OptimizeRasters

User Documentation

(2016/02/22)

Disclaimer

Applicable for OptimizeRasters Version 20160216 (see Header of OptimizeRasters.py)

Note: OptimizeRasters and the Associated GDAL library are currently provided as a Prototype and for testing only. The functionality has not been exhaustively tested and is not currently covered under ArcGIS Support. Questions or suggestions related to the running of these workflows should be addressed to the GitHub forum associated with the download or sent to ImageManagementWorkflows@esri.com

Contents

Introduction

OptimizeRasters is a tool that converts raster from one format to another, and can also be used to copy rasters and other data to and from cloud storage. During data conversion the output format is optimized so as to improve read performance. This is primarily achieved by ensuring that the data is internally tiled and includes appropriate pyramids. Options are also included to compress the imagery, so as to save storage space.

OptimizeRasters supports two output formats: TIF and MRF. TIF is a very popular format that can contain pixels in different layouts. TIF files can be optimized for access if the pixels are internally broken into tiles. Many TIF files are not tiled. For example the standard products DigitalGlobe and USGS are not

tiled, so it is often advantageous to run OptimizeRasters to copy the data from one set of directories to another while tiling the imagery to enable faster access. MRF (Meta Raster Format) is a format for the storage of rasters that was originally developed at NASA. Its simple structure based on tiles is optimized for access. MRF also includes additional compression options not supported in TIF. In many cases it is advantageous to optimize rasters by converting them to MRF.

One primary use of OptimizeRasters is to copy rasters into cloud storage such as Amazon S3 or Axure Blob Storage so as to provide lower cost storage, yet provide performant, scalable and elastic data access. OptimizeRasters includes special functionality to work with imagery already stored in cloud storage. It also has the capability to write the intermediate data on local fast disk during the conversion process if the input is from a slower storage.

There are many parameters that can be configured based on format and input and output location of the files. These configuration parameters are defined in configuration files so as to simplify the repeated calling of the command on different datasets.

So as to speed up the conversion and uploading of files parallel processing using multiple threads and processes is used by OptimizeRasters. During conversion and upload various checks are performed to help ensure that all files are correctly transferred. Optimize Rasters also includes extensive logging to provide a record of the conversion and upload. If there are unresolvable errors then a log of these files is maintained to enable simpler investigation to the issues and for the program to resume the conversion at a later time.

OptimizeRasters is implemented as open source Python code accessing the GDAL_translate and GDALaddo tools. By utilizing GDAL OptimizeRasters can read as input a large number of file formats. If the output format does not support the associated metadata then it is written to an .AUX.XML files along with the resulting output files.

Overview of MRF (MetaRasterFormat)

MRF exists as a format and a data access driver in ArcGIS. The MRF format is a raster format optimized for access, which typically breaks a raster into 3 files:

- A small XML file that contains metadata and properties of the raster including references to the other MRF files. This file represents the MRF.
- A data file consisting of the pixel data and any pyramid stored as tiles.
- An index file that is used by applications to quickly locate within the data file the required tiles needed to cover an area of interest at the appropriate scale.

Splitting a raster into these three files provides important advantages for cloud based access.

The MRF raster driver for GDAL in ArcGIS enables fast reading of the MRF formatted rasters, but also enables MRF to act as a caching proxy of other raster file formats. This allows MRF to speed up data access from the cloud or from slower storage. MRF is directly supported in ArcGIS 10.4. MRF is supported in ArcGIS 10.3.1 cumulative patch. Once the MRF driver is available, ArcGIS sees an MRF file as any another raster dataset.

The MRF driver can work in 4 modes:

StaticMRF – In this mode the MRF file acts in a similar way to a standard raster dataset e.g. geoTIF and the driver reads the tiles as required.

SplitMRF – Taking advantage of fact that MRF is stored in three separate files, it is possible to store the large data file on slower tiered storage, while keeping the metadata and index files on faster storage. This can speed up access by eliminating many requests to the slower storage devices.

ClonedMRF – In this mode the MRF is defines a local cache location for the storage of tiles. When the driver accesses a tile from the matching source MRF data file, it stores a copy of the tile on the local storage. Subsequent request for the same tile result in use of the local copy, speeding up access. The optimum way of scaling image access is to store imagery as MRF format on cloud storage and then use ClonedMRF to access it.

CachingMRF – This is similar to ClonedMRF, but the data source can be nearly any other raster format, stored in a local file or cloud storage. The rasters stored on the cloud storage will only be read when required and will be cached locally in MRF format. In this way it is possible to get ArcGIS to read a range of formats stored on cloud storage (or slow tiered storage) and have performance improved by MRF tile caching. The cache is typically stored in the LERC compressed format, which has a great speed/size ratio.

Note that the MRF driver works in both ArcGIS for Desktop and ArcGIS server, Windows or Linux. It is possible to directly read MRF files as rasters in ArcGIS for desktop with the data stored on Amazon S3 storage. Similarly one can create a Mosaic Dataset from large collections of MRF files. The MRF metadata files are stored locally to desktop or server, but can reference rasters stored on slower storage. As the MRF files are small, the data management process can be much faster. To serve the imagery the same MRF files can be transferred to a new server, which will then reference the same source data and create its own local copies as needed. For best performance the server should be well connected and located as close as possible to data store. In the Amazon AWS cloud this can be achieved by putting the server in the same region as the S3 storage. For some datasets it is also possible to embed the MRF XML files into the mosaic dataset removing the requirement for local files. The advanced process to do this requires the use of embedding the MRF XML into a feature service and using the Table Raster type.

Usage Patterns for OptimizeRasters

OptimizeRasters can be used in various ways to assist in Image Management and Sharing. The following are example workflows

Copying and optimizing standard raster products to enterprise storage

You may have a set of directories that contain data from a source, but access performance is not satisfactory. Typical examples would be data from most satellite imagery vendors that deliver imagery as TIF files that are not tiled and do not have pyramids, but do contain useful metadata as auxiliary or sidecar files. In this case, you can use OptimizeRasters to copy the all the data from one directory (possibly on an external hard disk) to a second device (e.g. your organizations shared file storage). OptimizeRasters will copy all files including metadata, but will also convert the TIF files into Tiled TIF

with internal pyramids. This conversion is lossless. The resulting file names will be the same, but the TIF files will be faster to access. Nearly all applications that access TIF files, can access Tiled TIF. Using OptimizeRasters in this way is similar to using the ArcGIS Copy Rasters command, but it ensures that all the data files and not only the rasters are copied.

A variation of the same workflow is to have the file format of the data converted during copying. For example the source data may be delivered in a flavor of JP2 that is slow to read. OptimizeRasters can convert the format to TIF which is faster to read (although it may be larger). GDAL and thus ArcGIS use the content of the first bytes of a file to identify the file format, while ignoring the extension. Therefor it is possible to convert the files while keeping the original file name. This is sometimes required when a product includes metadata files that reference the raster data by name. OptimizeRasters includes an option to enable custom extension naming.

Other variations of the same workflow could be to include a compression option during the conversion process so that the resulting data is compressed differently, reducing storage space. This compression can be lossless (e.g. using Deflate) or lossy (e.g. using JPEG). When converting to StaticMRF the compression can be also be LERC, a high speed compression which can be lossless or controlled lossy. LERC is especially valuable for higher bit depth data such as newer satellite imagery and elevation models.

Copying data to Cloud storage

Another common workflow is to use OptimizeRasters to copy raster data to cloud storage so that it can be accessed using elastic compute while reducing storage costs. By defining the destination (output directory) to be cloud storage (such as S3), OptimizeRasters can be used to move data to cloud storage. This transfer typically includes conversion of the raster format to MRF, but OptimizeRasters can also be used to transfer data with the output as TiledTIF.

Creating ClonedMRF or Caching MRF files

ArcGIS and many other applications inherently can only access rasters by referencing a file on a local or shared file system. This raises the issue on how to access imagery that is stored on cloud storage. The solution is to create a ClonedMRF or CachingMRF file that references the source on the cloud storage (or tiered file storage). OptimizeRasters can be used to transfer to a local (preferably fast direct access) storage a directory of rasters including auxiliary files that may contain product specific metadata, while replacing the data file with CachingMRF or CloningMRF. If the source data is a MRF file then a ClonedMRF file can be created on the local drive, otherwise a CachingMRF file is created. The resulting directory will contain all the same rasters as the source, but the size will be considerably smaller as the original raster files will be replaced by small MRF metadata files. The local MRFs will then be used to cache the remote raster data as it is being accessed. This is particularly valuable if it is known that not all remote data will be accessed. Since the large data files are not transferred initially, the time until the raster can be used is also much shorter.

Installing OptimizeRasters

Optional Packages

To upload to Amazon S3 and Microsoft Azure there are some third party packages for python that need to be installed. The easiest way to install these third party packages is by using a tool for installing python packages.

This tool is called PIP. If the version of python installed on your machine is version 2.7.9 and above pip is already installed.

With ArcGIS 10.3.1, the python version is 2.7.8. Therefore pip would need to be installed.

Installing PIP.

1. Open this link in a browser. https://pip.pypa.io/en/latest/installing/
2. Under the heading Installing with get-pip.py right click and download the file get-pip.py
3. run the command python get-pip.py.

    example c:\PYTHON27\ArcGIS10.3\python.exe c:\temp\download\get-pip.py

Installing boto (To use Amazon S3).

1. Open a command line window. (Start/Run/Cmd)
2. Browse to the Scripts folder within the python folder.

    example cd c:\PYTHON27\ArcGIS10.3\Scripts

3. Type in the line pip install boto

Installing Azure.

Azure python module is required to read/write to (Microsoft Azure block blob) cloud file system.

1. Open a command line window. (Start/Run/Cmd)
2. If pip is not installed, install pip first.
3. Browse to the Scripts folder within the python folder.

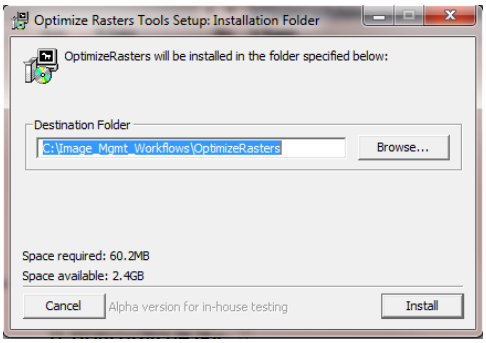
    example cd c:\PYTHON27\ArcGIS10.3\Scripts

4. Type in pip install azure
5. Browse to the link : https://pypi.python.org/pypi/azure/1.0.3

GUI Installation

It's recommended to use the setup EXE (OptimizeRastersToolsSetup.exe)

Step 1: Run the OptimizeRastersToolsSetup.exe.



Step 2: Click on Browse to change the installation folder. (optional)

Step 3: Click Install to begin the installation.

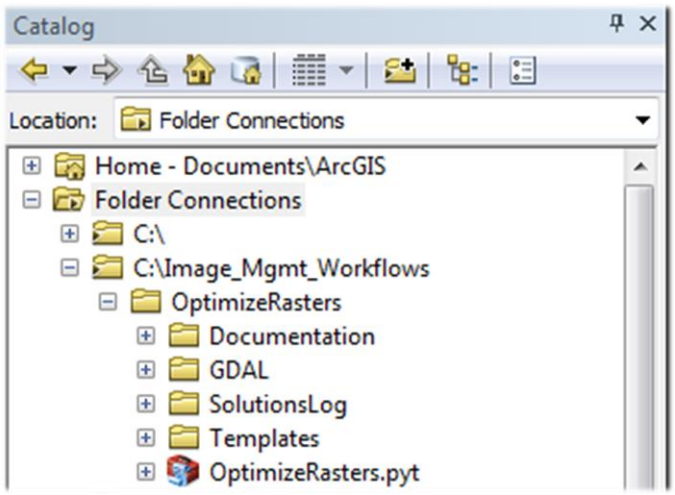
Step 4: Click Close to dismiss the installation dialog.

Accessing the toolbox.

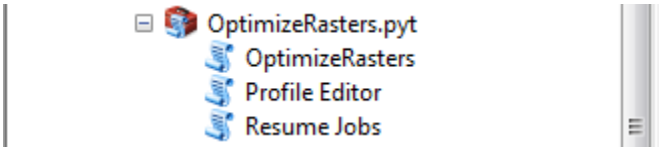
Once the installation is complete the OptimizeRasters toolbox can be accessed using ArcGIS Desktop. Please note the installation location before starting ArcMap.

Step 1: Start ArcMap.

Step 2: In the catalog window, browse to the installation folder (default: c:\Image_Mgmt_Workflows\OptimizeRasters)



Step 4: Open the tool box by clicking + sign next to OptimizeRasters.pyt



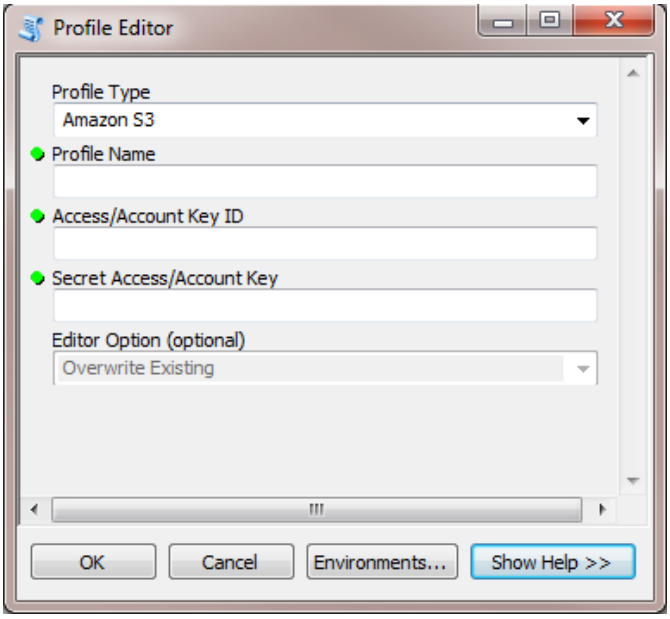
Using OptimizeRasters (GP ToolBox)

The OptimizeRasters GP Toolbox requires ArcGIS Desktop 10.3.1x and above. However if ArcGIS Desktop is not installed OptimizeRasters can be used as command line tool. Check Command Line Usage for more information.

The Toolbox consist of three tools. Each tool is briefly described below:

Profile Editor

Profile editor allows the user to Edit / Store the credential profiles for use with the preferred cloud storage solution to be used for uploading data. There are two types of cloud storage solutions supported, Amazon S3 and Microsoft Azure.



Adding a New Profile

To add a new profile follow these steps..

Step 1: Select a Profile Type

Step 2: Enter the Profile Name, Access Key and Secret Key.

Step 3: Click ok to Save.

Editing / Deleting an Existing Profile

To Edit or Delete an existing profile follow these steps.

Step 1: Select a Profile Type.

Step 2: Enter the Profile Name to Edit or Delete.

Step 2a: To edit a profile enter the new Access Key and Secret Key and choose Overwrite Existing in step 3.

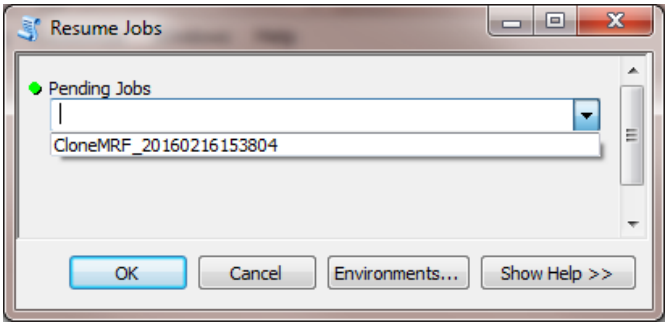
Step 2b: To delete a profile enter ONLY the name of the profile and select Delete Existing.

Step 3: Select an action in the Editor Option and Click ok to run.

NOTE: For security reasons, the Access Key and Secret key are not displayed for existing profiles.

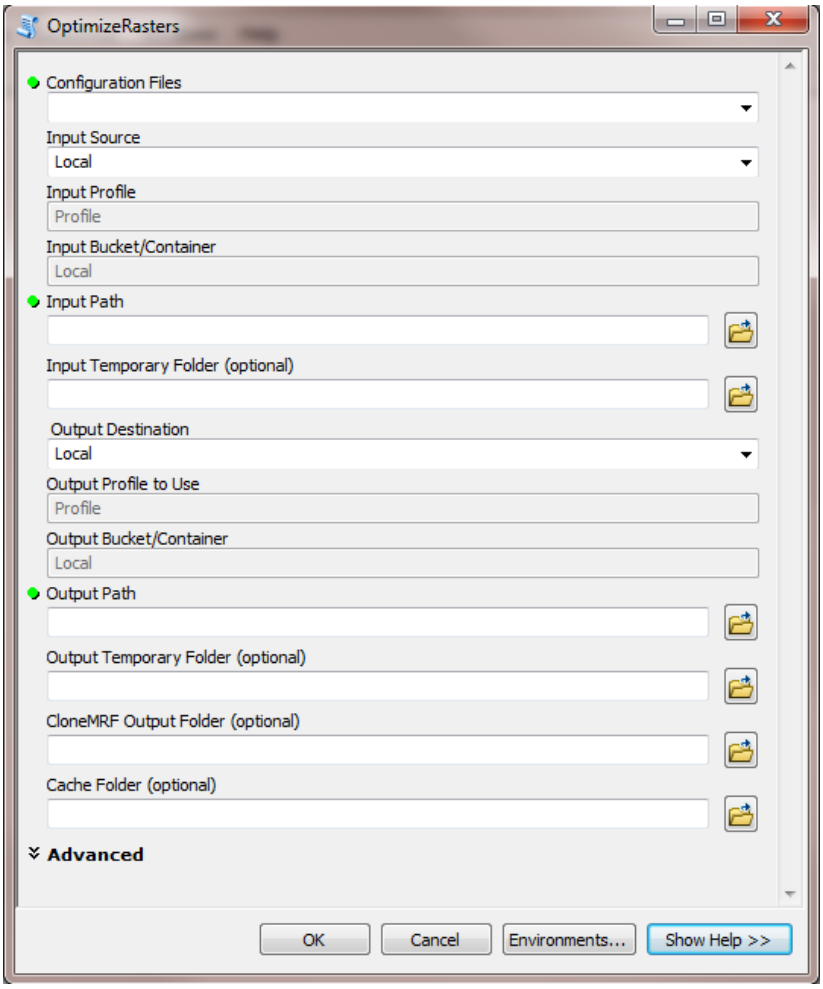
Resume Jobs

Processes that failed to run to succession using the OptimizeRasters tool will show up as a list of pending jobs. Select a job and click OK. If a job completes in its entirety it will no longer show up here.



OptimizeRasters

Once you have setup the appropriate profiles you can now use the OptimizeRasters dialog to perform jobs. Below is a brief explanation on the various entries required to run a process.



Configuration Files: OptimizeRasters comes with a set of preset templates that can be used to drive the conversion process of your data. Use the pull down list to select a template file that suits your needs.

Input Source: Input source defines the source of the input files. The input files can come from a local disk or a cloud source.

Input Profile: This option is active when the Input Source points to a cloud based solution like S3 or Azure.

Input Bucket / Container: A bucket or container is the unique storage location for your files. Refer to S3 or Azure documentation for more information.

Input Path: The Input path can either be a local folder or a cloud storage folder name. For local storage you can use the folder browse button to pick a location. For Cloud storage solutions Type the location here.

Input Temporary Folder: The temporary input location is used to process files while downloading from cloud storage before it is written to the output location. The temporary location while optional, can be useful if the space is restricted in the default location, which is the system temp location.

Output Destination: Output destination defines where the result of OptimizeRasters should be. It can can be a local disk or a cloud source.

Output Profile to Use: This is necessary when the Output Destination is either Amazon S3 or Azure. This passed as a separate entry to OptimizeRasters so it needs to be specified even if it is the same as input.

Output Bucket/Container : Similar to Input Bucket/Container.

Output Path: Similar to Input Path.

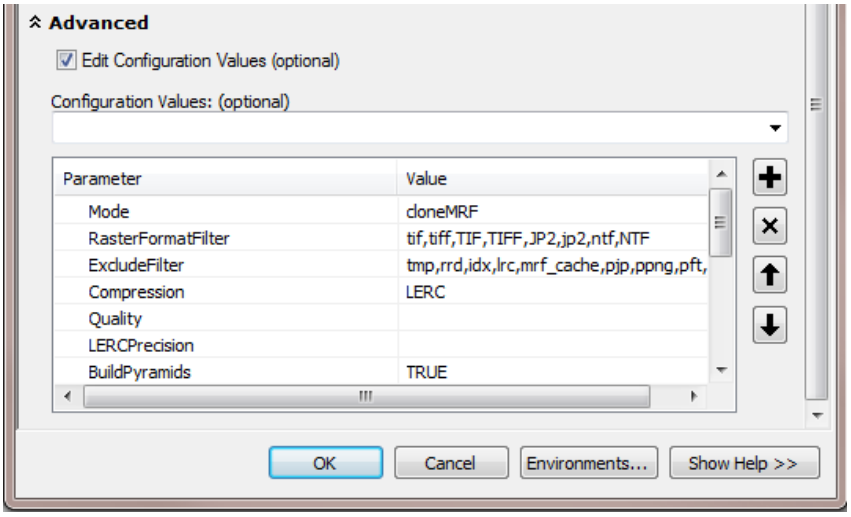
Output Temporary Folder: Similar to Input Temp Location.

Clone MRF Output Folder: Clone MRF files are local file pointers to the converted file that are stored either on the cloud or a local network. This option is disabled when the Mode is CloneMRF or CachingMRF

Cache Folder: The location where the cache files should be created.If left blank the default location is used. If there are any space constraints at the default location change this path to a different location where there is sufficient space to create cache

Advanced Section.

The Advanced section is for users who understand the parameters being used in OptimizeRasters and wish to change it. Based on the selected template the values are read and added to the table (see diagram below).



To edit the values select the check box labeled Edit Configuration Values. Edit the values, refer to the tool help for list of allowed entries for each parameter.

The edit configuration values are saved to a new file and used when the user click OK. It will have the time stamp appended to the original file name and displayed towards the end of the list of templates.

NOTE: If a user edited template is selected for further edits, the changes are saved back to the same file.

Using OptimizeRasters (Command Line)

Optimize Rasters can be called as acommand line tool.

Help for the command can be obtained by using the following command:

<path_to_python.exe> <path_to_optimizerastes.py> --help

A common command line usage is:

<path to python.exe> <path to optimizerasters.py> -input=<path to input folder> -output=<path to outputfolder> -mode=mrf

Command line parameters and switches

The following command line arguments can be used:

-input = Input directory path. Note all files in a directory will be processed.

-output = Output directory path

-config = Configuration file with default settings. All parameters not defined on the command line will be taken from this file. If undefined then OptimizeRasters.xml stored in the same location as OptimizeRasters.py will be used. The configuration file can be used to define often used configuration setting so reducing the command line parameters required. Parameter defined in the command line override configuration parameters.

-mode = Processing mode/output format. Typical values are tif, mrf, cachingmrf, clonemrf; refer below to the meaning of modes.

-cache = cache output directory path. Location to embed into MRF files for where to create the cache files. If not defined then will be the same location as the MRF files. (see below for information on cache management)

-quality =JPEG quality if compression is jpeg

-prec = LERC precision

-pyramids =If to generate pyramids - value should be true/false. Typically pyramids should always be created. Used to override the BuildPyramids parameter in config file

-subs = Include sub-directories - value should be true/false. If true all sub directories of the input directory will also be processed

-tempinput=Path to copy -input rasters before conversion. If the input is S3 then this is a required parameter. The input rasters will be temporarily copied to this directory before processing.

-tempoutput=Path to copy converted rasters before moving to output. If the output is S3 then this is a required parameter. The input rasters will be temporarily copied to this directory before being uploaded to S3. It is optimum if this is on a faster drive as pyramids will be generated by reading and writing to this location.

-clouddownload=Is the input data on a cloud storage? This flag lets the program know that the input location is a cloud storge or not. Enter true/false.

-cloudupload=Is the Output data destination on a cloud storage? This flag lets the program know that the output location is on a cloud storge or not. Enter true/false.

-cloduploadtype=Linked to –cloudupload. Specify if true. Valide values are 'amazon', 'azure'.

-clouddownloadtype= Linked to – clouddownload. Specify if true. Valide values are 'amazon', 'azure'.

-inputprofile= Input cloud profile name with for the input cloud storage. The profile name will be used to pick up the relevant credentials.

-outputprofile= Output cloud profile name for the output cloud storage. The profile name will be used to pick up the relevant credentials.

-inputbucket= Input cloud bucket/container name.

-outputbucket= Output cloud bucket/container name.

-clonepath= Path to auto-generate cloneMRF files during the conversion process

-job=Point to a pending job file name to resume.

Configuration File paramters.

All the above arguments can be defined in the configuration file, reducing the need for command line parameters. The configuration file contains additional parameters, with explanations of their use.

Some of the important configuration parameters the may need to be change include:

RasterFormatFilter – Defined the file extension of the rasters that should be converted. Typically this is set to “tif,TIF,mrf,tiff” and defines that files with these extensions are rasters that should be converted. All other files (e.g. JPEG file that may contain a Logo) do not get converted, but get copied from source to destination. Note the file extensions are case sensitive hence the definition of tif and TIF.

ExcludeFilter – Defines the extension of files that should not be copied as they are not needed. Typically if data is being converted say to TIF with internal pyramids or MRF files then existing ovr or rrd files are not required. Similarly there are other files that would typically be replaced with new files such as idx,

lrc, mrf_cache, pjp, ppng, pft, pzp. In some workflows other file extension from the source are also not required such as tfw aux.xml and can in included in this list.

KeepExtension – Defines if the output raster extensions should be changed or not. If true the extension will remain the same as the input even if the format was changed (eg to MRF). If false then the extension will be that most appropriate for the output format (typically .tif or .mrf)

PyramidFactor – Defines the factors to build. Typically this is 2 4 8 16 32 64. It can be set to a factor of 3 such as 3 9 27 81

PyramidSampling – Defines sampling to use for pyramids such as average or nearest. If undefined then average will be used except for some specific cases such as quality files for Landsat8

Interleave – Defines the interleave setting for use when compressing using JPEG. Can be Band or Pixel. If output is 3band and Interleave is Pixel then better compression is achieved.

NoDataValue – Defines if there is a NoDataValue in the source. This is important to define if rasters have 0 as NoData so that the pyramid generation will not include this value.

BlockSize – Define the size of the tiles (sometimes called blocks) in the output image. A value of 512 is recommended for most datasets. Smaller values that are powers of 2 (e.g. 128,256) may be used in dataset where the generation of temporal profiles is common.

Threads – Defines the number of simultaneous threads to use for parallel processing. This is typically default to 10

LogPath – Defines the location for LogFiles. By default these are created in the Logs directory in the same location as OptimizeRasters.py. The log file must be a UNC type location and cannot be on S3. The Log files are XML files and can be parsed as required.

Default Configuration Files

OptimizeRasters comes with a list of default configuration template files that can be used out of the box. Below is a list of configuration files and a brief description of each. The values used for conversion can be viewed in the advanced section of the OptimizeRasters tool, or via text editor. The files can be found in the OptimizeRasters\Templates or UserTemplates folder.

Airbus_SatelliteProduct_to_MRF_LERC : To aid in conversion of Airbus Satellite Product data to MRF using LERC compression.

CachingMRF : Converts raster files to a Caching MRF file.

CloneMRF : Creates clone MRF files from raster files.

DG_SatelliteProduct_to_MRF_LERC : To be used on Digital Globe Satellite imagery. Converts imagery to MRF using LERC compression.

Imagery_to_MRF_JPEG : Converts Imagery to MRF using JPEG compression.

Imagery_to_MRF_LERC : Converts Imagery to MRF using JPEG compression.

Imagery_to_TIF_JPEG : Converts Imagery to TIFF using JPEG compression.

Imagery_to_TIF_LZW : Converts Imagery to TIF using LZW compression.

Landat_to_MRF_LERC : Converts Landsat Imagery to MRF using LERC compression.

Overviews_to_MRF_LERC : Converts Overview Imagery to MRF using LERC compression.

Sentinel2_to_MRF : Converts Sentinal2 Imagery to MRF using LERC compression.

Resume interrupted workflows

OptimizeRasters has a built in resume functionality to help continue a workflow from a point of interruption. This is immensely helpful in cases where only a few of the input files have failed to process and to avoid redoing the entire workflow each time something goes wrong. OptimizeRasters by default takes care of processing only those files that have failed and there's no extra effort is necessary to get the workflow restarted to complete only the failed ones. This therefore increases the productivity by freeing the user to work with something else.

By default OptimizeRasters creates a (Job) file each time it's run. A Job file is a simple text file which has a header block at the start, a series of key value pairs each starting with '#' and followed by a list of files to process at the source input location. The naming convention for the (Job) file is OR_YYYYMMDDTHHMMSSssssss.orjob. The syntax followed by the underscore is the DATATIME format in the ISO format and denotes the date and time when the workflow was first run. However, it's possible to name the job files using the command-line flag –job along with other parameter that are required to process a workflow. The default .orjob extension will be added if omitted.

The resume functionality or the creation of the Job file by OptimizeRasters was designed to work in silence and behind the scene hence shouldn't require any attention by the user. On completion of processing a workflow successfully, the Job file will be moved to the Job folder for archive purpose or is left behind at the OptimizeRasters.py location in case of any errors. Users can resume job files that are left behind by using the –input command line flag pointing at the Job file.

E.g. python OptionizeRasters.py –input=OR_20151211T024329630000.orjob

To inspect any errors however, the user may manually inspect the Job file to see the list of files that have reported errors or the processing stage at which each file has failed.

Working with S3

AWS standards to manage S3 credentials

OptimizeRasters supports the AWS standards to manage credentials. This means credentials can use the default environment variables (AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY_ID) or by using the default AWS credential file located at

%USERPROFILE%\.aws\credentials.

Please note credentials are text/INI file without an extension.

The credentials for AWS also can be set using the amazon Command line tools, if you are using EC2 instance these tools are setup already on the system. You can use the following commands to setup the credentials

$ aws configure

More help on this can be found

http://docs.aws.amazon.com/cli/latest/reference/configure/index.html

If the tools are not installed on the system they can be installed from this location

http://aws.amazon.com/cli

The primary advantage of using an AWS credential file is to store S3 keys for the default user using the access security offered by the OS. The credentials file will only be accessible to the user who already has the read/write access to the profile location. This will ensure that when copying the OptimizeRasters package onto another machine, that credentials otherwise recoded in the OptimizeRasters config file are not inadvertently exposed.

Reading from public AWS buckets

The following configuration keys need to be left empty to enforce reading from a public AWS bucket.

<In_S3_AWS_ProfileName></In_S3_AWS_ProfileName>
<In_S3_ID></In_S3_ID>
<In_S3_Secret></In_S3_Secret>

Overriding AWS credentials

To bypass the default AWS credentials, the OptimizeRasters config file can be edited to include the necessary information as shown below to have the credentials added to S3 input and S3 output storage respectively.

<In_S3_ ID > _IN_S3_ID_</In_S3_ ID >
<In_S3_Secret>_IN_S3_SECRET_KEY_</In_S3_Secret>

<Out_S3_ ID > _Out_S3_ID_</Out_S3_ ID >
<Out_S3_Secret>_Out_S3_SECRET_KEY_</Out_S3_Secret>

The keys in the config file take the precedence over the AWS standard credential manager.

AWS credentials usage

If using the AWS credential manager for S3 bucket authentication, the entries related to AWS profiles need to be updated in the OptimizeRasters parameter file to reference the matching profile names in the AWS credentials file.

For e.g. In the OptimizeRasters config file you can have the AWS manager specific entries updated to match the AWS profile names,

<In_S3_AWS_ProfileName>OptimizeRaster_S3In</In_S3_AWS_ProfileName>
<Out_S3_AWS_ProfileName>OptimizeRaster_S3Out</Out_S3_AWS_ProfileName>

For the above entries to work, AWS profile file (credentials) has to be updated to reflect the profile names in the parameter file as shown below.

[OptimizeRaster_S3In]
aws_access_key_id=XXX_YOUR_ACCESS_KEY_ID_XXX
aws_secret_access_key = XXX_SECRET_ACCESS_KEY_XXX

[OptimizeRaster_S3Out]
aws_access_key_id = XXX_YOUR_ACCESS_KEY_ID_XXX

aws_secret_access_key = XXX_SECRET_ACCESS_KEY_XXX

Setting parameters to Read and Write from S3

Various parameters need to be set in the configuration file to enable data to be uploaded to an S3 bucket. For example to upload data to the following http://mydata.s3.amazonaws.com/abc/pqr/t then

The following additional changes must be made to the config file:

Out_S3_Upload – Need to be defined as True

Out_S3_Bucket - Specify S3 bucket name where the data should be upload e.g mydata

Out_S3_ParentFolder - Specify the s3 folder where the data is to be uploaded to. IE You need to exclude the bucket name. The outputfolder path in the above example would be abc/pqr/t

Out_S3_ID – Define the access ID which will be used to make a connection

Out_S3_Secret – Define the scretkey required to access the bucket

Out_S3_DeleteAfterUpload – Should be defined as true, as the process will create intermediate data files that should be deleted after uploading.

The command line would then be as follows:

<path to python.exe> <path to optimizerasters.py> -input=<path to input folder> -output=<path to s3 outputfolder> -tempoutput=<path_to-a_folder_on_localdisk> -s3output=true –mode=mrf

Caution on Case Sensitivity

Note that file names in S3 bucket are case sensitive. It is therefore very important not to change the case of the file names or their extensions when copying data to S3. References to these files must also maintain the case sensitivity.

Setting Access Control on S3

Access to data on S3 buckets is defined using the Amazon S3 Access Control Lists (ACLs). Details on the ACLs can be found on http://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#CannedACL

By default OptimizeRasters will use the *public-read* ACL that enables all users to read the data. The ACL to be used can be defined in the config file by using the following node.

Out_S3_ACL – Defines the canned ACL to apply to uploaded files. Acceptable values are:

private, public-read, public-read-write, authenticated-read, bucket-owner-read, bucket-owner-full-control

If don't want the data to be made public, you need to set the ACL as private and apply Sa bucket policy. The bucket policy can be set up to allow the data to be accessed from the machine where server or desktop is running. The bucket policy can be applied such that only that specific machine IP has an access to it. For more information on this you can refer to the documentation from amazon

http://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html

http://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies-vpc-endpoint.html

Working with Azure

Similar to working with S3, the values of few configuration parameters in the parameter file must be set to prepare OptimizeRasters to work with the Azure FS from Microsoft. Please note on comments in Italic.

<Out_Cloud_Type>azure</Out_Cloud_Type>

Cloud type, valid values for now are [azure, amazon]. if empty no default, must be set in the configuration parameter file.

<Out_Azure_ParentFolder>folder/output</Out_Azure_ParentFolder>

Root destination folder where files must be uploaded.

<Out_Azure_AccountName></Out_Azure_AccountName>

Account name. This is similar to Account ID for S3

<Out_Azure_AccountKey></Out_Azure_AccountKey>

Account key. Similar to the secret key for S3

<Out_Azure_Container>tiffs</Out_Azure_Container>

Container name. Similar to Bucket name for S3

<Out_Azure_Access>blob</Out_Azure_Access>

Access type. Similar to <Out_S3_ACL> for S3.

Value values are [private, blob, container]

private: accessible only to the user

blob: Files within the container are publicly accessible.

container: Same as type blob plus container metadata are publicly accessible.

Azure command line usage

python OptimizeRaster.py -input=c:/date/rasters -cloduploadtype=azure -cloudupload=true -tempoutput=c:/temp/tempoutput

python OptimizeRaster.py -input=c:/date/rasters -cloduploadtype=amazon -cloudupload=true -tempoutput=c:/temp/tempoutput

Internal Info

Parallel upload support: Yes. Similar to Amazon/S3 uploads. The upload file size does not matter.

No of parallel upload threads per file: 20

Azure blob type: Block blob

Upload payload size per thread: 4 MB.

NOTE: Please note, currently only uploading to Azure FS is supported.

Working with Cloned MRF from S3

If the rasters have been stored as MRF formatted rasters in S3 then it is recommend to use clonedMRF mode to access them. In the 'ClonedMRF' mode, a copy of the input directory is created including all auxiliary files, but excluding MRF data and index files. The MRF files are copied and modified to include appropriate links back to the original data and index files and appropriate cache files locations are defined.

For example if the input MRF files are in http://mydata.s3.amazonaws.com/abc/pqr/t

Following configuration changes are needed:

In_S3_Bucket - Name of the input bucket mydata

In_S3_ParentFolder - The s3 folder where the data needs to be downloaded. abc/pqr/t

In_S3_ID – Define the access ID which will be used to make a connection

In_S3_Secret - Required secretkey

The command line would then be as follows:

<path to python.exe> <path to optimizerasters.py> -input=<path to s3 input folder> -output=<path to outputfolder> -s3input=true –mode=clonemrf

Using Caching MRF with rasters stored in S3

If the rasters in a non MRF format (e.g. geoTIF, jpeg, jp200) on S3, then the following can be used to access the data within the arcgis framework using caching MRF. In the 'CachingMRF' mode, a copy of the input directory is created excluding all raster files such as TIF and JP2. These are substituted with CachingMRF files. CachingMRF files point back to the source data and appropriate cache files are defined. This has the advantage of not duplicating data but providing faster access and having requests cached to local machines. Prior to using CachingMRF one must ensure that pyramids exist on the source data. If the source rasters are large (>5000cols) and the data not tiled then there can be a considerable performance degradation.

The same changes to the configuration file as for cloned MRF are required.

The command line would then be as follows:

<path to python.exe> <path to optimizerasters.py> -input=<path to s3 input folder> -output=<path to outputfolder> -s3input=true –mode=cachingmrf

Using Landsat 8 data provided by Amazon to create caching MRF files

Amazon provides free access to Landsat 8 data as part of their PDS (Public Dataset) program. (see http://aws.amazon.com/public-data-sets/landsat) These datasets are in Tiled geoTIFF format, with pyramids build at a factor of 3. Using CahingMRF it is possible to directly use them in within ArcGIS. To access a scene the path row and scene-id are required. A configuration file (OptimizeRasters_PDScaching.xml) with all required parameters is provided.

Following is a sample command.

c:\Python27\ArcGIS10.3\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters\OptimizeRasters.py -config=c:\Image_Mgmt_Workflows\OptimizeRasters**OptimizeRasters_PDScaching.xml** -input=L8/160/043/LC81600432015109LGN00 -s3input=true -output=c:\temp\landsatpdsdata\L8\160\043\LC81600432015109LGN00

Creating CachingMRF files from raster's stored on a network attached storage

CachingMRF can be used to speed up access of data to slower network attached storage, by creating CachingMRF files of the source data on a fast local (or SSD based) drives.

The following command line can be used:

<path to python.exe> <path to optimizerasters.py> -input=<path to source data> -output=<path to fast disk> -mode=cachingmrf

Using Split MRF with raster's stored on a network attached storage

MRF files can be used also without the caching option, while still optimizing access. This is typical used for by users wishing to optimize access or imagery stored on NAS (Network Attached Storage), but not wanting to include caching. When ArcGIS access a raster it also accesses the associated metadata files. One way of optimizing the access it to ensure the metadata files and SplitMRF files are copied to faster local storage, while leaving the large data files on the NAS. In the 'SplitMRF' mode, a copy of the input directory is created, but it excludes the MRF data files and instead adds links in the MRF files to point back to them.

The following command line can be used:

<path to python.exe> <path to optimizerasters.py> -input=<path to s3 input folder> -output=<path to outputfolder> -mode=splitmrf

Cache Management

When using ClonedMRF or CachingMRF the MRF driver creates caches of the rasters tiles and associated indices. The MRF driver will only add to the cache, but not delete the cache for example if the disk gets full. These cache files can grow to become very large and care should be taken to ensure they are correctly managed and do not result in disk being full which would lead to errors.

By default the caches will be saved with the extension .mrf_cache in the same directory as the MRF files. It is advantageous to define separate location for the location of the cache so that the files can be easily deleted. The MRF driver has been designed to be robust to the deletion of these cache files. If a cache is found to be missing the driver will fetch new tiles and start recreating the cache. The location to store the cache is defined using the –cache option in the command line. This enables one to specify a separate location for the cache files that can be easily cleaned up. Typically one does not want all the cache files in a single directory. As the cache files have the same name as the source raster, it is possible for errors to occur if two files have the same name. It is therefore recommended to have cache follow a similar directory naming as the source data. The MRF driver will attempt to create the directory structure specified for the location of the MRF files, so the provided directory structure need not exist in advance,

but the cache must be writable else blank imagery will be returned. Best practice is to define a drive or directory specifically for the cache. It is then easy to delete the files in this directory and subdirectories when additional space is required.

CleanMRF

The CleanMRFCache.py tool is a simple cache clean up tool that clears the cache based on amount of memory required.

Usage:- <path to python.exe> <path to CleanMRFCache.py> -input=<path_to_rootdirectory> -ext=<extension of files to be deleted> -size=< Size in Bytes that should remain on the disk>

e. c:\Python27\ArcGIS10.3\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache -ext=txt,mrf_cache -size=1

c:\Python27\ArcGIS10.3\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache

The following are optional arguments:

-mode : Can be set to del or scan.

-ext : Extension of files to delete. By default this is mrf_cache

-size: Size in Gigabytes that should remain on the disk. Default is 2 (2GB)

The 'del' mode will delete the files found until the target free -size has been achieve. 'Scan'/any other will just display the files found and their sizes without deleting. This can be used to demonstrate how much can be freed up if files get deleted/for info purpose. The results are displayed least/oldest accessed files first along with the size and access time (epoch) time of each.

Note: The program should be scheduled to run using the windows task scheduler at intervals requested.

Special Considerations of ArcGIS Server

The MRF driver has been designed to enable multiple processes to write the same cache simultaneously without corrupting the cache.

When using ArcGIS for Desktop and Server on the same machine some special considerations are required. By default if ArcGIS Server creates a file then ArcGIS for desktop which is logged in as a different users is not able to read the file. This can cause corrupting in that some files cannot be read.

To resolve this problem it is necessary to allow all everyone with read and write permission to the folder where the cache will be stored.

---------- End of Document ----------