

```
import arcpy


# Create a Describe object
#
desc = arcpy.Describe("C:/data/Install.log")

# Print some Describe Object properties for the file
#
print "Data Type: " + desc.dataType
print "Path:      " + desc.path
print "Base Name: " + desc.baseName
print "Extension: " + desc.extension
```

Folder properties (arcpy)

Top

Summary

The **`Describe`** function returns Workspace Properties for Folders.
A Folder returns a **`dataType`** of `"Folder"`.

Code Sample

Folder properties example (stand-alone script)

The following stand-alone script displays some properties for a folder.

```
import arcpy


# Create a Describe object
#
desc = arcpy.Describe("C:/data")

# Print the dataType and a workspace property
#
print "Data Type:      " + desc.dataType
print "Workspace Type: " + desc.workspaceType
```

GDB FeatureClass properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Geodatabase Feature Classes. FeatureClass Properties, GDB Table Properties, Editor Tracking Dataset Properties, Table Properties, and Dataset Properties are also supported.
A Geodatabase Feature Class returns a **dataType** of "FeatureClass".

Properties

Property	Explanation	Data Type
areaFieldName (Read Only)	The name of the geometry area field.	String
lengthFieldName (Read Only)	The name of the geometry length field.	String
representations (Read Only)	A list of Describe objects for the representations associated with the feature class.	Describe

Code Sample

GDB FeatureClass properties example (stand-alone script)

The following stand-alone script displays the GDB FeatureClass properties.

```
import arcpy

# Create a Describe object from the GDB Feature Class
#
desc = arcpy.Describe("C:/data/chesapeake.gdb/chesapeake/bayshed_1")

# Print GDB FeatureClass properties
#
print "Area Field Name  : " + desc.areaFieldName
print "Length Field Name: " + desc.lengthFieldName
```

GDB Table properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Geodatabase Tables. Editor Tracking Dataset Properties, Table Properties, and Dataset Properties are also supported.
A Geodatabase Table returns a **dataType** of "Table".

Properties

Property	Explanation	Data Type
aliasName (Read Only)	The alias name for the table.	String
defaultSubtypeCode (Read Only)	The default subtype code.	String
extensionProperties (Read Only)	The properties for the class extension.	Object
globalIDFieldName (Read Only)	The name of the GlobalID field.	String
hasGlobalID (Read Only)	Indicates whether the table has a GlobalID field.	Boolean
modelName (Read Only)	The model name for the table.	String
rasterFieldName (Read Only)	The name of the raster field.	String
relationshipClassNames (Read Only)	The names of the Relationship Classes that this table participates in.	String
subtypeFieldName (Read Only)	The name of the subtype field.	String
versionedView (Read Only)	The name of a Versioned View for a versioned feature class.	String

Code Sample

GDB Table properties example (stand-alone script)

The following stand-alone script displays some properties of a GDB table.

```
import arcpy

# Create a Describe object from the GDB table.
#
desc = arcpy.Describe("C:/data/chesapeake.gdb/munich")

# Print GDB Table properties
#
print "%-22s %s" % ("AliasName:", desc.aliasName)
print "%-22s %s" % ("DefaultSubtypeCode:", desc.defaultSubtypeCode)
print "%-22s %s" % ("GlobalIDFieldName:", desc.globalIDFieldName)
print "%-22s %s" % ("ModelName:", desc.modelName)
print "%-22s %s" % ("RasterFieldName:", desc.rasterFieldName)
print "%-22s %s" % ("RelationshipClassNames:",
desc.relationshipClassNames)
```

Geometric Network properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Geometric Networks. Dataset Properties are also supported.
A Geometric Network returns a **dataType** of "GeometricNetwork".

Properties

Property	Explanation	Data Type
featureClassNames (Read Only)	A Python List of the feature classes participating in the Geometric Network	String
networkType (Read Only)	The type of associate logical network • StreetNetwork • UtilityNetwork	String
orphanJunctionFeatureClassName (Read Only)	The name of the feature class that contains the Orphan Junction features	String

Code Sample

Geometric Network properties example (stand-alone script)

The following stand-alone script displays properties for a geometric network.

```
import arcpy

# Create a Describe object from the geometric network.
#
desc = arcpy.Describe("C:/data/wellingham.gdb/water/water_Net")

# Print some geometric network properties
#
print "NetworkType:                    " + desc.networkType
print "OrphanJunctionFeatureClassName: " + \
    desc.orphanJunctionFeatureClassName
print "Feature Class Names:"
for fcname in desc.featureClassNames:
    print "  " + fcname
```

LAS Dataset properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for LAS dataset files. File Properties and Dataset Properties are also supported. A LAS Dataset returns a **`dataType`** of `"LasDataset"`.

Properties

Property	Explanation	Data Type
constraintCount (Read Only)	The number of surface constraint features referenced by the LAS dataset.	Long
fileCount (Read Only)	The number of LAS files referenced by the LAS dataset.	Long
hasStatistics (Read Only)	Indicates if statistics had been calculated for the LAS files referenced by the LAS dataset.	Boolean
needsUpdateStatistics (Read Only)	Indicates if statistics are out-of-date or had not been calculated. Returns false if statistics are up-to-date.	Boolean
pointCount (Read Only)	The number of data points in the LAS files referenced by the LAS dataset.	Long
usesRelativePath (Read Only)	Indicates if the LAS dataset references its data elements using relative paths.	Boolean

Code Sample

LAS Dataset properties example (stand-alone script)

The following script demonstrates the application of LAS dataset properties.

```
import arcpy
desc = arcpy.Describe(r'E:\GIS_Data\lidar\test_bmore.lasd')

if desc.usesRelativePath:
    pathType = 'Relative'
else: pathType = 'Absolute'

# Determine state of statistics
if desc.needsUpdateStatistics:
    if desc.hasStatistics:
        statistics = 'Out-of-date'
    else:
        statistics = 'Missing'
else:
    statistics = 'Current'
print 'LAS Dataset Name: {0} \r'\
      'Point Count: {1} \r'\
      'Surface Constraint Count: {2} \r'\
      'Path Type: {3} \r'\
      'Statistics Status: {4}'.format(desc.basename, desc.pointCount,
                          desc.constraintCount, pathType,statistics)
```

Layer properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Layers. Dataset Properties are also supported, as well as the properties of the data type, the layer references. For example, a layer that references a feature class will have access to FeatureClass Properties, while a layer that references a raster dataset will have access to Raster Dataset Properties.

- If the data element being described is a layer in ArcMap or an in-memory layer, then the **dataType** returned gives information about the data source of layer being described. Some examples of this are `"MosaicLayer"`, `"FeatureLayer"`, and`"GroupLayer"`.
- If the data element being described is a .lyr file, a **dataType** of `"Layer"` is returned.
- You can get information about the layer contained by a .lyr file by inspecting the **Describe Object** returned by the **layer** property.

Properties

Property	Explanation	Data Type
dataElement (Read Only)	The Describe object of the data source to which the layer refers.	Describe
featureClass (Read Only)	The Describe object of the feature class associated with the feature layer.	Describe
FIDSet (Read Only)	A semicolon-delimited string of selected feature IDs (record numbers).	String
fieldInfo (Read Only)	The FieldInfo object (property set) of the layer.	FieldInfo
layer (Read Only)	The Describe object of the Layer within a .lyr file.	Describe
nameString (Read Only)	The name of the layer.	String
table (Read Only)	The Describe object of the Table within a FeatureLayer.	Describe
whereClause (Read Only)	The layer's definition query WHERE clause.	String

Code Sample

Layer properties example (stand-alone script)

The following stand-alone script displays some layer properties from an in-memory feature layer.

```
import arcpy

# Create an in memory feature layer from a feature class.
#
arcpy.MakeFeatureLayer_management(
        "C:/data/chesapeake.gdb/bayshed",
        "mainlines_layer")

# Create a Describe object from the feature layer.
desc = arcpy.Describe("mainlines_layer")

# Print some properties of the feature layer, and its featureclass.
#
print "Name String:         " + desc.nameString
print "Where Clause:        " + desc.whereClause
print "Feature class type: " + desc.featureClass.featureType
```

Layer properties example 2(stand-alone script)

The following stand-alone script displays some layer properties from a .lyr file.

```
import arcpy

# Create a Describe object from a .lyr file.
desc = arcpy.Describe("c:/data/water pipes.lyr")

# Print some properties of the feature layer
#
print "Name String:         " + desc.nameString
print "Where Clause:        " + desc.whereClause

# Find out if the layer represents a feature class
if desc.dataElement.dataType == "FeatureClass":
    print "Feature class:       " + desc.dataElement.catalogPath
    print "Feature class Type: " + desc.featureClass.featureType
else:
    print "Not a regular feature class"
```

Map Document properties (arcpy)

Top

Summary

When used with the **`Describe`** function, a Map Document returns a **`dataType`** of `"MapDocument"`.
The MapDocument class from the **`arcpy.mapping`** module can be used to get additional information about a map document.

Mosaic Dataset properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Mosaic Datasets. Raster Dataset Properties and Dataset Properties are also supported. Editor Tracking Properties are supported if editor tracking has been enabled for this mosaic dataset.
A Mosaic Dataset returns a **dataType** of "MosaicDataset".
Learn more about Mosaic Dataset properties

Properties

Property	Explanation	Data Type
allowedCompressionMethods (Read Only)	The methods of compression that could be used to transmit the mosaicked image from the server to the client. This property influences an image service generated from the mosaic dataset. • None —No compression • LZ77 —LZ77 compression • JPEG —JPEG compression • LERC —LERC compression	String
allowedFields (Read Only)	The attribute table fields visible to the client when the mosaic dataset is served as an image service.	String
allowedMensurationCapabilities (Read Only)	The mensuration tools can be used with an image service. • None — No mensuration tools can be used. • Basic — The Distance, Area, Point Location, and Centroid Location mensuration tool can be used. • Base-Top Height — The height of a structure is calculated by measuring from the base of the structure to the top of the structure. • Top-Top Shadow Height — The height of a structure is calculated by measuring from the top of the structure to the top of the structure's shadow on the ground. • Base-Top Shadow Height — The height of a structure is calculated by measuring from the base of the structure to the top of the structure's shadow on the ground. • 3D — Used to determine the heights of your 3D features such as buildings, when a DEM is available.	String
allowedMosaicMethods (Read Only)	The order of the rasters mosaicked together to render the mosaicked display. • None — Orders rasters based on the order (ObjectID) in the mosaic dataset attribute table. • Closest to Center — Enables rasters to be sorted based on a default order where rasters that have their centers closest to the view center are placed on top. • Closest to Nadir — Enables rasters to be sorted by the distance between the nadir position and view center. This is similar to the Closest to Center method but uses the nadir point to a raster, which may be different than the center, especially for oblique imagery. • Closest to Viewpoint — Orders rasters based on a user-defined location and nadir location for the rasters using the Viewpoint tool. • By Attribute — Enables raster ordering based on a defined metadata attribute and its difference from a base value. • North-West — Enables raster ordering in a view-independent way, where rasters with their centers to the northwest are displayed on top. • Seamline — Cuts the raster using the predefined seamline shape for each raster using optional feathering along the seams and orders images based on the SOrder field in the attribute table. • Lock Raster — Enables a user to lock the display of single or multiple rasters based on the ObjectID.	String
applyColorCorrection (Read Only)	If color correction information is available in the mosaic dataset, then a value of True means it will be applied.	Boolean
blendWidth (Read Only)	The distance used by the Blend mosaic operator.	Double
blendWidthUnits (Read Only)	The units in which the blend width is specified. • Pixels — The blend width is measured in pixels. • Mosaic Dataset Unit — The unit for the blend width will be in the same units as mosaic dataset.	String
cellSizeToleranceFactor (Read Only)	How mosaic dataset items with different pixel sizes are grouped together for some operations, such as mosaicking or seamline generation.	Double
childrenNames (Read Only)	A list of side table names that are components of the mosaic dataset.	String
clipToBoundary (Read Only)	**True** means the image extent is limited to the geometry of the boundary. **False** means it is limited to the extent of the boundary.	Boolean
clipToFootprint (Read Only)	Whether the extent of each raster is limited to its footprint.	Boolean
defaultCompressionMethod	The default compression method for the image.	String

(Read Only)	This value comes from the list of methods returned by the **allowedCompressionMethods** property.	
defaultMensurationCapability (Read Only)	The default mensuration tool that will be used with an image service. This value comes from the list returned by the**allowedMensurationCapabilities** property.	String
defaultMosaicMethod (Read Only)	The default mosaic method for the mosaicked image. This value comes from the list returned by the **allowedMosaicMethods** property.	String
defaultMosaicOperator (Read Only)	The default method for resolving overlapping cells. • FIRST —The output cell value of the overlapping areas will be the value from the first raster dataset mosaicked into that location. • LAST —The output cell value of the overlapping areas will be the value from the last raster dataset mosaicked into that location. This is the default. • BLEND —The output cell value of the overlapping areas will be a horizontally weighted calculation of the values of the cells in the overlapping area. • MEAN —The output cell value of the overlapping areas will be the average value of the overlapping cells. • MINIMUM —The output cell value of the overlapping areas will be the minimum value of the overlapping cells. • MAXIMUM —The output cell value of the overlapping areas will be the maximum value of the overlapping cells. • SUM —The output cell value of the overlapping areas will be the total sum of the overlapping cells.	String
defaultResamplingMethod (Read Only)	The default sampling method of the pixels. • NEAREST — The nearest neighbor resampling method will be used. • BILINEAR — The bilinear interpolation resampling method will be used. • CUBIC — The cubic convolution resampling method will be used. • MAJORITY — The majority resampling method will be used.	String
defaultSortingOrder (Read Only)	The default ordering of the images defined by the mosaic methods. • ASCENDING — Rasters will be sorted in ascending order. • DESCENDING — Rasters will be sorted in descending order.	String
endTimeField (Read Only)	The field that defines the end time.	String

footprintMayContainNoData (Read Only)	True if NoData is a valid pixel value.	Boolean
GCSTransforms (Read Only)	The transformations applied if the spatial references of the source rasters differ from the spatial reference of the mosaic dataset.	String
JPEGQuality (Read Only)	The percentage of image quality retained if JPEG compression is used on this dataset.	Long
LERCTolerance (Read Only)	The maximum error value applied per pixel if the mosaic dataset uses LERC compression.	Double
maxDownloadImageCount (Read Only)	The maximum number of rasters that a client can download from an image service.	Long
maxDownloadSizeLimit (Read Only)	The maximum size, in megabytes, of rasters that a client can download from an image service.	Long
maxRastersPerMosaic (Read Only)	The maximum number of rasters mosaicked. This property affects an image service generated from the mosaic dataset.	Long
maxRecordsReturned (Read Only)	Maximum number of records returned by the server when viewing the mosaic dataset as a published image service.	Long
maxRequestSizeX (Read Only)	The maximum number of columns each time a mosaicked image is generated.	Long
maxRequestSizeY (Read Only)	The maximum number of rows each time a mosaicked image is generated.	Long
minimumPixelContribution (Read Only)	The minimum number of pixels needed within the area of interest in order for a mosaic dataset item to be considered as part of that area.	Long
orderBaseValue (Read Only)	The images are sorted based on the difference between this value and the value specified in the orderField. This is applicable only for the **By Attribute** mosaic method.	Double
orderField (Read Only)	The metadata attribute used for raster ordering. This is applicable only for the **By Attribute** mosaic method.	String
rasterMetadataLevel (Read Only)	How much metadata will be transmitted from the server to the client. • None — No metadata will be transmitted. • Full — The basic raster dataset information and the function chain's details will be transmitted. • Basic — The raster dataset level of information will be transmitted, such as the columns and rows, cell size, and spatial reference information.	String
referenced (Read Only)	True if it is a referenced mosaic dataset. False if it is a regular mosaic dataset.	Boolean
startTimeField (Read Only)	The field that defines the start time.	String
timeValueFormat (Read Only)	The format in which the start time and end times are specified.	String
useTime (Read Only)	True if the mosaic dataset is time aware.	Boolean
viewpointSpacingX	The horizontal offset used to calculate where the	Double

(Read Only)	center of the area of interest (display view) is when you click an arrow button on the **Viewpoint**dialog box. These values are calculated in the units of the spatial reference system of the mosaic dataset. This is applicable only for the **Closest to Viewpoint** mosaic method.	
viewpointSpacingY (Read Only)	The vertical offset used to calculate where the center of the area of interest (display view) is when you click an arrow button on the **Viewpoint** dialog box. These values are calculated in the units of the spatial reference system of the mosaic dataset. This is applicable only for the **Closest to Viewpoint** mosaic method.	Double

Network Analyst Layer properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for Network Analyst Layers. A Network Analyst Layer returns a **`dataType`** of `"NALayer"`.
A Network Analyst Layer contains information about the inputs and parameters used in the analysis of a network problem using the ArcGIS Network Analyst extension.

Properties

Property	Explanation	Data Type
network (Read Only)	The Network Dataset object. This object can be used to get all the properties, such as **`catalogPath`**, of the underlying network dataset used by the analysis layer.	Object
nameString (Read Only)	The name of the Network Analyst layer.	String
solverName (Read Only)	The Network Analyst solver being referenced by the layer. Each layer can reference only one solver. This property returns the following keywords: • Route Solver • Closest Facility Solver • Service Area Solver • OD Cost Matrix Solver • Vehicle Routing Problem Solver • Location-Allocation Solver	String
impedance (Read Only)	The network cost attribute used as the impedance during the analysis.	String
accumulators (Read Only)	A semicolon-separated list of network cost attributes that are accumulated as part of the analysis.	String
restrictions (Read Only)	A semicolon-separated list of restriction attributes that are applied for the analysis.	String
ignoreInvalidLocations (Read Only)	A Boolean expression indicating the way in which the solver considers invalid network locations within the Network Analyst classes. A value of True indicates that the invalid locations are ignored by the solver. False indicates that the invalid locations are not ignored by the solver.	Boolean
uTurns (Read Only)	Indicates how the U-turns at junctions, which could occur during network traversal between stops, are being handled by the solver. This property returns the following keywords: • ALLOW_UTURNS • NO_UTURNS • ALLOW_DEAD_ENDS_ONLY • ALLOW_DEAD_ENDS_AND_INTERSECTIONS_ONLY	String
useHierarchy (Read Only)	Indicates if the Network Analyst Layer is using Hierarchy. This property returns the following keywords: • USE_HIERARCHY • NO_HIERARCHY	String
hierarchyAttribute (Read Only)	The name of the hierarchy attribute.	String
hierarchyLevelCount (Read Only)	The number of hierarchy ranges used to define the hierarchy attribute. The maximum value is 3.	Integer
maxValueForHierarchyX (Read Only)	A given hierarchy attribute can have values that define the hierarchy ranges. The maximum values for each range can be obtained from**`maxValueForHierarchyX`** property, where X indicates the hierarchy level. For example, the maximum value for the first hierarchy range (used to define the primary roads) can be obtained from the property **`maxValueForHierarchy1`**. Use the **`hierarchyLevelCount`** property to determine the possible number of **`maxValueForHierarchy`** properties for a given hierarchy attribute. For example, if the **`hierarchyLevelCount`** property returns 3, then the Describe object will support **`MaxValueForHierarchy1`** and**`MaxValueForHierarchy2`** properties.	Integer
locatorCount (Read Only)	The total number of classes that are used to search through for determining a network location.	Integer
locators (Read Only)	The Network Analyst Locator object. This object can be used to obtain name, snap type, and the search query information for the classes that are used for finding network locations.	Object
findClosest (Read Only)	Indicates how the network source features are searched when locating network analysis objects. This property returns the following keywords: • MATCH_TO_CLOSEST • PRIORITY	String
searchTolerance (Read Only)	A space-separated string indicating the search tolerance and the units used when finding the network locations.	String
excludeRestrictedElements (Read Only)	Indicates if the network analysis objects can be located only on traversable portions of network sources This property returns the following keywords: • INCLUDE • EXCLUDE	String
solverProperties (Read Only)	The Network Analyst Solver object. This object can be used to determine the solver-specific properties in a Network Analyst layer.	Object
children (Read Only)	Returns a Python List of describe objects that reference individual network analysis classes (such as stops and routes) as layers. This property cannot be used with the network analysis layer stored on disk as a layer file.	List
parameterCount (Read Only)	The total number of attribute parameters defined for all the network attributes of the underlying network dataset used by the Network Analyst layer.	Integer
parameters (Read Only)	The Network Attribute Parameter object. This object can be used to determine the attribute parameters used for the network analysis.	Object

Code Sample

NetworkAnalystLayer properties example

Display properties of a specified Network Analyst layer file.

```
# Name: NALayerProperties_ex01.py
# Description: Lists all the properties that can be derived by
describing a network analysis layer.
import arcpy

#Arguments: name of Network Analysis layer file to be described.
in_layer = "C:/Data/Route.lyr"

#Get the description object using Describe.
desc = arcpy.Describe(in_layer)

arcpy.AddMessage(" ")
arcpy.AddMessage("== Description of network analysis layer " +
in_layer + " ==")
arcpy.AddMessage(" ")

#Print general infomation.
justify = 35
nds = desc.network
solvername = desc.solverName
arcpy.AddMessage("---- General information:")
arcpy.AddMessage(" %*s: %s" % (justify, "Network" , nds.catalogPath))
arcpy.AddMessage(" %*s: %s" % (justify, "SolverName" ,
desc.solverName))
arcpy.AddMessage(" %*s: %s" % (justify, "Impedance" ,
desc.impedance))
arcpy.AddMessage(" %*s: %s" % (justify, "Accumulators" ,
desc.accumulators))
arcpy.AddMessage(" %*s: %s" % (justify, "Restrictions" ,
desc.restrictions))
arcpy.AddMessage(" %*s: %s" % (justify, "Ignore invalid locations?" ,
              str(desc.ignoreInvalidLocations)))
arcpy.AddMessage(" %*s: %s" % (justify, "UTurn policy" ,
desc.UTurns))
arcpy.AddMessage(" %*s: %s" % (justify, "Using hierarchy?" ,
desc.useHierarchy))
arcpy.AddMessage(" %*s: %s" % (justify, "Exclude Restricted
Elements?" ,
                               desc.excludeRestrictedElements))
arcpy.AddMessage(" ")

#A note about the dynamic properties (indicated by X in the help
system). In order to access the dynamic properties use Python's built
in getattr()function. In order to find out if a dynamic property is
supported by the describe object, use Python's built in hasattr()
function.

# Print attribute parameter information
arcpy.AddMessage(" ---- Attribute Parameter information ----")
count = desc.parameterCount
if count == 0:
    arcpy.AddMessage(" ---- No Attribute Parameters defined ----")
else:
    parameters = desc.parameters
    for i in range(0, count):
        attributeName = getattr(parameters,"attributeName" + str(i))

        parameterName = getattr(parameters, "parameterName" + str(i))

        parameterValue = getattr(parameters, "parameterValue" +
str(i))
        arcpy.AddMessage(" %*s: %s: %s" % (justify, attributeName,
                                           parameterName,
parameterValue))

# Print hierarchy information
if desc.useHierarchy.lower() == "use_hierarchy":
    arcpy.AddMessage(" ---- Hierarchy information ----")
    arcpy.AddMessage(" %*s: %s" % (justify, "Hierarchy Attribute
Name",
                                   desc.hierarchyAttribute))
    count = desc.hierarchyLevelCount
    arcpy.AddMessage(" %*s: %d" % (justify, "Hierarchy Level Count" ,
count))
    for i in range(0, count ):
        levelRange = ""
        if i == 0:
            levelUB = getattr(desc,"maxValueForHierarchy" + str(i+1))
            levelRange = "up to %s" % levelUB
        elif i == (count - 1):
            prevLevelUB = getattr(desc, "maxValueForHierarchy" +
str(i))
            levelRange = "%s and higher" % (prevLevelUB + 1)
        else:
            prevLevelUB =  getattr(desc, "maxValueForHierarchy" +
str(i))
            levelUB = getattr(desc,"maxValueForHierarchy" + str(i +
1))
            levelRange = "%s - %s" % ((prevLevelUB + 1), levelUB)

        arcpy.AddMessage(" %*s %d range: %s" % (justify, "level", (i
+ 1),
                                                levelRange))
    arcpy.AddMessage(" ")

# Print locator information.
arcpy.AddMessage("---- Locator information:")
count = desc.locatorCount
arcpy.AddMessage(" %*s: %d" % (justify, "Count" , count))
arcpy.AddMessage(" %*s: %s" % (justify, "Find Closest?" ,
desc.findClosest))
arcpy.AddMessage(" %*s: %s" % (justify, "Search Tolerance",
                               desc.searchTolerance))
arcpy.AddMessage(" %*s: %s" % (justify, "Exclude Restricted
Elements",
                               desc.excludeRestrictedElements))
locators = desc.locators
for i in range(0, count):
    sourceName = getattr(locators, "source" + str(i))
    sourceType = getattr(locators, "snapType" + str(i))
    searchQuery = getattr(locators, "searchQuery" + str(i))
    arcpy.AddMessage(" %*s: %s" %(justify, sourceName, sourceType))
    arcpy.AddMessage(" %*s: %s" %(justify, sourceName, searchQuery))
```

Network Dataset properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Network Datasets. Dataset Properties are also supported.
A Network Dataset returns a **dataType** of "NetworkDataset".
A network dataset is used to model transportation networks.

Properties

Property	Explanation	Data Type
networkType (Read Only)	The type of workspace containing the network dataset. This property returns the following keywords: • Geodatabase • Shapefile • SDC	String
supportsTurns (Read Only)	Indicates if the network dataset supports turns.	Boolean
isBuildable (Read Only)	Indicates if the network dataset can be built. SDC-based network datasets cannot be built as they are read-only.	Boolean
catalogPath (Read Only)	The path of the network dataset.	String
attributes (Read Only)	Returns a Python list of Network Attribute objects.	Object
edgeSources (Read Only)	Returns a Python list of Edge Source objects.	Object
junctionSources (Read Only)	Returns a Python list of Junction Source objects.	Object
turnSources (Read Only)	Returns a Python list of Turn Source objects.	Object
systemJunctionSource (Read Only)	Returns a System Junction Source object defined for the network dataset. This property is not available with SDC-based network datasets as they do not support system junction sources.	Object
supportsDirections (Read Only)	Indicates if the network dataset supports generating directions.	Boolean
directions (Read Only)	Returns a Network Directions object defined for the network dataset. This object can be used to get directions information at the network dataset level. The **directions** property is available only if the **supportsDirections** property returns true.	Object
sources (Read Only)	Returns a Python list of Network Source objects. This property returns all the sources for the network dataset. If you want to get a list of particular source type—for example, only the edge sources—use the **edgeSources** property.	Object
elevationModel (Read Only)	The network elevation model used to refine the connectivity of the network dataset. This property returns the following keywords: • None • Elevation Fields • Z Coordinate Values	String
timeZoneAttributeName (Read Only)	The name of the time zone attribute. If the network dataset does not support time zones, this property returns an empty string.	String
timeZoneTableName (Read Only)	The name of the time-zone table that stores the list of time zones used by the network dataset.	String
supportsHistoricalTrafficData (Read Only)	Indicates if the network dataset supports the use of historical traffic information.	Boolean
historicalTrafficData (Read Only)	Returns an Historical Traffic Data object defined for the network dataset. This object can be used to get historical traffic information such as the historical traffic tables used by the network dataset. This property is available only if the **supportsHistoricalTrafficData** property returns true.	Object
supportsLiveTrafficData (Read Only)	Indicates if the network dataset supports the use of live traffic information.	Boolean
liveTrafficData (Read Only)	Returns a Live Traffic Data object defined for the network dataset. This object can be used to get the information about live traffic properties such as traffic feed name used by the network dataset. This property is available only if the **supportsLiveTrafficData** property returns true.	Object

Code Sample

Network Dataset Properties example

Display some network dataset properties.

```
# Description: Print some of the network dataset properties.
import arcpy

# Set the workspace
arcpy.env.workspace = "C:/Data/Paris.gdb/Transportation"
# Create Describe object for the network dataset
desc = arcpy.Describe("ParisMultimodal ND")

# Print general network dataset properties
print "Network type:    " + desc.networkType
print "Supports turns? " + str(desc.supportsTurns)
print "Supports directions? " + str(desc.supportsDirections)
print "Is buildable?    " + str(desc.isBuildable)
print "Elevation model: " + desc.elevationModel
print "Supports historical traffic data: " +
str(desc.supportsHistoricalTrafficData)
print "Time zone attribute name: " + desc.timeZoneAttributeName
```

```
print "Time zone table name: " + desc.timeZoneTableName
```

Prj File properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Prj Files.
A Prj File returns a **dataType** of "PrjFile".

Properties

Property	Explanation	Data Type
spatialReference (Read Only)	The SpatialReference class instance of the .prj file	SpatialReference

Code Sample

Prj File properties example (stand-alone script)

The following stand-alone script displays some SpatialReference class properties for a prj file.

```
import arcpy

# Create a Describe Object from a prj file.
#
desc = arcpy.Describe("C:\data\mexico.prj")

# Print some properties of the SpatialReference class object.
#
SR = desc.spatialReference
print "Name:            " + SR.name
print "Type:            " + SR.type
print "isHighPrecision: " + str(SR.isHighPrecision)
print "scaleFactor:     " + str(SR.scaleFactor)
```

Raster Band properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Raster Bands. Table Properties and Dataset Properties are also supported.
A Raster Band returns a **dataType** of "RasterBand".

Properties

Property	Explanation	Data Type
height (Read Only)	The number of rows	Integer
isInteger (Read Only)	Indicates whether the raster band has integer type	Boolean
meanCellHeight (Read Only)	The cell size in y direction	Double
meanCellWidth (Read Only)	The cell size in x direction	Double
noDataValue (Read Only)	The NoData value of the raster band	String
pixelType (Read Only)	The pixel type • U1 —1 bit • U2 —2 bits • U4 —4 bits • U8 —Unsigned 8 bit integers • S8 —8 bit integers • U16 —Unsigned 16 bit integers • S16 —16 bit integers • U32 —Unsigned 32 bit integers • S32 —32 bit integers • F32 —Single precision floating point • F64 —Double precision floating point	String
primaryField (Read Only)	The index of the field	Integer
tableType (Read Only)	The class names of the table • Value —Values in the table are used for values only, not for indexing. • Index —Values in the table are used as indexes in the raster table. • Invalid —Values are invalid.	String
width (Read Only)	The number of columns	Integer

Code Sample

Raster Band properties example (stand-alone script)

The following stand-alone script displays some properties for a raster band.

```
import arcpy

# Create a Describe object from the raster band
#
desc = arcpy.Describe("C:/data/preston.img/Band_1")

# Print some raster band properties
#
print "Height: %d" % desc.height
print "Width:  %d" % desc.width
print "Integer Raster: %s" % desc.isInteger
```

Raster Catalog properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Raster Catalogs: GDB FeatureClass Properties, FeatureClass Properties, GDB Table Properties, Table Properties, and Dataset Properties are also supported. Editor Tracking Properties are supported if editor tracking has been enabled for this raster catalog.

A Raster Catalog returns a **dataType** of "RasterCatalog".

Properties

Property	Explanation	Data Type
rasterFieldName (Read Only)	The name of the raster column in the raster catalog.	String

Code Sample

Raster Catalog properties example (stand-alone script)

The following stand-alone script displays the rasterFieldName property for a raster catalog.

```
import arcpy

# Create a Describe object from the raster catalog
#
desc = arcpy.Describe("C:/data/simon.gdb/idaho")

# Print the RasterFieldName property
#
print "Raster field name: " + desc.rasterFieldName
```

Raster Dataset properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for Raster Datasets. Dataset Properties are also supported. Single-band raster datasets also support Raster Band Properties.
A Raster Dataset returns a **`dataType`** of `"RasterDataset"`.

Properties

Property	Explanation	Data Type
bandCount (Read Only)	The number of bands in the raster dataset.	Integer
compressionType (Read Only)	The compression type • LZ77 • JPEG • JPEG2000 • None	String
format (Read Only)	The raster format • Grid • ERDAS IMAGINE • TIFF	String
permanent (Read Only)	Indicates the permanent state of the raster: False if the raster is temporary and True if the raster is permanent.	Boolean
sensorType (Read Only)	The sensor type used to capture the image.	String

Code Sample

Raster Dataset properties example (stand-alone script)

The following stand-alone script displays some properties for a raster dataset.

```
import arcpy

# Create a Describe object from the raster dataset
#
desc = arcpy.Describe("C:/data/preston.img")

# Print some raster dataset properties
#
print "Band Count:       %d" % desc.bandCount
print "Compression Type: %s" % desc.compressionType
print "Raster Format:    %s" % desc.format
```

RecordSet/FeatureSet properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for instances of RecordSet and FeatureSet. Table Properties and Dataset Properties are also supported by both. In addition, FeatureSet supports FeatureClass Properties. Many of these properties will be empty unless the RecordSet or FeatureSet has been populated by calling its `load` method.

A RecordSet returns a **`dataType`** of `"RecordSet"`.

A FeatureSet returns a **`dataType`** of `"FeatureSet"`.

A REST client (such as a web application that uses a geoprocessing service) uses JSON streams to exchange data with a Service End point. You can use the `json` or `pjson` property for generating test input for a service or to examine the JSON representation of features and tables that are used by an ArcGIS REST Service. For an example of this, see Using a service in Python scripts.

Note:

For typical ArcPy use, passing a table or feature class as an argument to a service is faster than passing the JSON string representation of the same table or feature class.

Properties

Property	Explanation	Data Type
json (Read Only)	A JSON string representing the table or feature class that underlies the arcpy.RecordSet or arcpy.FeatureSet.	String
pjson (Read Only)	Pretty JSON. A JSON string formatted to be easily readable. This string is a little larger because it includes extra newline and whitespace characters.	String

Code Sample

FeatureSet properties example (stand-alone script)

The following stand-alone script loads a feature class into an arcpy.FeatureSet, then prints the pjson string.

```
import arcpy

# Describe a populated arcpy.FeatureSet
#
fSet = arcpy.FeatureSet()
fSet.load("C:\data\moad.gdb\Water Bodies")
desc = arcpy.Describe(fSet)

# print a JSON representation
print (desc.pjson)
```

RelationshipClass properties (arcpy)

Top

Summary

The **`Describe`** function returns the following properties for Relationship Classes. Also supported are GDB Table Properties, Table Properties, and Dataset Properties.
A Relationship Class returns a **`dataType`** of `"RelationshipClass"`.

Properties

Property	Explanation	Data Type
backwardPathLabel (Read Only)	The backward path label for the relationship class.	String
cardinality (Read Only)	The cardinality for the relationship class. • OnetoOne • OneToMany • ManyToMany	String
classKey (Read Only)	Class key used for the relationship class. • Undefined • ClassID • ClassCode	String
destinationClassNames (Read Only)	A list containing the names of the destination classes.	String
forwardPathLabel (Read Only)	The forward path label for the relationship class.	String
isAttachmentRelationship (Read Only)	Indicates if the relationship class represents a table attachment.	Boolean
isAttributed (Read Only)	Indicates if the relationships in this relationship class have attributes.	Boolean
isComposite (Read Only)	Indicates if the relationship class represents a composite relationship in which the origin object class represents the composite object.	Boolean
isReflexive (Read Only)	Indicates if the origin and destination sets intersect.	Boolean
keyType (Read Only)	Key type for the relationship class. • Single • Dual	String
notification (Read Only)	The notification direction for the relationship class. • None • Forward • Backward • Both	String
originClassNames (Read Only)	A list containing the names of the origin classes.	String

Code Sample

RelationshipClass properties example (stand-alone script)

The following stand-alone script displays properties for a relationship class.

```
import arcpy

# Create a Describe object
#
desc = arcpy.Describe("C:/data/moad.gdb/West/bapCompAttRel")

# Print relationship class properties
#
print "%-25s %s" % ("Backward Path Label:", desc.backwardPathLabel)
print "%-25s %s" % ("Cardinality:", desc.cardinality)
print "%-25s %s" % ("Class key:", desc.classKey)
print "%-25s %s" % ("Destination Class Names:",
desc.destinationClassNames)
print "%-25s %s" % ("Forward Path Label:", desc.forwardPathLabel)
print "%-25s %s" % ("Is Attributed:", desc.isAttributed)
print "%-25s %s" % ("Is Composite:", desc.isComposite)
print "%-25s %s" % ("Is Reflexive:", desc.isReflexive)
print "%-25s %s" % ("Key Type:", desc.keyType)
print "%-25s %s" % ("Notification Direction:", desc.notification)
print "%-25s %s" % ("Origin Class Names:", desc.originClassNames)
```

RepresentationClass properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Representation Classes. Dataset Properties are also supported.
Describe properties for Representation Classes can be obtained from either the GDBFeatureClass.representations property or the DescribeObject.children property of a GDB FeatureClass.
A Representation Class returns a **dataType** of "RepresentationClass".

Properties

Property	Explanation	Data Type
overrideFieldName (Read Only)	The name of the Override field.	String
requireShapeOverride (Read Only)	Indicates if a shape override is required for feature representations.	Boolean
ruleIDFieldName (Read Only)	The name of the RuleID field.	String

Code Sample

RepresentationClass properties example (stand-alone script)

The following stand-alone script displays properties for all the representation classes in a feature class.

```
import arcpy

# Create a Describe object
#
desc = arcpy.Describe("C:/data/moad.gdb/Water_Bodies")

# Print RepresentationClass properties for each representation
#   in the feature class.
#
for child in desc.representations:
    if child.datasetType == "RepresentationClass":
        print child.name
        print "\t%-25s %s" % ("Override field name:",
child.overrideFieldName)
        print "\t%-25s %s" % ("Shape override required:",
child.requireShapeOverride)
        print "\t%-25s %s" % ("RuleID field name:",
child.ruleIDFieldName)
```

Schematic Dataset properties (arcpy)

Top

Summary

The **`Describe`** function returns Dataset Properties for Schematic Datasets.
A Schematic Dataset returns a **`dataType`** of `"SchematicDataset"`.

Schematic Diagram properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Schematic Diagrams. Dataset Properties are also supported.
A Schematic Diagram returns a **dataType** of "SchematicDiagram".

Properties

Property	Explanation	Data Type
diagramClassName (Read Only)	The name of the schematic diagram class associated with the diagram.	String

Code Sample

Schematic Diagram properties example (stand-alone script)

The following stand-alone script displays the diagramClassName property of a schematic diagram.

```
import arcpy

# Create a Describe object for a schematic diagram
#
desc = arcpy.Describe("C:/data/blanding.gdb/CityPower/Feeders/Feeder
0801-Rice Creek")

# Print the diagram class name property
#
print "Diagram Class Name: " + desc.diagramClassName
```

Schematic Folder properties (arcpy)

Top

Summary

The **`Describe`** function returns Dataset Properties for Schematic Folders.
A Schematic Folder returns a **`dataType`** of `"SchematicFolder"`.

SDC FeatureClass properties (arcpy)

Top

Summary

The **`Describe`** function returns GDB FeatureClass Properties, FeatureClass Properties, GDB Table Properties, Table Properties, and Dataset Properties for SDC Feature Classes.
A SDC Feature Class returns a **`dataType`** of `"FeatureClass"`.

Shapefile FeatureClass properties (arcpy)

Top

Summary

The **`Describe`** function returns FeatureClass Properties, Table Properties and Dataset Properties for Shapefiles.
A Shapefile returns a **`dataType`** of `"ShapeFile"`.

Table properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Tables. Dataset Properties are also supported. Editor Tracking Properties are supported if editor tracking has been enabled for this table.
Table properties are available in many types of Describe objects.

Properties

Property	Explanation	Data Type
hasOID (Read Only)	Indicates whether the table has an ObjectID field.	Boolean
OIDFieldName (Read Only)	The name of the OID field if it exists.	String
fields (Read Only)	A Python list of Field objects for this table. This is the same as using the ListFields function.	Field
indexes (Read Only)	A Python list of Index objects for this table. This is the same as using the ListIndexes function.	Index

Code Sample

Table properties example (stand-alone script)

The following stand-alone script displays the OID field name if the table has one. It then prints the name and type for each field in the table.

```
import arcpy

# Create a Describe object from the table.
#
desc = arcpy.Describe("C:/data/chesapeake.gdb/munich")

# If the table has an OID, print the OID field name
#
if desc.hasOID:
    print "OIDFieldName: " + desc.OIDFieldName

# Print the names and types of all the fields in the table
#
for field in desc.fields:
    print "%-22s %s %s" % (field.name, ":", field.type)
    #print field.name + " = " + field.type
```

TableView properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Table Views. Table Properties are also supported.
A Table View returns a **dataType** of "TableView".

Properties

Property	Explanation	Data Type
table (Read Only)	A Describe object of the Table associated with the table view	Describe
FIDSet (Read Only)	A semicolon-delimited string of selected feature IDs (record numbers)	String
fieldInfo (Read Only)	The FieldInfo object (property set) of the table	FieldInfo
whereClause (Read Only)	The table view selection WHERE clause	String
nameString (Read Only)	The name of the table view	String

Code Sample

TableView properties example (stand-alone script)

The following stand-alone script creates an in memory TableView from a feature class. It then displays some of the properties for the TableView.

```
import arcpy

# Create a table view from a feature class
#
arcpy.MakeTableView_management(
        "C:/data/wellingham.gdb/water/water_pipes",
        "pipes_view")

# Create a Describe object from the table view
#
desc = arcpy.Describe("pipes_view")

# Print some table view properties
#
print "Table View Name: " + desc.nameString
print "Where Clause:    " + desc.whereClause
print "Table Name:      " + desc.name
```

Text File properties (arcpy)

Top

Summary

If the text file contains tabular data, the **Describe** function returns Table Properties and Dataset Properties for Text Files, otherwise refer to File Properties.

A Text File returns a **dataType** of "TextFile".

Code Sample

Text File properties example (stand-alone script)

The following stand-alone script displays some Text File properties for a text file that has tabular data.

```
import arcpy

# Create a Describe object from the text file.
#
desc = arcpy.Describe("C:/data/evac_table.txt")

# Print some table properties
#
print "HasOID: " + str(desc.hasOID)

print "\nField names:"
for field in desc.fields:
    print "   " + field.name
```

TIN properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for TINs. Dataset Properties are also supported.
A TIN returns a **dataType** of "Tin".

Properties

Property	Explanation	Data Type
fields (Read Only)	A Python List containing Field objects for the TIN dataset	Field
hasEdgeTagValues (Read Only)	Indicates whether the TIN dataset has edge tag values	Boolean
hasNodeTagValues (Read Only)	Indicates whether the TIN dataset has node tag values	Boolean
hasTriangleTagValues (Read Only)	Indicates whether the TIN dataset has triangle tag values	Boolean
isDelaunay (Read Only)	Indicates whether the TIN dataset was constructed using Delaunay triangulation	Boolean
ZFactor (Read Only)	Multiplication factor applied to all z-values in a TIN to provide unit congruency between coordinate components	Integer

Code Sample

TIN properties example (stand-alone script)

The following stand-alone script displays properties for a TIN. It also prints all the field names for the TIN.

```
import arcpy

# Create a Describe object
#
desc = arcpy.Describe("C:/data/antelope_island")

# Print TIN properties
print "%-21s %s" % ("HasEdgeTagValues:", desc.hasEdgeTagValues)
print "%-21s %s" % ("HasNodeTagValues:", desc.hasNodeTagValues)
print "%-21s %s" % ("HasTriangleTagValues:",
desc.hasTriangleTagValues)
print "%-21s %s" % ("IsDelaunay:", desc.isDelaunay)
print "%-21s %s" % ("ZFactor:", desc.ZFactor)

# Print the field names in the TIN
print "\nFields in the TIN:"
for field in desc.fields:
    print "\t%s" % field.name
```

Tool properties (arcpy)

Top

Summary

The **`Describe`** function returns Dataset Properties for Tools.
A Tool returns a **`dataType`** of `"Tool"`.

Toolbox properties (arcpy)

Top

Summary

The **`Describe`** function returns Dataset Properties for Toolboxes.
A Toolbox returns a **`dataType`** of `"ToolBox"`.

Topology properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Topologies. Dataset Properties are also supported.

A Topology returns a **dataType** of "Topology".

Properties

Property	Explanation	Data Type
clusterTolerance (Read Only)	The cluster tolerance of the topology	Double
featureClassNames (Read Only)	A Python list containing the names of the feature classes participating in the topology	String
maximumGeneratedErrorCount (Read Only)	The maximum number of errors to generate when validating a topology	Double
ZClusterTolerance (Read Only)	The z cluster tolerance of the topology	Double

Code Sample

Topology properties example (stand-alone script)

The following stand-alone script displays properties for a topology.

```
import arcpy

# Create a Describe object from a topology.
#
desc = arcpy.Describe("C:/data/moad.gdb/East/ParkZones_topology")

# Print some topology properties
#
print "%-27s %s" % ("ClusterTolerance:", desc.clusterTolerance)
print "%-27s %s" % ("ZClusterTolerance:", desc.ZClusterTolerance)
print "%-27s %s" % ("FeatureClassNames:", desc.featureClassNames)
print "%-27s %s" % ("MaximumGeneratedErrorCount:",
desc.maximumGeneratedErrorCount)
```

VPF Coverage properties (arcpy)

Top

Summary

The **`Describe`** function returns Dataset Properties for VPF Coverages.
A VPF Coverage returns a **`dataType`** of `"VPFCoverage"`.

VPF FeatureClass properties (arcpy)

Top

Summary

The **`Describe`** function returns FeatureClass Properties, Table Properties, and Dataset Properties for VPF Feature Classes.
A VPF Feature Class returns a **`dataType`** of `"FeatureClass"`.

VPF Table properties (arcpy)

Top

Summary

The **`Describe`** function returns Table Properties and Dataset Properties for VPF Tables.

A VPF Table returns a **`dataType`** of `"VPFTable"`.

Workspace properties (arcpy)

Top

Summary

The **Describe** function returns the following properties for Workspace. A Workspace returns a **dataType** of "Workspace".

Properties

Property	Explanation	Data Type
connectionProperties (Read Only)	The connectionProperties is a property set. The connection properties for an enterprise geodatabase workspace will vary depending on the type of SDE database being used. Possible properties include: • authentication_mode—Credential authentication mode of the connection. Either OSA or DBMS. • database—Database connected to. • historical_name—The historical marker name of the historical version connected to. • historical_timestamp—A date time that represents a moment timestamp in the historical version connected to. • is_geodatabase—String. Returns true if the database has been enabled to support a geodatabase; otherwise, it's false. • instance—Instance connection to. • server—SDE server name connected to. • user—Connected user. • version—The transaction version name of the transactional version connected to. Only one of historical_name, historical_timestamp, or version exists for any given workspace.	Object
connectionString (Read Only)	The connection string being used in conjunction with the SDE database type. For any other workspace type, returns an empty string.	String
currentRelease (Read Only)	For a geodatabase workspace, returns True if the geodatabase's version is current. **currentRelease** can be used to assess if the geodatabase can be upgraded.	Boolean
domains (Read Only)	A Python List containing the geodatabase domain names. To work with these domain names, you can use tools from the Domains toolset.	String
release (Read Only)	For a geodatabase workspace, returns the geodatabase release value. Below is the mapping of geodatabase release values to ArcGIS version numbers. **Geodatabase release value** / **ArcGIS version** 2,2,0 / 9.2 2,3,0 / 9.3, 9.3.1 3,0,0 / 10.0, 10.1	String
workspaceFactoryProgID (Read Only)	The ID is a string. You can use this to distinguish between specific workspace types with a finer granularity than you can with workspaceType. For example, workspaceFactoryProgID can distinguish between a file geodatabase and a personal geodatabase. Using workspaceType, you cannot make that distinction. Following are **workspaceFactoryProgID** strings returned for the common workspace types: • esriDataSourcesGDB.AccessWorkspaceFactory.1 — Personal geodatabase • esriDataSourcesGDB.FileGDBWorkspaceFactory.1 — File geodatabase • esriDataSourcesGDB.InMemoryWorkspaceFactory.1 — in_memory workspace • esriDataSourcesGDB.SdeWorkspaceFactory.1 —SDE geodatabase • (empty string) —Other (shapefile, coverage, CAD, VPF, and so on)	String
workspaceType (Read Only)	The workspace type. • FileSystem —File-based (coverage, shapefile, and so forth) workspaces and in_memory workspaces • LocalDatabase —Geodatabases that are local (a file or personal geodatabase) • RemoteDatabase —Geodatabases that require a remote connection (ArcSDE, OLE DB, and so forth)	String

Code Sample

Workspace properties example (stand-alone script)

The following stand-alone script displays some workspace properties for an SDE database.

```
import arcpy
# Create a Describe object for an SDE database
desc = arcpy.Describe(r"C:data\Connection to state.sde")
# Print workspace properties
print "%-24s %s" % ("Connection String:", desc.connectionString)
print "%-24s %s" % ("WorkspaceFactoryProgID:",
desc.workspaceFactoryProgID)
print "%-24s %s" % ("Workspace Type:", desc.workspaceType)
# Print Connection properties
cp = desc.connectionProperties
print "\nDatabase Connection Properties:"
print "%-12s %s" % ("  Server:", cp.server)
print "%-12s %s" % ("  Instance:", cp.instance)
print "%-12s %s" % ("  Database:", cp.database)
print "%-12s %s" % ("  User:", cp.user)
print "%-12s %s" % ("  Version:", cp.version)
# Print workspace domain names
domains = desc.domains
print "\nDomains:"
for domain in domains:
    print "\t" + domain
```