GISPython Documentation

Release 1.41.1

lvmgeo

May 12, 2020

Contents

LVM GEO Python Core (*GISPython*) is an open source automation and scripting core developed by the LVM GEO team. Based on this core any developer can build simple and structured Python programming scripts with a rich functionality. The programming core allows management of all system's maintenance processes within a unified environment. *GISPython* project can be found at GitHub.

Attention: There is undergoing code arrangement according to PEP 8, so module and some code object names can be changed. All changes will be documented in changelog.

CHAPTER 1

Table of contents

1.1 General information

1.1.1 Installation

Dependencies

- ArcGIS 10.x /recommended with newest patches and service packs/ (*GISPython* is currently running on production systems based on ArcGIS 10.2.1, ArcGIS 10.3.1 and has been tested on ArcGIS 10.6.1)
- Python 2.7 (included in ArcGIS installation) (arcpy and numpy modules included)
- Additional python modules:
  - PyCrypto (manual installation)
  - NTLM: `pip install python-ntlm` (included in package setup process)
  - Paramiko: `pip install paramiko` (included in package setup process)
  - patool: `pip install patool` (included in package setup process)
  - simpleJson: `pip install simplejson` (included in package setup process)

Package installation

GISPython package is available on the Python Package Index, so you can get it via pip:

```
pip install GISPython
```

Note: If pip isn't installed, you can get it here!

1.1.2 Configuration & basic usage

Before using *GISPython* modules in custom geoprocessing scripts, you need to set up your scripting environment with *SetupDefaultEnvironment* module which also includes template for user scripts.

SetupDefaultEnvironment module also includes basic parameters (variable *paramsFileSource*) for parameter file (e.g. SysGISParams.py) which is important, because *GISPython* relies of several parameters to be present to function successfully:

- OutDir - directory for storing script output log files `OutDir = r'C:\GIS\Log\Outlog\'`
- OutDirArh - directory for storing script output log file archive (all non active files) `OutDirArh = r'C:\GIS\Log\Outlog\Archive\'`
- ErrorLogDir - directory for storing script error log files `ErrorLogDir = r'C:\GIS\Log\ErrorLog\'` *(Important! This directory can be monitored for non empty files. If this directory has a file that is non empty - this indicates that a script has failed)*
- ErrorLogDirArh - directory for storing script error log files `ErrorLogDirArh = r'C:\GIS\Log\ErrorLog\Archive'`
- TmpFolder - Temp folder `TmpFolder = r'C:\GIS\tmp'`
- encodingPrimary - encoding of Windows shell `encodingPrimary = 'cp775'`
- encodingSecondary - encoding of Windows unicode language used `encodingSecondary = 'cp1257'`
- SetLogHistory - enable or disable Geoprocessing history logging `SetLogHistory = False`

Note: It is recommended to define additional script parameters in SysGISParams.py file, to keep the main code clean. Our approach is to define all the parameters that define current system environment be kept in this one file. In case of moving environment (e.g. test system and production system) this one file has the specific connections and can be easily modified without changing the scripts.

Recommendations

Set up the variables at the beggining of the main function, to shorten the main code:

```
Tool = self.Tool
gp = Tool.gp
callGP = Tool.callGP
pj = os.path.join
```

Basic operations

ArcPy function call:

```
gpCaller = self.Tool.callGP
slay = 'some_layer'
callGP('AddField_management', slay, 'Day_TXT', 'TEXT', '#', '#', 10)
callGP('AddField_management', slay, 'CAR', 'TEXT', '#', '#', 128)
callGP('AddField_management', slay, 'WorkID', 'DOUBLE', 12, 0)
callGP('AddField_management', slay, 'REC_DATE_FROM', 'DATE')
```

Tool message output:

```
Tool = self.Tool
self.Tool.AddMessage(u'This is a message')
self.Tool.AddWarning(u'This is a warning')
self.Tool.AddError(u'This is an error')
```

1.2 Modules and examples

1.2.1 Main modules

GISPython package core modules

GISPythonModule

Main module, which contains frame for all GISPython package modules. Module allows the code unification, and ensures the code execution from both the ArcGIS Desktop Python console and the Command Prompt.

Module for the GISPython module frame and code unification

class GISPythonModule.**GISPythonModule**(*ToolName*, *SysGISParams*, *ExecutePatch='/home/docs/checkouts/readthedocs.org/user_builds/gispython/chec*, *statusMailRecipients=[]*, *errorMailRecipients=[]*, *licenceLevel='arceditor'*, *toollevel='full'*)

Bases: object

Interface class for all the GISPython modules. Interface class allows the code unification, and ensures the code execution from both the ArcGIS Desktop Python console and the Command Prompt.

Standalone tool execution: SetName('Tool Name') DoJob()

The tool executes within an another tool: SetTool(reference to SysGISTools.GISTools10) Get the tool name with the command 'PrintText()' mainModule()

DoJob()
Procedure which runs the tool with environment preparation and deletion (in case the tool runs as a standalone tool)

MyDispose()
Procedure which closes the tool environment after running it (in case the tool runs as a standalone tool)

MyEnd()
Procedure for the tool end message output, if the tool runs as a standalone tool

PrintText()
The auxiliary procedure for the tool name output

SetTool(*Tool*)
Sets up the GISPython environment object, if the tool runs within an another tool

initModule()
Procedure which initializes the GISPython environment, if the tool runs as a standalone tool

mainModule()
Rewritable procedure which contains the logic of the module

runInsideJob(*Tool*)
Procedure executes the tool, if it's to be executed within an another Python tool

Parameters

- **self** – The reserved object ‘self’
- **Tool** – The tool name to execute

class GISPythonModule.**GISPythonModuleArgsHelper**(*InitValue=None*)
Bases: object

Class for handling the argument passing for the module in three different places:

1. In the class initialization process.
2. In the arguments.
3. In the “main” operation call.

GetResultValue(*asBool=False*, *Default=None*)
Procedure makes a choice from the given attributes

Parameters

- **self** – The reserved object self
- **value** – Setup value

SetInitValue(*value*)
Procedure processes a parameter setup from the tool initialization procedure

Parameters

- **self** – The reserved object ‘self’
- **value** – Setup value

SetMainModuleValue(*value*)
Procedure processes a parameter setup from the base module of the tool

Parameters

- **self** – The reserved object ‘self’
- **value** – Setup value

processArgument(*argumentNumber=1*)
Procedure processes a parameter acquisition from the argument

Parameters

- **self** – The reserved object ‘self’
- **argumentNumber** – Argument from which to read the data

Examples

Executes the tool if it’s to be executed within an another Python tool:

```
from  GISPython import TimerHelper
import OnlineCompress

with TimerHelper.TimedSubprocess(Tool, u'Compress DB'): # Adds a message to the tool␣
↪output
        with TimerHelper.TimedSubprocess(Tool, u'disconnect users from DB', 2): #␣
↪Adds a message to the tool output
```

(continues on next page)

(continued from previous page)

```
                self.Tool.RunSQL('KillUsers', Pr.u_sde, Pr.p_sde) # Runs custom SQL␣
↪script
        OnlineCompress.MainModule(None, False).runInsideJob(Tool) # Runs SDE␣
↪Compression procedure from OnlineCompress module (custom geoprocessing module)
```

GISPythonTool

Module defines abstract classes for the ESRI Toolbox tool definition and contains functions which helps to create an ArcGIS Toolbox, validates the tool's parameter values and controls a behavior of the tool's dialog.

Module defines abstract classes for the ESRI Toolbox tool definition

class GISPythonTool.**GISPythonTool**
Bases: object

Class which helps to create an ArcGIS Toolbox

execute(*parameters*, *messages*)
Executes a tool

getParameterInfo()
Define parameter definitions

isLicensed()
Determines if the tool is licenced for execution

updateMessages(*parameters*)
This method is called after an inner validation, and is intended for carrying out an additional validations

updateParameters(*parameters*)
This method is called in case the parameters are changed, and is used for setting up the parameters

class GISPythonTool.**ToolValidator**(*parameters*)
Bases: object

Class for validating the tool's parameter values and controlling the behavior of the tool's dialog.

initializeParameters()
Refine properties of the tool's parameters. This method is called when the tool is opened.

updateMessages()
Modify the messages created by internal validation for each tool parameter. This method is called after internal validation.

updateParameters()
Modify the values and properties of parameters before internal validation is performed. This method is called whenever a parameter has been changed.

Examples

Define parameters in ESRI Python toolbox:

```
    from  GISPython import GISPythonTool

    class ToolAtributeValidator(GISPythonTool.GISPythonTool):
def __init__(self):
    """Define tool (tool name is the class name)"""
```

(continues on next page)

(continued from previous page)

```
    self.label = u"Tool for attribute data validation (test mode)"
    self.description = u"Tool for attribute data validation (test mode)"
    self.category = 'GEO Maintenance'

def getParameterInfo(self):
    """Define the parameters"""
    param_0 = arcpy.Parameter(
        displayName=r'Key:',
        name=u"key",
        datatype=u"String",
        parameterType=u"Required",
        direction=u"Input")

    ret_val = [param_0]
    return ret_val

def execute(self, parameters, messages):
    """Tool execution"""
    AtributeValidator.MainModule(parameters[0].valueAsText).DoJob()
    return
```

GISPythonToolBase

Module contains geoprocessing tool operations and automation mechanisms for different operations.

Base class for GISPython Tool object

class GISPythonToolBase.**GISPythonToolBase**(*ToolName*, *Params*)

Bases: object

AchiveFiles(*Dir*, *AchiveDir*, *FileName*, *PrintOut=True*)

Function moves log files to the archive

Parameters

- **self** – The reserved object 'self'
- **Dir** – Directory from which to archive
- **AchiveDir** – Archive directory
- **FileName** – Parameter for searching a file (full or partial filename)

AddError(*strMessage*, *newline=True*)

Procedure for the GP object error message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object 'self'
- **strMessage** – Output message text

AddMessage(*strMessage*, *newline=True*)

Procedure for a message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object 'self'
- **strMessage** – Output message text
- **newline** – Flag that marks that the output must be continued in a new line

AddWarning(*strMessage*, *newline=True*)
Procedure for the GP object warning message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object 'self'
- **strMessage** – Output message text

AutorizeNTWLocation(*Adress*, *user*, *pwd*)
Network directory authorization

Parameters

- **self** – The reserved object 'self'
- **Adress** – Network adress
- **user** – Username
- **pwd** – Password

CorrectStr(*Str*)
Function doubles symbol '' in path for some external execution compatibility :param self: The reserved object 'self' :param Str: Input string

Returns string

GetPS(*Name*, *Path='#'*)
Function gets the PowerShell file location from the installation directory

Parameters

- **self** – The reserved object 'self'
- **Name** – Filename without an extension

GetSQL(*Name*)
Function gets the SQL file location from the installation directory

Parameters

- **self** – The reserved object 'self'
- **Name** – Filename without an extension

Returns SQL file full path

MyDateForParam(*paramStr*, *includeTime=False*)
Function returns date from GEO parameter processing saved string

Parameters

- **self** – The reserved object 'self'
- **paramStr** – Parameter value as text
- **includeTime** – Include time in param (True/False)

Returns datetime

MyDateFromParam(*dateString*, *includeTime=False*)
Function converts date written in the parameter file to a date object

Parameters

- **self** – The reserved object 'self'
- **includeTime** – Include time in param (True/False)

MyDispose()
Disposal of the class

Parameters **self** – The reserved object 'self'

MyEnd()
End of the process

Parameters **self** – The reserved object 'self'

MyNow()
Function returns formatted date for the output

Parameters **self** – The reserved object 'self'

Returns Date, formatted as text

MyNowFile()
Function returns formatted date for the filename output

Parameters **self** – The reserved object 'self'

Returns Date, formatted as text

MyNowFileSafe()
Function returns formatted date for the filename output (additional compatibility)

Parameters **self** – The reserved object 'self'

Returns Date, formatted as text

MyNowForParam(*minusdays=0*, *includeTime=False*)
Function returns formatted date (date now) for the GEO parameter processing

Parameters

- **self** – The reserved object 'self'
- **minusdays** – Number of days to subtract from today
- **includeTime** – Include time in param (True/False)

Returns Date, formatted as text

MyNowOracle()
Function returns formatted date for the data selection in SQL

Parameters **self** – The reserved object 'self'

Returns Date, formatted as text

MyNowUTC()
Function returns formatted date for the UTC date output

Parameters **self** – The reserved object 'self'

Returns Date, formatted as text

RunPS(*Name*, *Args*, *Path='#'*)
Procedure for the PowerShell file execution

Parameters

- **self** – The reserved object 'self'
- **Name** – Filename without an extension
- **Args** – Arguments

RunSQL(*Name, user='#', pwd='#', SpoolFile='#', ErrorStrings=['ERROR', 'FAILED', u'K\u013b\u016aDA', 'EXCEPTION', 'ORA-'], params=[], DBType='Oracle', hidenStrings=[], DBName='#'*)

Procedure for SQL file execution (only Oracle sqlplus supported) Typically used for execution, passing only SQL filename parameter

Parameters

- **self** – The reserved object 'self'
- **Name** – Filename without an extension
- **u** – Username
- **p** – Password
- **SpoolFile** – SQL output
- **ErrorStrings** – A list of keyword error strings that defines an error in the execution
- **params** – aditional parameters
- **DBType** – only Oracle is supported
- **hidenStrings** – List of strings tha has to be hiden in the output (used for hiding passwords)
- **DBName** – Oracle TNS name

_outputLines(*lines, doMessges, noErr=False, ErrorStrings=['ERROR', 'FAILED', u'K\u013b\u016aDA', 'EXCEPTION', 'ORA-'], Silent=False, hidenStrings=[]*)

Procedure for outputing set of lines to screen with error key word recognition. (for example for log file output processing)

Parameters

- **self** – The reserved object 'self'
- **lines** – Lines to process
- **noErr** – True ir no error logging is necesery
- **ErrorStrings** – List or keywords with will be recognized as errors
- **Silent** – if True no Errors will be rised
- **hidenStrings** – List of strings tha has to be hiden in the output (used for hiding passwords)

_runProcess(*exe*, *noErr=False*, *Detached=False*, *Silent=False*, *hidenStrings=[]*)

Shell command execution support function (see the runShell function)

Parameters

- **self** – The reserved object 'self'
- **exe** – Executable command

Returns stdoutdata

_tryCovertStringEncoding(*txt*)

Function for working with strings in diferent encodings. Converts string from input to string in correct encoding.

Parameters

- **self** – The reserved object 'self'

- **txt** – String to be converted

Returns converted string

outputLogfile(*file, encoding='utf8', noErr=False, ErrorStrings=['ERROR', 'FAILED', u'K\u013b\u016aDA', 'EXCEPTION', 'ORA-'], Silent=False, hidenStrings=[]*)
Procedure prints text file to screent - processing error keywords

Parameters

- **self** – The reserved object 'self'
- **file** – path to file to process
- **noErr** – True ir no error logging is necesery
- **ErrorStrings** – List or keywords with will be recognized as errors
- **Silent** – if True no Errors will be rised
- **hidenStrings** – List of strings tha has to be hiden in the output (used for hiding passwords)

runShell(*exe, noErr=False, ErrorStrings=['ERROR', 'FAILED', u'K\u013b\u016aDA', 'EXCEPTION', 'ORA-'], Detached=False, Silent=False, hidenStrings=[]*)
Shell command execution procedure. It can detect errors in execution and can output results to screen.

Parameters

- **self** – The reserved object 'self'
- **exe** – Executable command
- **noErr** – 'True' indicates that the errors doesn't have to be logged
- **ErrorStrings** – List of error strings to look for in the output. Found error will be considered as an error in the execution process
- **Detached** – Whether to execute seperately from the main process (Default: False)
- **hidenStrings** – List of strings tha has to be hiden in the output (used for hiding passwords)

run_with_limited_time(*func*, *args*, *kwargs*, *time*)
Runs a function with time limit

Parameters

- **self** – The reserved object 'self'
- **func** – The function to run
- **args** – The functions args, given as a tuple
- **kwargs** – The functions args, given as a tuple
- **time** – The time limit in seconds

Returns True if the function ended successfully. False if it was terminated.

Examples

Run a script from Shell with parameters:

```
Tool = self.Tool
# Executes your custom process script (mainly maintenance scripts) within runShell␣
↪function,
# which implements _runProcess function (message output in terminal window).
# Function executes script seperately from main process (Detached=True) and indicates,
# that the errors doesn't have to be logged (noErr=True).
Tool.runShell('SomeProcess.py', Detached = True, noErr = True)
time.sleep(10) # after 10 seconds launch another runShell process
```

Executes a custom SQL script file (only Oracle sqlplus supported):

```
from GISPython import TimerHelper

Tool = self.Tool
# Executes process from SQL file
with TimerHelper.TimedSubprocess(self.Tool, u'datu atlasi no nogabaliem'): # Adds a␣
↪message to the tool output
        Tool.RunSQL('LoadSomeDataFromTable') # Runs SQL file within RunSQL function,␣
↪which implements GetSQL function (gets the SQL file location)
```

Duplicates path seperator symbol ‘’ for external execution compatibility:

```
from GISPython import SimpleFileOps
from GISPython import TimerHelper

Tool = self.Tool
# Your code
with TimerHelper.TimedSubprocess(Tool, u'prepare environment'): # Adds a message to␣
↪the tool output
        DirHelper = SimpleFileOps.SimpleFileOps(Tool) # Set variable for␣
↪SimpleFileOps module functions
        tmpFolder = os.path.join(Pr.TmpFolder, "Soil") # Set tmp folder path
        # Your code
        Tool.AddMessage(u'\n        ...delete previous tmp data') # Add tool output␣
↪message
        DirHelper.CheckCreateDir(Tool.CorrectStr(tmpFolder)) # Check/create tmp␣
↪directory (with modified path seperators)
        DirHelper.ClearDir(Tool.CorrectStr(tmpFolder)) # Clear tmp directory (with␣
↪modified path seperators)
```

SysGISTools

Base module which contains GISPython scripting framework, geoprocessing message delivery, logging and error processing. Inherits GISPythonToolBase.

GIS function support module

class SysGISTools.**GISTools10**(*ToolName*, *Params*, *licenceLevel='arceditor'*)

Bases: *GISPythonToolBase.GISPythonToolBase*

Class for storing the auxiliary batch processing and GIS geoprocesing functions

AddError(*strMessage*, *newline=True*)

Procedure for the GP object error message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object ‘self’

- **strMessage** – Output message text

AddMessage(*strMessage*, *newline=True*)
Procedure for a message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object 'self'
- **strMessage** – Output message text
- **newline** – Flag that marks that the output must be continued in a new line

AddWarning(*strMessage*)
Procedure for the GP object warning message output (screen, logfile and if necessary e-mail)

Parameters

- **self** – The reserved object 'self'
- **strMessage** – Output message text

MyDispose()
Disposal of the class

Parameters **self** – The reserved object 'self'

OutputErrors()
Procedure to output messages and errors stored in the GP object. !!! Depricated - Left for backwards compatibility - use OutputMessages with ErrorSeverity 0 !!!

Parameters **self** – The reserved object 'self'

OutputMessages(*ErrorSeverity=2*)
Procedure to output messages stored in the GP object

Parameters

- **self** – The reserved object 'self'
- **ErrorSeverity** – Maximum Severity to report as error

callGP(*functionName*, **args*)
Function to call the arcPy GP functions with automatic output messge display and output result returning

Parameters

- **self** – The reserved object 'self'
- **functionName** – Name of the arcPy GP function
- **args** – arguments as Tuple

callGPSilent(*functionName*, **args*)
Function to call the arcPy GP functions without output

Parameters

- **self** – The reserved object 'self'
- **functionName** – Name of the arcPy GP function
- **args** – arguments as Tuple

class SysGISTools.**MySR**
Class for storing often used coordinate system parameters

SysGISToolsSysParams

Module for storing the scripting parameters

SysTools_unittest

Test procedure module

class SysTools_unittest.**GISTools_unittest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

GEOPython unit test class

setUp()
The Test setting up procedure

tearDown()
"The Test tear down - cleaning up objects after test

test_Tool_init()
Check if it is possible to get the geoprocessor object

test_shellRun()
Check the shell execution commands

class SysTools_unittest.**Pr**
Bases: object

Defines test parameters

ErrorLogDir = 'C:\\GIS\\Log\\ErrorLog\\'

ErrorLogDirArh = 'C:\\GIS\\Log\\ErrorLog\\Archive\\'

OutDir = 'C:\\GIS\\Log\\Outlog\\'

OutDirArh = 'C:\\GIS\\Log\\Outlog\\Archive\\'

encodingPrimary = 'cp1257'

encodingSecondary = 'cp775'

1.2.2 Helper modules

Additional *GISPython* package modules

AGServerHelper

Module contains procedures for typical operations with ArcGIS server. All procedures use token authorization. For procedures with NTLM authorization use AGServerHelperNTLM module.

Module for operations with ArcGIS Server services

class AGServerHelper.**AGSServerHelper**(*username*, *password*, *serverName*, *serverPort=6080*, *Tool=None*, *https=False*)
Bases: object

GetServerJson(*token*, *serverName*, *serverPort*, *serverService*)
Retrieve service parameters

Parameters

- **self** – The reserved object 'self'
- **token** – Token
- **serverName** – Server name
- **serverPort** – Server port
- **serverService** – Service which parameter configuration shall be retrieved

IsServiceRunning (*folder*, *service*)
Retrieve the service status from the server

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service

PublishServerJson (*service*, *serverName*, *dataObj*, *token*, *serverPort*)
Publish service parameters to server

Parameters

- **self** – The reserved object 'self'
- **service** – Service which parameter configuration shall be renewed
- **serverName** – Server name
- **dataObj** – Parameter configuration
- **token** – Token
- **serverPort** – Server port

StartService (*folder*, *service*)

StartStopService (*folder*, *service*, *action*)

StopService (*folder*, *service*)

assertJsonSuccess (*data*)
A function that checks that the input JSON object is not an error object.

Parameters

- **self** – The reserved object 'self'
- **data** – JSON data object

genToken (*adminUser*, *adminPass*, *server*, *port*, *expiration=60*)
Create ArcGIS server connection file

Parameters

- **server** – Server name
- **port** – Server port
- **adminUser** – Username
- **adminPass** – Password

getServiceFromServer (*services*, *service*, *serviceDir*)
Retrieve the full service name from the server

Parameters

- **self** – The reserved object 'self'
- **services** – List of all services on server
- **service** – Name of the service from which to get corresponding name from the server services list
- **serviceDir** – Name of the service directory which is shown in the configuration of services to be published on the server

getServiceList (*server*, *port*, *adminUser*, *adminPass*, *token=None*)
Retrieve ArcGIS server services

Parameters

- **self** – The reserved object 'self'
- **server** – Server name
- **port** – Server port
- **adminUser** – Username
- **adminPass** – Password
- **token** – Token (if created)

Examples

Check if ArcGIS Server service is running:

```
from GISPython import AGServerHelper

Tool = self.Tool
JsonParams = JsonParamsHelper.JsonParams(Tool, 'Config', 'ServiceCheck')
AGS = AGServerHelper.AGSServerHelper(Pr.AGS_u, Pr.AGS_p, Pr.AGS_Servers[0], Pr.AGS_
↪Port, self.Tool)
configData = JsonParams.GetParams()
dirs = configData[u'chechServiceList']
for dir in dirs:
        services = dir["services"]
        for service in services:
                serviceName = service["serviceName"]
                type = service["type"]
                description = service["description"]
                try:
                        running = AGS.IsServiceRunning(dir["folderName"], serviceName␣
↪+ "." + type)

                        if running:
                                Tool.AddMessage(u"\t{0}\\{1}.{2} ({3}) - OK".
↪format(dir["folderName"], serviceName, type, description))
                        else:
                                txt = u"\t{0}\\{1}.{2} ({3}) - Stopped".format(dir[
↪"folderName"], serviceName, type, description)
                                Tool.AddMessage(txt)
                                errorTxt += "\n" + txt

                except Exception, e:
                        tb = sys.exc_info()
```

(continues on next page)

(continued from previous page)

```
                        orgLine = "Line %i" % tb[2].tb_lineno
                        orgTraceback = unicode(traceback.format_exc(), errors='ignore
↪')

                        txt = u"\t{0}\\{1}.{2} ({3}) - Error - Line:{4} Error: {5} ".
↪format(dir["folderName"], serviceName, type, description, orgLine, orgTraceback)
                        Tool.AddMessage(txt)
                        errorTxt += "\n" + txt
```

AGServerHelperNTLM

Module contains procedures for typical operations with ArcGIS server. All procedures use NTLM authorization. For token authorization use AGServerHelpaer module.

Module for operations with ArcGIS Server services

class AGServerHelperNTLM.**AGServerHelperNTLM**(*username*, *password*, *ags_admin_url*, *tool=None*, *basic=False*, *allowunverifiedssl=False*)

Bases: object

Class for operations with ArcGIS Server services

GetDatasetNames(*folder*, *service*)

Retrieve the service Dataset Names from the server

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service

Returns: list of strings

GetDatasetNamesWithObjects(*folder*, *service*)

Retrieve the service Dataset Names from the server

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service

Returns: list of strings

GetRightsGroupsNames(*folder*, *service*)

Retrieve the service permission role names from service

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service

Returns: list of strings

GetServerJson(*server_service*)
Retrieve service parameters

Parameters

- **self** – The reserved object 'self'
- **serverService** – Service which parameter configuration shall be retrieved

Returns json data object

GetServiceInfo(*folder*)
Retrieve the Folder List from the server

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory

Returns: list of service objects

_AGServerHelperNTLM__assert_json_success(*data*)
Function for aserting json request state

Parameters

- **self** – The reserved object 'self'
- **data** – Request response string

Returns boolean False if request has errors

_AGServerHelperNTLM__process_folder_string(*folder*)
Function for processing folder name string

Parameters

- **self** – The reserved object 'self'
- **folder** – folder string

Returns corrected folder string

_AGServerHelperNTLM__request_from_server(*adress*, *params*, *content_type='application/json'*, *method='POST'*)
Function for ntlm request creation

Parameters

- **self** – The reserved object 'self'
- **adress** – Adress of request
- **params** – Params as dictionary
- **content_type** – Http content type
- **method** – Http method

Returns Response string

addRole(*rolename*, *description=''*)
Retrieve the Role Names from the server

Parameters

- **self** – The reserved object 'self'

- **rolename** – The name of the role. The name must be unique in the role store.

addServicePermisions (*folder*, *service*, *principal*, *is_allowed='true'*)
Add service permisions

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service
- **principal** – The name of the role for whom the permission is being assigned.
- **is_allowed** – Tells if access to a resource is allowed or denied.

addUsersToRole (*rolename*, *users*)
assign a role to multiple users with a single action

Parameters

- **self** – The reserved object 'self'
- **rolename** – The name of the role.
- **users** – A comma-separated list of user names. Each user name must exist in the user store.

getRoles (*pageSize=5000*)
Retrieve the Role Names from the server

Parameters **self** – The reserved object 'self'

Returns: list of strings

getServiceFromServer (*services*, *service*, *serviceDir*)
Retrieve the full service name from the server

Parameters

- **self** – The reserved object 'self'
- **services** – List of all services on server
- **service** – Name of the service from which to get corresponding name from the server services list
- **serviceDir** – Name of the service directory which is shown in the configuration of services to be published on the server

getServiceList (*folder*)
Retrieve ArcGIS server services

Parameters

- **self** – The reserved object 'self'
- **folder** – Folder of the service (ROOT for root services)

getServicePermisions (*folder*, *service*)
Check service permisions

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory

- **service** – Name of a service

Returns: Dictionary of service principals

getUsersWithinRole(*rolename*, *maxCount=5000*)
Retrieve the Role Names from the server

Parameters **self** – The reserved object 'self'

Returns: list of strings

isServiceRunning(*folder*, *service*)
Check if service is running

Parameters

- **self** – The reserved object 'self'
- **folder** – Service directory
- **service** – Name of a service

Returns: True if is running

publishServerJson(*service*, *data_object*)
Publish service parameters to server

Parameters

- **self** – The reserved object 'self'
- **service** – Service which parameter configuration shall be renewed
- **data_object** – Parameter configuration

removeRole(*rolename*)
Retrieve the Role Names from the server

Parameters

- **self** – The reserved object 'self'
- **rolename** – The name of the role.

removeUsersFromRole(*rolename*, *users*)
Removes a role assignment from multiple users

Parameters

- **self** – The reserved object 'self'
- **rolename** – The name of the role.
- **users** – A comma-separated list of user names. Each user name must exist in the user store.

startService(*folder*, *service*)
Starts AGS Service

Parameters

- **folder** (*string*) – AGS folder of the service. (CASE sensitive) Use ROOT for services without folder.
- **service** (*string*) – Service name (CASE sensitive)

stopService(*folder*, *service*)
Stops AGS Service

Parameters

- **folder** (*string*) – AGS folder of the service. (CASE sensitive) Use ROOT for services without folder.
- **service** (*string*) – Service name (CASE sensitive)

CachingHelper

Module generates and carries out map scale caching in ArcGIS Server services.

Carries out service caching on ArcServer providing x3 retry capabilities

class CachingHelper.**CachingHelper**(*Tool*, *vServer*, *vExtent='#'*, *vTerritoryLayer='#'*)
Support class which generates the caches

GenerateCache(*vService*, *vInstances*, *vCashScales*, *vFolder='#'*, *vDeleteScales='#'*)
Base procedure of the tool

Parameters

- **self** – The reserved object 'self'
- **vService** – Cacheable serviss
- **vInstances** – Cacheable instance count
- **vCashScales** – Cacheable scales
- **vFolder** – Cache service directory
- **vDeleteScales** – Scales to delete

Examples

Daily server caching procedure:

```
from GISPython import CachingHelper

GCache = CachingHelper.CachingHelper(self.Tool, Pr.ConnAGSCache, self.Tool.gp.
↪Extent(307950, 167920, 767480, 443890), Pr.ConnAuto + "\\SDEOWNER.CashLVBuffer")
GCache.GenerateCache('Nog_Dalplans_cache',8,'30000;20000;15000;10000;5000;2000',
↪'CacheDinamic')
GCache.GenerateCache('Nog_MezaudzuPlans_cache',8,'30000;20000;15000;10000;5000;2000',
↪'CacheDinamic')
```

FTPHleper

Module contains procedures for typical FTP server file operations.

FTP operations module

class FTPHleper.**FTPFile**(*file*, *date*, *size*)
Class for describing the FTP file

class FTPHleper.**FTPHleper**(*FTPHost*, *FTPUser*, *FTPPwd*, *FTPDir=None*)
Class for easing the FTP operations

delete_file(*fileName*)
Deletes the file from the FTP server

Parameters

- **self** – The reserved object 'self'
- **fileName** – Name of the file to delete

dir_callback(*line*)
Processes callback from the procedure 'list_files'

Parameters

- **self** – The reserved object 'self'
- **line** – Row with the FTP file description

get_file(*filename*, *savePath*)
Retrieves the file from the FTP server

Parameters

- **self** – The reserved object 'self'
- **filename** – Ftp file name

get_file_date(*fileName*)
Determines the ftp file modification date

Parameters

- **self** – The reserved object 'self'
- **fileName** – Ftp file name

get_file_size(*fileName*)
Determines the ftp file size

Parameters

- **self** – The reserved object 'self'
- **fileName** – Ftp file name

list_files()
Procedure to retrieve a file description in the specific connection directory

Parameters **self** – The reserved object 'self'

upload_file(*fileName*, *filePath*)
Uploads the binary file, using FTP

Parameters

- **self** – The reserved object 'self'
- **filePath** – Uploadable file local path
- **fileName** – Uploadable file name

Examples

Deletes old and uploads new files to FTP server:

```
from GISPython import FTPHleper

tmpFolder = pj(Pr.TmpFolder, "mobileInfo")
FTP = FTPHleper.FTPHleper(Pr.mobileFTPHost, Pr.mobileFTPuser, Pr.mobileFTPpwd, Pr.
↪mobileFTPmobileinfoFolder)
ftpfiles = FTP.list_files()

# Delete old files
for file in (f for f in ftpfiles if (f.file == 'file1.geojson' or f.file == 'file2.
↪geojson')):
        Tool.AddMessage(u'Delete file ' + file.file + ' from ftp ...')
        FTP.delete_file(file.file)

# Upload new files
FTP.upload_file('file1.geojson', tmpFolder)
Tool.AddMessage(u'\nfile ' + 'file2.geojson' + u' uploaded to ftp ...')
FTP.upload_file('file2.geojson', tmpFolder)
Tool.AddMessage(u'file ' + 'file2.geojson' + u' uploaded to ftp ...')
```

GDBHelper

Module for typical GDB (File Geodatabase) operations.

GDB operations module

class GDBHelper.**GDBHelper**(*gp*, *Tool=None*)

Class for easing the ESRI geodatabase operations

AddWarning(*str*)

CalculateXY(*Layer*, *Field*, *type*, *Query*)

Function for calculating the fields X and Y

Parameters

- **self** – The reserved object 'self'
- **Layer** – Input layer
- **Field** – The field to be calculated
- **type** – X or Y
- **Query** – Layer query

ClearData(*ConnDBSchema*, *ObjectName*, *ObjectType='TABLE'*)

Function eases feature deletion from layers and tables

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **ObjectName** – The table or layer containing features to be deleted
- **ObjectType** – Type: "TABLE" or "FC"

CopyDomain(*inWorkspace*, *inDomain*, *outWorkspace*, *outDomain*)

Procedure doeas copy one domain to another.

Parameters

- **self** – The reserved object 'self'
- **inWorkspace** – in workspace
- **inDomain** – in domain name
- **outWorkspace** – out workspace
- **outDomain** – out workspace name

CreateIndex(*ConnDBSchema*, *TABLENAME*, *FIELDNAME*)
Function eases work with the ESRI indexing function 'AddIndex_management'

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **Table** – The table
- **Field** – The field to be calculated

Decode(*val*, *DecodeDict*)
Function decodes one value to another

Parameters

- **self** – The reserved object 'self'
- **val** – Value to be decoded
- **DecodeDict** – Python Dictionary syntax which describes the decoding process - for example {'Key1':'Val1','Key2':'Val2'}

DecodeField(*Layer*, *Field*, *DecodeField*, *DecodeDict*, *Query=None*, *workspace=None*, *startEditing=False*, *startOperation=False*, *with_undo=False*, *multiuser=False*)
Function does field value recalculation decoding values from one to another. Used, for example, in clasificator value recalculation.

Parameters

- **self** – The reserved object 'self'
- **Layer** – Layer in which to decode
- **Field** – Field in which to decode
- **DecodeField** – Field from which to decode
- **DecodeDict** – Python Dictionary syntax which describes the decoding process - for example {'Key1':'Val1','Key2':'Val2'}
- **Query** – Layer query
- **workspace** – DB in which to start data editing
- **startEditing** – Start the edit session in the DB specified in the 'workspace' parameter
- **startEditing** – Start the edit operation in the DB specified in the 'workspace' parameter
- **with_undo** – Sets whether the undo and redo stacks are enabled or disabled for an edit session.
- **multiuser** – When False, you have full control of editing a nonversioned, or versioned dataset.

DelleteDomain(*ConnDBSchema*, *ObjectName*)
Function eases work with the ESRI domain deletion function 'DeleteDomain_management'

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **ObjectName** – The domain to be deleted

DelleteField(*ConnDBSchema*, *TABLENAME*, *FIELDNAME*)
Function eases work with the ESRI field deletion function DeleteField_management

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **Table** – Table
- **Field** – The field to be calculated

DelleteObject(*ConnDBSchema*, *ObjectName*, *ObjectType='#'*)
Function eases work with the ESRI object deletion function 'Delete_management'

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **ObjectName** – Object to be deleted
- **ObjectType** – Field to be deleted

GetDSConnectedElements(*ConnDBSchema*, *DBName*, *IncludeMe=False*)
Function defines all the tables which are linked with the FeatureDataset

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to DB schema
- **DBName** – DB name to process
- **IncludeMe** – Optional. Default = False. Indicates if DB classes will be returned in the returned list

Returns

- Result list with the found object classes

GetRelations(*ConnDBSchema*, *dfc*, *existingObjects*)
Auxiliary function for 'GetDSConnectedElements' function

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **dfc** – Describe 'FeatureClass' object
- **existingObjects** – List of existing objects

Returns List of the found objects

HasField(*ConnDBSchema*, *Table*, *Field*)
Function determines if field already exists in the table

Parameters

- **self** – The reserved object 'self'
- **ConnDBSchema** – Connection to the DB schema
- **Table** – Table
- **Field** – The field to be calculated

OutputMessages()

class GDBHelper.**RowHelper**(*gp*, *Tool=None*)
Bases: *GDBHelper.RowHelper2*

Row processing class

ValidateRowsForFieldValueList(*festureClass*, *getField*, *fields*, *valuelist*, *where_clause*, *outStringformat*)
Check rows if the field matches the unique value list

Parameters

- **self** – The reserved object 'self'
- **festureClass** – The feature class containing the rows to be searched
- **getField** – Field to check
- **fields** – A list of field names (order is important, because parameter 'outStringformat' is configured by this parameter !!! 'getField' value should be in this list!!!
- **valuelist** – A list of values
- **where_clause** – SQL WHERE clause to obtain the data
- **outStringformat** – Error output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values. For example - a field with the index 0 in the list of fields with 'outStringformat' parameter value: u"faulty feature OID: {0}" will return the string: u"faulty feature OID: 123", where 123 is the 'objectid' field value for found record.

ValidateRowsForSQLClause(*festureClass*, *fields*, *where_clause*, *outStringformat*)
Validate the rows with the SQL clause

Parameters

- **self** – The reserved object 'self'
- **festureClass** – The feature class containing the rows to be searched
- **fields** – A list of the field names (order is important, because the parameter 'outStringformat' is configured by this parameter)
- **where_clause** – SQL WHERE clause to obtain the data
- **outStringformat** – Error output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values. For example - a field with the index 0 in the list of fields with 'outStringformat' parameter value: u"faulty feature OID: {0}" will return the string: u"faulty feature OID: 123", where 123 is the 'objectid' field value for found record.

class GDBHelper.**RowHelper2**(*gp*, *Tool=None*)
Row processing class

GetUniqueValues(*festureClass*, *getField*, *where_clause=None*)
Get unique values from the field in the table (Only in DB)

Parameters

- **self** – The reserved object 'self'
- **festureClass** – The feature class containing the rows to be searched
- **getField** – The field from which to retrieve unique values
- **where_clause** – SQL WHERE clause to obtain the data

ValidateRowsForFieldValueList(*festureClass*, *getField*, *fields*, *valuelist*, *where_clause*, *outStringformat*, *idStringformat='{0}'*)
Check rows if the field matches the unique value list

Parameters

- **self** – The reserved object 'self'
- **festureClass** – The feature class containing the rows to be searched
- **getField** – Field to check
- **fields** – A list of field names (order is important, because parameter 'outStringformat' is configured by this parameter !!! 'getField' value should be in this list!!!
- **valuelist** – A list of values
- **where_clause** – SQL WHERE clause to obtain the data
- **outStringformat** – Error output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values. For example - a field with the index 0 in the list of fields with 'outStringformat' parameter value: u"faulty feature OID: {0}" will return the string: u"faulty feature OID: 123", where 123 is the 'objectid' field value for found record.
- **idStringformat** – Row id output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values.

Returns List containig list of two values - row id string and row error description string

ValidateRowsForSQLClause(*festureClass*, *fields*, *where_clause*, *outStringformat*, *idStringformat='{0}'*)
Validate the rows with the SQL clause

Parameters

- **self** – The reserved object 'self'
- **festureClass** – The feature class containing the rows to be searched
- **fields** – A list of the field names (order is important, because the parameter 'outStringformat' is configured by this parameter)
- **where_clause** – SQL WHERE clause to obtain the data
- **outStringformat** – Error output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values. For example - a field with the index 0 in the list of fields with 'outStringformat' parameter value: u"faulty feature OID: {0}" will return the string: u"faulty feature OID: 123", where 123 is the 'objectid' field value for found record.
- **idStringformat** – Row id output text on the found error. You can use the 'unicode.format' function notation {#} to transfer the row values.

Returns List containig list of two values - row id string and row error description string

class GDBHelper.**SimpleAppend**(*_Tool*, *_InWSp*, *_OutWSp*)
Class for easing the operations with the 'Append' function

Append(*inName*, *OutName*, *do_data_delete=True*)
Function for 'Append' operation automation

Parameters

- **self** – The reserved object 'self'
- **inName** – Input object name
- **OutName** – Output object name

GDBHelper.**uni**(*value*)

Examples

Clear old data from a table or a feature class and append new data:

```
from  GISPython import GDBHelper
from GISPython import TimerHelper

GDB = GDBHelper.GDBHelper(self.Tool.gp, self.Tool)
Appender = GDBHelper.SimpleAppend(self.Tool, Pr.ConnAuto, Pr.VAADConn)
t = TimerHelper.TimerHelper()
layer = 'Dataset.Table'

# Deletes old data
self.Tool.AddMessage(u'\n>>>> Begin the old file deletion procedure {0}'.format(Tool.
↪MyNow()))
GDB.ClearData(Pr.Connection, layer)
self.Tool.AddMessage(u'\n>>>> End the old file deletion procedure {0}'.format(t.
↪GetTimeReset()))

# Appends new data
self.Tool.AddMessage(u'\n>>>> Begin data appending procedure {0}'.format(Tool.
↪MyNow()))
Appender.Append('SDEOWNER.MKViews\\SDEOWNER.TABLEVIEW1', layer)
self.Tool.AddMessage(u'\n>>>> End data appending procedure {0}'.format(t.
↪GetTimeReset()))
```

Validate the rows with the SQL clause:

```
from GISPython import GDBHelper

RHelper = GDBHelper.RowHelper(self.Tool.gp, self.Tool)

# Validate rows
rezultList = RHelper.ValidateRowsForSQLClause(fc, validator[u'Fields'], validator[u
↪'Query'], validator[u'OutputPatern'])

# Output message
if len(rezultList)>0:
        self.Tool.AddMessage(u'         !!! {0} faulty records found ... '.
↪format(len(rezultList)))
```

GDPSyncroniserHelper

Data synchronization module (synchronizes data between tables)

class GDPSyncroniserHelper.**GDPSyncroniserHelper**(*gp*, *Tool=None*)

Bases: object

ESRI table synchronization class

AddMessage(*str*)

Wrapper for the 'AddMessage' procedure

Parameters

- **self** – The reserved object 'self'
- **str** – Output string

AddWarning(*str*)

Wrapper for the 'AddWarning' procedure

Parameters

- **self** – The reserved object 'self'
- **str** – Output string

DoSync(*definition*, *workspace=None*, *startEditing=False*, *startOperation=False*, *with_undo=False*, *multiuser=False*)

Data synchronization procedure

Parameters

- **self** – The reserved object 'self'
- **definition** – 'SyncDefinition' object which describes the synchronization parameters
- **workspace** – DB in which to start data editing
- **startEditing** – Start the edit session in the DB specified in 'workspace' parameter? (Default = False)
- **startOperation** – Start the edit operation in the DB specified in 'workspace' parameter? (Default = False)
- **with_undo** – Sets whether the undo and redo stacks are enabled or disabled for an edit session. (Default = False)
- **multiuser** – Sets whether a DB contains a nonversioned, or versioned dataset. (Default = False)

Returns

- output - Returns the report about the process execution
- outputErrors - Returns the error description
- errIDs - list of IDs that had an error
- SyncIDs - list of IDs that was syncronized

_GDPSyncroniserHelper__DoSyncFields(*inRow*, *outRow*)

Procedure performs the field synchronization

Parameters

- **self** – The reserved object 'self'

- **inRow** – Row to synchronize
- **outRow** – Row to which synchronize

Returns

- output - 0, If no changes were necessary; 1, If there were any changes
- outRow - Altered row

_GDPSyncroniserHelper__DoSyncRow(*inRow*, *outTable*, *outTableFields*, *outTableJoinField*, *messageString*, *idvalueseparator*, *createNew*)

Procedure performs row synchronization

Parameters

- **self** – The reserved object 'self'
- **inRow** – Row to synchronize
- **outTable** – Output table
- **outTableFields** – Output table fields
- **outTableJoinField** – Output table join field
- **messageString** – Output message formatting
- **createNew** – Create new record if needed

Returns

- output - 0, If no changes were necessary; 1, If there were any changes
- Error description (in case there were errors)

class GDPSyncroniserHelper.**SyncDefinition**

Bases: object

Synchronization definition description class

Examples

Synchronize data between two tables with internal parameter definition:

```
from GISPython import GDPSyncroniserHelper

ErrOutput = ''
sync = GDPSyncroniserHelper.GDPSyncroniserHelper(gp, Tool)
deff = GDPSyncroniserHelper.SyncDefinition()
bridges = callGP('MakeFeatureLayer_management', pj(Pr.ConnAuto, 'SDEOWNER.Roads',
↪'SDEOWNER.Bridge'), '#', 'BRIDGESUBTYPE = 1')

# Define input table parameters
deff.inTable = bridges
deff.inTableJoinField = 'OBJECTID' # join field
deff.inTableFields = ('OBJECTID', 'BRIDGENUMBER', 'BRIDGENAME', 'LVM_DISTRICT_CODE',
↪'MI_SPECIALIST', 'MIREGION', 'SHAPE@XY') # fields

# Define output table parameters
deff.outTable = pj(Pr.ConnAuto, 'SDEOWNER.Roads', 'SDEOWNER.BridgeInspection')
deff.outTableJoinField = 'BRIDGEOID' # join field
deff.outTableFields = ('BRIDGEOID', 'BRIDGENUMBER', 'BRIDGENAME', 'LVM_DISTRICT_CODE',
↪'MI_SPECIALIST', 'MIREGION', 'SHAPE@XY') # fields
```

(continues on next page)

(continued from previous page)

```
deff.createNew = True
deff.messageDefinition = u'Nr: {1} - {2}' # output mask 'inTableJoinField +␣
↪inTableFields' defines sync order

# End of synchronization
output, outputErrors = sync.DoSync(deff, Pr.ConnAuto, True, True, True, True)

# Error message in case there are any error
if not outputErrors == u"":
        ErrOutput += u'Found errors in synchronization process:\n' + outputErrors +
↪'\n\n';
```

GDPSyncroniserHelper2

Data synchronization module 2 (synchronizes data between tables)

Second version is ment for advanced scenarious, but opereates data inMemory so it,s not intended for large data sets.

class GDPSyncroniserHelper2.**GDPSyncroniserHelper2**(*gp*, *Tool=None*)

Bases: object

ESRI table synchronization class 2

Second version is ment for advanced scenarious, but opereates data inMemory so it,s not intended for large data sets

AddMessage(*str*)

Wrapper for the 'AddMessage' procedure

Parameters

- **self** – The reserved object 'self'
- **str** – Output string

AddWarning(*str*)

Wrapper for the 'AddWarning' procedure

Parameters

- **self** – The reserved object 'self'
- **str** – Output string

DoSync(*definition*, *workspace=None*, *startEditing=False*, *startOperation=False*, *with_undo=False*, *multiuser=False*)

Data synchronization procedure

Parameters

- **self** – The reserved object 'self'
- **definition** – 'SyncDefinition' object which describes the synchronization parameters
- **workspace** – DB in which to start data editing
- **startEditing** – Start the edit session in the DB specified in 'workspace' parameter? (Default = False)
- **startOperation** – Start the edit operation in the DB specified in 'workspace' parameter? (Default = False)

- **with_undo** – Sets whether the undo and redo stacks are enabled or disabled for an edit session. (Default = False)
- **multiuser** – Sets whether a DB contains a nonversioned, or versioned dataset. (Default = False)

Returns

- id: record id,
- syncResult: notInitialized, synchronized, inserted, error
- error: error mesage or '',
- updated: true if inserted or updated

Return type

- output - list of dictionary items containig

_GDPSyncroniserHelper2__SyncDestinationToSource()
Procedure performs syncronization of type Destination To Source

Parameters **self** – The reserved object 'self'

Returns

- id: record id,
- syncResult: notInitialized, synchronized, inserted, error
- error: error mesage or '',
- updated: true if inserted or updated

Return type

- output - list of dictionary items containig

_GDPSyncroniserHelper2__SyncSourceToDestination()
Procedure performs syncronization of type Source To Destination

Parameters **self** – The reserved object 'self'

Returns

- id: record id,
- syncResult: notInitialized, synchronized, inserted, error
- error: error mesage or '',
- updated: true if inserted or updated

Return type

- output - list of dictionary items containig

_GDPSyncroniserHelper2__SyncSourceToDestination_TableObject()
Procedure performs syncronization of type Source To Destination

Parameters **self** – The reserved object 'self'

Returns

- id: record id,
- syncResult: notInitialized, synchronized, inserted, error
- error: error mesage or '',

- updated: true if inserted or updated

Return type

- output - list of dictionary items containig

class GDPSyncroniserHelper2.**SyncDefinition2**
Bases: object

Synchronization definition description class

HasInTableQuery()
Returns whether input table has query

HasOutTableQuery()
Returns whether input table has query

SourceToDestination()
Returns whether sync mode is Source To Destination

class GDPSyncroniserHelper2.**SyncItem2**
Bases: object

class for storing sysnc data item

ClerRowInfo()
Clears savec row objects

DoSyncFields()
Procedure performs the field synchronization

Parameters

- **self** – The reserved object 'self'
- **inRow** – Row to synchronize
- **outRow** – Row to which synchronize

Returns

- output - 0, If no changes were necessary; 1, If there were any changes
- outRow - Altered row

GetRowStatussInfo()
Gets Dict of row statuss reprezenting objects

JsonParamsHelper

Module for Json parameter file procedures

class JsonParamsHelper.**JsonParams**(*Tool*, *ConfigFolder*, *ConfigFile*)
Bases: object

Json parameter reading support class

AppendValueByPath(*path*, *key*, *Value*, *valueIsStringJson=False*)

GetParams()
Get parameters from the parameter file

Parameters **self** – The reserved object 'self'

GetValueByPath(*path*)

UpdateValueByPath(*path*, *Value*, *valueIsStringJson=False*)

WriteParams(*sort_keys=True*)
Save parameters in the parameter file

Parameters **self** – The reserved object 'self'

Examples

Update attributes from JSON file:

```
from GISPython import JsonParamsHelper

# User defined data update function
def UpdateData(self, key, configData, origKey):
        """Executes attribute selection from configuration file and pass the␣
↪parameters to the attribute calculation tool

        Args:
                self: The reserved object 'self'
                key: Dictionary key, which corresponds to the 'rightsType' argument
                configData: Retrieved data from the configuration file
                origKey: Primary rights type
        """

def mainModule(self, rightsType = '#'):
        #Define variables
        rightsType = self.rightsType

        JsonParams = JsonParamsHelper.JsonParams(self.Tool, 'ConfigFolder',
↪'ConfigFile')
        configData = JsonParams.GetParams()

        # Call UpdateData function (user defined) with according rights
        if (rightsType.upper() == "ALL" or rightsType.upper() == "SOMETYPE"):
                for key in configData.keys():
                        self.UpdateData(key, configData, rightsType)
        else:
                self.UpdateData(rightsType, configData, rightsType)
```

MailHelper

Module for e-mail operations. Module contains functions for typical SMTP operations, and parameter processing from user parameter file.

Module for e-mail operations

class MailHelper.**GISPythonMailHelper**(*Pr*, *recipients*, *Subject*, *Text*)
Bases: *MailHelper.MailHelper*

MailHelper wrapper class, which acquires parameters from the GISPython parameter file

class MailHelper.**MailHelper**(*From*, *recipients*, *Subject*, *Text*)
Class for easing the SMTP operations

AttachFile(*FilePath*)
Procedure to add attachments

Parameters

- **self** – The reserved object 'self'
- **FilePath** – Path to the attachment

SendMessage(*Mailserver*, *port=None*, *user=None*, *password=None*, *useTLS=False*, *useSSL=False*)
Procedure for sending an e-mail

Parameters

- **self** – The reserved object 'self'
- **Mailserver** – Mailserver name
- **port** – Mailserver port number
- **user** – Username
- **password** – Password
- **useTLS** – Use TLS (Default = False)
- **useSSL** – Use SSL (Default = False)

Examples

Send e-mail using parameters from parameter file:

```
from  GISPython import MailHelper

MailHelper.GISPythonMailHelper(self.Pr, ['***@mail.com'], 'Subject', 'e-mail content')
```

This script depends on following parameters which needs to be configured in *SysGISParams.py* file:

- Mailserver - mail server name `Mailserver = 'mail.server.com'`
- MailserverPort - mail server port number `MailserverPort = 587`
- MailserverUseTLS - use TLS in mail server? `MailserverUseTLS = True`
- MailserverUseSSL - use SSL in mail server? `MailserverUseSSL = False`
- MailserverUser - user name `MailserverUser = 'UserName`
- MailserverPWD - user password `MailserverPWD = r'userPassword'`
- MailFromAdress - e-mail adress from which to send the e-mail `MailFromAdress = 'userAdress@mail.com'`

Send e-mail:

```
from  GISPython import MailHelper

mailSender = MailHelper.MailHelper('mail@from.com', ['***@mail.com'], 'Subject', u'e-
↪mail content') # Set up the e-mail for sending
mailSender.SendMessage('smtp.server.com', 587, 'username', 'password', useTLS=True) #␣
↪Send e-mail
```

MyError

Module contains class for storing an error object.

Error module

exception MyError.**MyError**(*strerror*)
Bases: exceptions.Exception

Class for storing an error object

Examples

Raise error in the tool output:

```
from geopythoncore import MyError

l = 12
linecount = 10
if linecount != l-2:
        raise MyError.MyError(u'Error in line count') # Add the error in the tool␣
↪output
```

PublisherHelper

Module for deployment operations.

Deployment publishing operations module

class PublisherHealper.**PublisherHealper**
Bases: object

Class for easing the Rar file operations

Deply(*config*)
Does the dployment

Parameters

- **self** – The reserved object 'self'
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__clear(*folder*, *config*)
Clears unnecesery files

Parameters

- **self** – The reserved object 'self'
- **folder** (*[string]*) – relative path to folder to be processed
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__create_backup(*config*)
Does the backup creation

Parameters

- **self** – The reserved object 'self'
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__do_copy_files_to_dest (*folder*, *files_to_copy*, *config*)
Finds files to be copyed

Parameters

- **self** – The reserved object 'self'
- **folder** (*[string]*) – relative path to folder to be processed
- **files_to_copy** (*[list]*) – path of files to be copyed
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__do_deploy (*folder*, *config*)
Does the backup creation

Parameters

- **self** – The reserved object 'self'
- **folder** (*[string]*) – relative path to folder to be processed
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__do_process_json (*config*)
Replace required values by sring replacement

Parameters

- **self** – The reserved object 'self'
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__do_process_xml (*config*)
Changes required values in config xml

Parameters

- **self** – The reserved object 'self'
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

_PublisherHealper__files_to_copy (*folder*, *config*)
Finds files to be copyed

Parameters

- **self** – The reserved object 'self'
- **folder** (*[string]*) – relative path to folder to be processed
- **config** (*[PublisherHealperConfig]*) – Configuration of deplyment

init()
Class initialization procedure

Parameters **self** – The reserved object 'self'

class PublisherHealper.**PublisherHealperConfig**
Class for setting up publisher Healper

backupFolder = ''

configFilesJson = []

configFilesXML = []

destinationDir = ''

doBackup = False

includeFolders = []

moduleName = ''

replacementMap = {}

sourceDir = ''

PublisherHealper.**_find_all_files**(*directory*)
Finds files in the directory

Parameters **dir** – The directory in which to look for the file

PublisherHealper.**_find_all_folders**(*directory*)
Finds files in the directory

Parameters

- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for

PublisherHealper.**_find_file**(*directory*, *ext*)
Finds files in the directory

Parameters

- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for

PublisherHealper.**_find_file_by_name**(*directory*, *file_name*)
Finds files in the directory

Parameters

- **Dir** – The directory in which to look for the file
- **fileName** – File name to search for

PublisherHealper.**_md5**(*filename*)
calculates file md5 cheksumm

Parameters **fname** (*[string]*) – File path

Returns hex digest

Return type [string]

PublisherHealper.**_now_for_file**()
returns date now formated for filename

Returns [date reprezentation as string]

Return type [string]

PublisherHealper.**_replace_in_file**(*path*, *replace_map*)
replaces values in files using replace_map

RarHelper

Rar file operations module

class RarHelper.**RarHelper**
Class for easing the Rar file operations

ExtractRarFile(*RarFileName*, *destPath*)
Rar file extraction procedure

Parameters

- **self** – The reserved object 'self'
- **RarFileName** – Extractable file path + name
- **destPath** – Destination path for extracted files

Examples

Extract the Rar file:

```
from GISPython import RarHelper

# File extraction procedure
RH = RarHelper.RarHelper()
rarFile = 'c:\\tmp\fileName.rar' # Rar file full path
workDir = 'c:\\tmp\someDirectory' # Directory in which to extract the Zip file
RH.ExtractRarFile(rarFile, workDir) # Extraction procedure
```

SFTPHelper

SFTP operations module

class SFTPHelper.**SFTPHelper**(*userName*, *password*, *host*, *port*, *pkey_file=None*)
Class for easing the SFTP operations

close()
Closes the connection, if it is active

download(*remote*, *local*)
Download the file, using SFTP

Parameters

- **self** – The reserved object 'self'
- **remote** – Downloadable file path on the server
- **local** – Local download path

upload(*local*, *remote*)
Upload the file, using SFTP

Parameters

- **self** – The reserved object 'self'
- **local** – Uploadable file local path
- **remote** – Uploadable file path on the server

Examples

Function which compresses files and sends to SFTP server:

```
from GISPython import SFTPHelper
from GISPython import ZipHelper

tmpFolder  = os.path.join(Pr.TmpFolder, 'DataFolder')
WorkDBName = 'DBName'
WorkDB = callGP('CreateFileGDB_management', tmpFolder, WorkDBName)
zipFileName = os.path.join(tmpFolder, WorkDBName + self.Tool.MyNowFile() + '.zip')
ZIP = ZipHelper.ZipHelper()
ZIP.CompressDir(WorkDB, zipFileName, ['lock']) # Compress directory contents

# Call parameters from the external parameter file
SFTP = SFTPHelper.SFTPHelper(Pr.SFTPUser, Pr.SFTPPwd, Pr.SFTPHost, Pr.SFTPPort)
SFTP.upload(zipFileName, os.path.basename(zipFileName)) # Upload file to SFTP
```

SimpleFileOps

*File and filesystem operations module. Module contains functions for typical file and filesystem operations, and locking control and processing. This module uses windows PowerShell to address file locking situations. For module with the same functionality without PowerShell script usage use *SimpleFileOpsSafe* module.*

File and filesystem operations module

class SimpleFileOps.**LockResults**(*processName*)
Bases: object

Class for saving the data processing results (for LockSubprocess)

GetStatusTxt()
Get status description

Parameters **self** – The reserved object 'self'

class SimpleFileOps.**LockSubprocess**(*Tool*, *Dir*, *processName*)
Bases: object

Class that provides directory locking control and processing

__enter__()
With statement opening procedure :param self: The reserved object 'self'

Returns Locked - The file is locked by another process; DoneBefore - Process is already done; NoDir - Directory not found; Running - Process is running;

Return type LockResults with status

__exit__(*type*, *value*, *traceback*)
With statement closing procedure :param self: The reserved object 'self'

readJson()
Get the data from the lock file :param self: The reserved object 'self'

writeJson()
Save parameters in the file :param self: The reserved object 'self'

class SimpleFileOps.**SimpleFileOps**(*_Tool*)
Bases: object

Class for easing typical file and filesystem operations

BackupFiles(*InDirName*, *OutDirName*, *D*)
File archiving automation procedure (Overriding)

Parameters

- **self** – The reserved object 'self'
- **InDirName** – Input directory
- **OutDirName** – Output directory
- **D** – How old files to archive (number of days)

BackupOneFile(*InFileName*, *OutDirName*)
Specific file archiving automation procedure

Parameters

- **self** – The reserved object 'self'
- **InFileName** – Input file
- **OutDirName** – Output directory

CheckCreateClearDir(*DirName*)
Automation procedure which creates directory, in case it doesn't exist and if it exists then clear this dir

Parameters

- **self** – The reserved object 'self'
- **DirName** – Output directory

CheckCreateDir(*OutDirName*)
Automation procedure which creates directory, in case it doesn't exist

Parameters

- **self** – The reserved object 'self'
- **OutDirName** – Output directory

ClearDir(*DirName*, *searchPatern='*'*)
Directory cleaning automation procedure

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be cleaned

CopareFileLists(*SourceFiles*, *DestFiles*)
Compare two file lists to determine changes

Parameters

- **SourceFiles** (*List*) – Sorce file path list
- **DestFiles** (*List*) – Destination file path list

Returns AbsentSource (List): Files in Source and not in Destination AbsentDestination (List): Files in Destination and not in Source

Return type Dictionary

CopyAllFilesInDir(*SourceDir*, *DestDir*, *Searchpattern='*'*, *Ignorepattern=None*)
Copy entire directory tree from one directory to another

Parameters

- **self** – The reserved object 'self'

- **SourceDir** – Source directory
- **DestDir** – Destination directory
- **Searchpattern** – Searching condition

DelClearDir(*DirName*, *searchPatern='*'*)
Delete non-empty directory

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be deleted

FindDirectory(*Dir*, *Searchpattern='*'*)
Find subdirectories in the given directory

Parameters

- **self** – The reserved object 'self'
- **Dir** – The directory in which to look for subdirectories
- **Searchpattern** – Searching condition

FindFile(*Dir*, *Ext*)
Finds files in the directory

Parameters

- **self** – The reserved object 'self'
- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for

FindFileByDate(*Dir*, *Ext='*'*, *Date=datetime.datetime(2020, 5, 12, 13, 35, 25, 557301)*, *Mode='New'*)
Find files in the given directory which are newer than the given date

Parameters

- **self** – The reserved object 'self'
- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for ('*' - search any file)
- **Date** – Find files newer than given date
- **Mode** – File searching modes: New - search newer files; Old - search older files (Default = New)

FindFileRecursive(*Dir*, *Ext*)
Find files by extension

Parameters

- **self** – The reserved object 'self'
- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for

FindNewestFile(*Dir*, *Ext='*'*)
Finds the newest file in the directory

Parameters

- **self** – The reserved object 'self'
- **Dir** – The directory in which to look for the file
- **Ext** – The extension to search for ('*' - search any file)

GetLog (*server*, *EventLogOutputDir*, *D*)
File archiving automation procedure (None overriding)

Parameters

- **self** – The reserved object 'self'
- **server** – Server which eventlog backup to create
- **EventLogOutputDir** – Event log output directory
- **D** – How old files to archive (number of days)

GetSafeName (*text*, *substituteChar='_'*, *aditionalScaryChars=''*, *aditionalSpaceChars=''*, *noDotsInName=False*)
Modifies the text for use in the filesystem

Parameters

- **self** – The reserved object 'self'
- **text** – Text string which will be transformed
- **substituteChar** – Character to substitute the whitespace
- **aditionalScaryChars** – Additional characters to eliminate
- **aditionalSpaceChars** – Additional characters to replace
- **noDotsInName** – Forbid dots in filename (except the file extension seperator) (Default = False)

Returns

- Modified text as string

GetfileNameWithDate (*file*)
Add a date at the end of the filename

Parameters

- **self** – The reserved object 'self'
- **file** – Full path to the file to add the date

delFileIfExists (*fileName*)
"Deletes file if file exists

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be cleaned

printFile (*filePath*)
Print file content to the screen

Parameters

- **self** – The reserved object 'self'
- **filePath** – File to print

Examples

Check for the directory, and clear its contents:

```
from  GISPython import SimpleFileOps

FO = SimpleFileOps.SimpleFileOps(self.Tool)
workDir = pj(Pr.TmpFolder, 'WorkingDirectory') # Set the working directory
FO.CheckCreateDir(workDir) # Check if directory exists, if not, create the directory
FO.ClearDir(workDir) # Clear directory contents
```

Find the newest file in the directory and create a backup:

```
from  GISPython import SimpleFileOps

FO = SimpleFileOps.SimpleFileOps(self.Tool)
workDir = pj(Pr.TmpFolder, 'WorkingDirectory') # Set the working directory
backupDir = pj(Pr.TmpFolder, 'BackupDirectory') # Set the working directory
newFile = FO.FindNewestFile(workDir, '*') # Find newest file with any extension
FO.BackupOneFile(newFile, backupDir)
```

SimpleFileOpsSafe

File and filesystem operations module. Module contains SimpleFileOpsSafe class, which contains functions for typical file and filesystem operations. Class inherits SimpleFileOps functionality, but does not relay on Windows Powershell scripts to address locked files.

File and filesystem operations module

class SimpleFileOpsSafe.**SimpleFileOpsSafe**(*_Tool*)

Bases: *SimpleFileOps.SimpleFileOps*

Class for easing the most typical file and filesystem operations

BackupFiles(*InDirName*, *OutDirName*, *D=0*, *Ext='*'*)

File archiving automation procedure

Parameters

- **self** – The reserved object 'self'
- **InDirName** – Input directory
- **OutDirName** – Output directory
- **D** – How old files to archive (number of days)
- **Ext** – The extension to search for ('*' - search any file)

BackupOneFile(*InFileName*, *OutDirName*)

Specific file archiving automation procedure

Parameters

- **self** – The reserved object 'self'
- **InFileName** – Input file
- **OutDirName** – Output directory

ClearDir(*DirName*, *searchPatern='*'*)

Directory cleaning automation procedure

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be cleaned

DelClearDir(*DirName*)
Delete non-empty directory

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be deleted

_clear_dir(*path_*, *pattern='*'*)
Clear directory contents

Parameters

- **self** – The reserved object 'self'
- **path** – Path to the directory
- **pattern** – Check if the file matches the given pattern (Default = '*'(Matches everything))

_force_remove_file_or_symlink(*path_*)
Force remove files and symlinks

Parameters

- **self** – The reserved object 'self'
- **path** – Path to the file/symlink

_is_regular_dir(*path_*)
Check if directory is regular directory

Parameters

- **self** – The reserved object 'self'
- **path** – Path to the directory

_remove_readonly(*fn*, *path_*, *excinfo*)
Remove read-only files and directories

Parameters

- **self** – The reserved object 'self'
- **fn** – Function for removing either a directory or a file
- **path** – Path to the directory/file

delFileIfExists(*fileName*)
"Deletes file if file exists

Parameters

- **self** – The reserved object 'self'
- **DirName** – The directory to be cleaned

TimerHelper

Timing module. Module contains functions for countdown procedures in a code block.

Timing module

class TimerHelper.**TimedSubprocess**(*Tool*, *txt*, *lvl=1*)
Class for providing a code block with timing capabilities

__enter__()
With statement opening procedure :param self: The reserved object 'self'

__exit__(*type*, *value*, *traceback*)
With statement closing procedure :param self: The reserved object 'self'

class TimerHelper.**TimerHelper**
Bases: object

Class for easing the countdown procedures

GetTime()
Get the elapsed time

Parameters **self** – The reserved object 'self'

Returns Output text as a string

GetTimeReset()
Reset the elapsed time

Parameters **self** – The reserved object 'self'

Returns Output text as a string

TimerReset()
Reset the countdown

Parameters **self** – The reserved object 'self'

Examples

Add a message to the tool output:

```
from  GISPython import TimerHelper

with TimerHelper.TimedSubprocess(self.Tool, u'some process'):
                # Your code
                self.Tool.AddMessage(u'some action')
```

Output:

```
------------------------
>>>> Begin some process - yyyy-mm-dd hh:mm:ss
------------------------
some action

------------------------
>>>> End some process - h:mm:ss.ssssss
------------------------
```

xmlParamsHelper

Module for XML parameter file procedures.

Module for xml parameter file procedures

class xmlParamsHealper.**XMLParams**(*Tool*, *ConfigFolder*, *ConfigFile*)
Bases: object

xml parameter reading support class

AppendValueByPath(*path*, *key*, *Value*, *attrib*, *index=0*, *isString=False*)

GetAtributeByPath(*path*, *atribute*)

GetParams()
Get parameters from the parameter file

Parameters **self** – The reserved object 'self'

GetValueByPath(*path*)

UpdateAtributeByPath(*path*, *atribute*, *Value*, *index=0*)

UpdateValueByPath(*path*, *Value*, *index=0*, *isString=False*)

WriteParams()
Save parameters in the parameter file

Parameters **self** – The reserved object 'self'

ZipHelper

Zip file operations module. Module contains functions for common Zip file operations.

Zip file operations module

class ZipHelper.**ZipHelper**
Class for easing the Zip file operations

CompressDir(*dirPath*, *zipFileName*, *excludeExt=[]*)
Zip all files in the directory

Parameters

- **self** – The reserved object 'self'
- **zipFileName** – New Zip file path + name
- **dirPath** – Directory which contains archivable files
- **excludeExt** – File extensions not to include in the archive

CompressFile(*filePath*, *zipFileName*)
Compress file

Parameters

- **self** – The reserved object 'self'
- **filePath** – Archivable file path + name
- **zipFileName** – New Zip file path + name

CompressFileList(*filePathList*, *zipFileName*)
Zip all files in the list

Parameters

- **self** – The reserved object 'self'
- **filePathList** – List of archivable file paths + names
- **zipFileName** – New Zip file path + name

ExtractZipFile(*zipFileName*, *destPath*)
Extracts the compressed file

Parameters

- **self** – The reserved object 'self'
- **zipFileName** – Extractable file full path + name
- **destPath** – Destination path in which to extract the files

Examples

Archiving procedure:

```
from  GISPython import ZipHelper

ZH = ZipHelper.ZipHelper()
workDir = 'c:\\tmp\someDirectory' # Directory to archive
zipFile = 'c:\\tmp\fileName' + self.Tool.MyNowFileSafe() + '.zip' # New zip file with␣
↪formatted date (simple example)
zipFile = 'c:\\tmp\fileName{0}.zip'.format(self.Tool.MyNowFileSafe()) # New zip file␣
↪with formatted date (good example)
ZH.CompressDir(workDir, zipFile)
```

Extraction procedure:

```
from  GISPython import ZipHelper

ZH = ZipHelper.ZipHelper()
workDir = 'c:\\tmp\someDirectory' # Directory in which to extract the Zip file
zipFile = 'c:\\tmp\fileName{0}.zip'.format(self.Tool.MyNowFileSafe()) # Zip file with␣
↪formatted date
ZH.ExtractZipFile(zipFile, workDir)
```

1.3 Changelog

1.3.1 v1.46.1 (2020.05.12)

- SFTPHelper added ability to autorise SFTP connection with RSA private key file

1.3.2 v1.45.3 (2020.05.05)

- bug fixes for AGServerHelperNTLM

1.3.3 v1.45.2 (2020.04.28)

- aditional functionality for PublisherHealper

1.3.4 v1.45.1 (2019.15.12)

- refactoring and aditional functionality for AGServerHelperNTLM. New functionality for working with permisions, roles and users
- refactoring and aditional functionality for PublisherHealper

1.3.5 v1.44.1 (2019.09.11)

- Mor functionality for PublisherHealper module and coresponding functionality for JsonParamsHelper and xmlParamsHealper modules to support automated soulution publishing. New functionality further develops capabilities of automatic configuration file changing in CI/CD solutions.

1.3.6 v1.43.3 (2019.08.16)

- MXDHelper module bug fixes

1.3.7 v1.43.2 (2019.07.26)

- PublisherHealper module bug fixes

1.3.8 v1.43.1 (2019.07.19)

- Some code cleanup and bug fixes
- PublisherHealper module has added functionality for updating Json parameter files

1.3.9 v1.42.1 (2019.05.24)

- Added GISPythonToolBase module for geoprocessing tool operations and automation mechanisms for different operations
- Added PublisherHealper module for deployment operations and xmlParamsHealper for XML parameter file procedures
- AGServerHelperNTLM - added method for ArcGIS dataset name retrieval
- GISPythonModule - additional functionality for tool licence level check
- JsonParamsHelper - additional functionality for value update
- SysGISTools - restructured, multiple methods migrated to new GISPythonToolBase module
- SysTools_unittest - added encoding parameters
- Bug fix for TimerHelper

1.3.10 v1.41.1 (2019.01.03)

- GDPSyncroniserHelper2 - added functionality for synchronising geodtabase tables with Python list objects

1.3.11 v1.40.2 (2018.11.24)

- AGServerHelperNTLM.py added function to get rights group names for AGS services
- Added capabilities for SimpleFileOps and SimpleFileOpsSafe to CheckCreateClearDir - check if dir exist, creates it and clears in one function
- Added aditional support for outputting unicode text in script output

1.3.12 v1.40.1 (2018.09.26)

- AGServerHelperNTLM.py added support for self generated SSL sites (Unverified SSL cases)
- Added capabilities for GDBHelper.py
- Bug fixes and added capabilities for GDPSyncroniserHelper2.py

1.3.13 v1.39.2 (2018.08.15)

- Added aditional possibilities for SimpleFileOps.py

1.3.14 v1.39.1 (2018.07.29)

- Added aditional possibilities for AGServerHelperNTLM.py

1.3.15 v1.38.1 (2018.07.20)

- Bug fixes for GDPSyncroniserHelper2.py
- Added parameter EnvironmentName. Used for Error Email generation, to indicate Environment in with error ocured.
- Added SetupDefaultEnvironment.py module for fast environment setup.

1.3.16 v1.37.2 (2018.03.29)

- Bug fixes for AGServerHelperNTLM

1.3.17 v1.37.1 (2018.03.29)

- Bug fixes for ZipHelper and GISTools10

1.3.18 v1.36.2 (2018.03.04)

- Added additional capabilities for GDBHelper

1.3.19 v1.36.1 (2018.02.25)

- Added additional capabilities in shell command running
- Bug fixes in SimpleFileOps file locking functionality
- Added possibility to store time in date time parametrs in Json parameter file

1.3.20 v1.35.2 (2018.01.13)

- **Additional functionality added to SysGISTools module:**
  - As there is many performance problems with *arcpy* geoprocessing script history logging, GISPython sets by default for *arcpy* not to log GP history. You can change this behavior by setting `SetLogHistory = True` in parameters file
  - GISPython Shell command launcher and SQL command launcher has added capabilities to hide passwords that they do not apear in output (logfiles and on screen)

1.3.21 v1.35.1 (2017.11.09)

- Added GDPSyncroniserHelper2 module
- **Changes and bug fixes in:**
  - AGServerHelperNTLM
  - SimpleFileOps
  - SysGISTools
  - GISPythonModule
  - GDPSyncroniserHelper
- Added functionality in GDBHelper

1.3.22 v1.34.3 (2017.07.10)

- Aditional functionality helping of reprinting MyError objects

1.3.23 v1.34.2 (2017.06.19)

- Bug fixes in unicode output management
- Added functionality in SimpleFileOps and SimpleFileOpsSafe

1.3.24 v1.34.1 (2017.06.09)

- Released documentation on Read The Docs
- Renamed AGSServerHelpaer module and class to AGSServerHelper
- Added AGSServerHelperNTLM module
- Added additional package dependencies (python-ntlm, patool)
- More module examples

- Modified README.md

1.3.25 v1.33.1 (2017.06.05)

- Initial release

1.4 Authors

LVM GEO is a collection of geospatial information technology (GIT) products and services provided by the Geospatial Information Technologies business unit of the JSC Latvia's State Forests (LVM). We have been developing GIT since 2009 to support our business operations, and we offer GIT products and services not only internally, but also to clients specializing in various industries. LVM GEO products range from a modular and multifunctional geospatial information technology platform with interfaces for companies and organizations to open tools for spatial data processing for any GIT user.

Website http://www.lvmgeo.lv/en/

E-mail lvmGEOGit@lvm.lv

CHAPTER 2

Package index

- genindex

Python Module Index

Index

Symbols

A

B

C

H

I

J

L

M

O

P

R

S

T

U

V

W

X

Z