OptimizeRasters

User Documentation | July 6, 2021

esri | THE SCIENCE OF WHERE

This document applies to OptimizeRasters Version 20180705 (see the OptimizeRasters.py header). 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. Please address questions or suggestions related to this workflow to the OptimizeRasters GitHub forum or to ImageManagementWorkflows@esri.com.

Contents

Overview

What is OptimizeRasters?

Use OptimizeRasters to accomplish three tasks:

1. Optimization through file conversion and compression—convert rasters from a variety of file formats to optimized formats: MRF, tiled TIFF, or Cloud Optimized GeoTIFF.
2. Transfer data to and from cloud storage (including Amazon S3, Microsoft Azure, and Google Cloud Storage) or enterprise storage.
3. Create raster proxies—small files stored on local files systems that reference much larger files stored in cloud storage—which simplify data management.

Benefits of using OptimizeRasters to manage raster collections include:

- Streamlined data management
- Faster read performance
- Simplified transfer into and out of cloud storage
- Minimized storage requirements

This OptimizeRasters package contains the following geoprocessing tools:

OptimizeRasters Tool: Converts rasters to optimized output formats, transfers data to and from the cloud, and creates raster proxies

Profile Editor Tool: Maintains cloud storage profiles for Amazon S3 and Microsoft Azure

Resume Jobs Tool: Tracks incomplete jobs and allows the user to resume later

What are the system requirements?
- Python 2.7+ or Python 3.0+ (installed with ArcMap 10.4+ and ArcGIS Pro 1.4+, respectively)
- The OptimizeRasters geoprocessing toolbox requires ArcMap 10.4.1+ or ArcGIS Pro 1.3+
- Windows or Linux
- OptimizeRasters can be run from the command line even if ArcGIS is not installed
- OptimizeRasters can be used with Amazon Web Services Lambda serverless compute service (for rasters smaller than 500 MB)

What's in this document?
- An introduction to OptimizeRasters
- A description of common workflows using OptimizeRasters
- Instructions for installing the OptimizeRasters geoprocessing tools
- How to use the OptimizeRasters geoprocessing tools and CLI commands
- Appendixes describing technical specifications for working with raster proxies and cloud storage

Introduction

OptimizeRasters is an efficient, configurable, robust tool for converting raster data to optimized file formats, moving data to cloud storage, and creating raster proxies.

Converting raster data to optimized formats

OptimizeRasters converts a variety of non-optimized raster formats into optimized MRF, tiled TIFF, or Cloud Optimized GeoTIFF (COG) formats. The result is more efficient, scalable, and elastic data access with a lower storage cost.

For more information about the differences between these file formats, see Appendix E.

Files can be converted to tiled TIFF, Cloud Optimized GeoTIFF, or MRF format:

Tiled TIFF	- TIFF raster products from data providers are often not tiled internally
	- Tiling minimizes the number of disk access requests to get a subset of pixels
	- Optional JPEG or LZW compression can reduce file sizes
	- Optional pyramids can increase access efficiency at smaller scales
Cloud-Optimized GeoTIFF	- Similar to tiled TIFF, but pyramids are required and the file organization is optimized for HTTP range requests
	- Can provide a slight performance improvement for applications that only view the image at small scales or need to crawl for the metadata
	- Take longer to create than tiled TIFF
	- In ArcGIS, performance improvements are negligible compared to tiled TIFF
MRF	- MRF is a tile-based format developed by NASA specifically for storing and accessing rasters more efficiently.
	- Optional Limited Error Rate Compression (LERC) saves additional storage space while speeding up data access with faster compression and decompression rates

File conversion to tiled TIFF, Cloud Optimized GeoTIFF, or MRF (including optional compression) can speed up read performance in three ways:

- Improving the data structure, which makes data access and transfer (especially from cloud storage) more efficient
- Generating pyramids, which provides faster access to data at smaller scales
- Performing optional JPEG, LERC, or LZW/Deflate compression, which further reduces the amount of data stored and transmitted

Note: All three formats can be read from cloud storage, but read performance depends on minimizing the requests made in order to access the data.

Moving data to cloud storage

As part of the data conversion process, OptimizeRasters can simultaneously transfer raster data to and from cloud (or enterprise) storage, speeding up the process of getting rasters into the cloud.

OptimizeRasters supports Amazon S3, Microsoft Azure, and Google Cloud Storage services.

See Using OptimizeRasters with Cloud Storage for more information on Amazon S3, Microsoft Azure, and Google Cloud Storage.

Creating raster proxies to simplify data management

OptimizeRasters can generate raster proxies to simplify access to raster data stored on cloud or network storage.

Raster proxies are small pointer files, stored on local file systems, that contain minimal metadata and reference much larger raster data files stored remotely. A user can work efficiently with collections of

small raster proxy files, using them in ArcGIS as if they were conventional raster files. ArcGIS uses the raster proxies to access the large-volume, remotely stored raster data as needed. Raster proxy files can also cache tiles read from the slower remote storage, speeding up access and reducing the need to access the same data multiple times. Raster proxies can have any extension and typically are given the same extension as the source files to ensure that raster product support in ArcGIS, which is based on file extensions, works correctly.

Note that raster proxies are not the same as the MRF file format, though they are sometimes confused. MRF is a file format similar to TIFF. A raster proxy is file that references a larger file. When caching is enabled, the cache of the raster proxy is stored as an MRF file. See Appendix A for more information on raster proxy files.

Additional benefits of OptimizeRasters

OptimizeRasters employs processing strategies to maximize efficient data transfer:

- Uses parallel processing to speed up conversion and upload of multiple files simultaneously.
- Writes intermediate data on local fast disk to speed up the conversion process.

OptimizeRasters implements strategies to ensure robust data processing:

- Performs various checks during file conversion and upload to ensure all files are correctly transferred.
- Creates a record of the conversion and upload process with extensive logging, which can be used (1) to investigate unresolvable errors and (2) to help the program resume the conversion at a later time, in cases where the processing was prematurely halted.

OptimizeRasters is both configurable and open-source:

- Uses editable configuration files to define parameters and simplify automation.
- OptimizeRasters code is open source and implemented in Python using GDAL 2.1. As a result, users can adapt and expand the code as new needs arise.
  Note: GDAL 2.1+ without the Kakadu driver, as implemented in OptimizeRasters, is equivalent to the ArcGIS 10.5.1 GDAL DLLs.
- The GDAL library enables OptimizeRasters to handle a wide variety of raster file formats.

OptimizeRasters installation

Windows installation

1. Download the OptimizeRasters setup file from GitHub. In a browser, navigate to https://github.com/Esri/OptimizeRasters/raw/master/Setup/OptimizeRastersToolsSetup.exe. The file should begin downloading immediately.

2. Double click the downloaded OptimizeRastersToolsSetup.exe file and step through the wizard to install.

   Note: By default, OptimizeRasters is installed in the

C:\Image_Mgmt_Workflows\OptimizeRasters directory. If you use a different directory, ensure the OptimizeRasters user has write access to it so OptimizeRasters report files can be written.

Installing third-party Python packages for cloud storage

If you are using OptimizeRasters to read to or write from cloud storage (Amazon S3, Microsoft Azure, or Google Cloud Storage), you will need to install additional packages for Python:

Cloud platform	Python package needed
Amazon S3	boto3 (boto is no longer supported)
Microsoft Azure	azure
Google Cloud Storage	google-cloud-storage (google-cloud is no longer supported)

ArcGIS Pro
1. Clone the Pro Python environment. Before installing third-party Python packages, you should go to the Python Package Manager tab in Pro and click **Manage Environments** to clone the Python environment. New packages should then be added to the clone.

    Note: There is a known issue cloning Python environments in Pro 2.2; this blog post provides a command line workaround if you run into problems.

2. Install packages.
    a. boto3 and azure: The boto3 and azure Python packages can be installed directly by selecting **Add Packages** in the Python Package Manager in ArcGIS Pro.
    b. google-cloud-storage: The google-cloud-storage Python package will need to be installed manually. Pip (a common package manager for Python, included with Pro) can be used to install these packages:
        1. In a Windows command prompt, navigate to the Scripts folder in the cloned Pro Python environment you plan to use (e.g. C:\Users\[your_username]\AppData\Local\ESRI\conda\envs\arcgispro-py3clone\Scripts).
        2. Enter **pip install google-cloud-storage** to install the Python package

ArcMap
The Python packages need to be installed manually. Pip (a common package manager for Python) can be used to install these packages (Pip is installed with ArcMap; learn more here):
1. In a Windows command prompt, navigate to the Scripts folder in the Python27 directory that was installed with ArcMap (e.g. C:/Python27/ArcGIS10.6/Scripts).

2. Use a pip install command to install the necessary packages (**pip install boto3**, for example)

Updating third-party Python packages for cloud storage

If these packages are already installed, ensure you are running the most up-to-date version before working with OptimizeRasters in the cloud. To update these packages, open a Command Prompt and run the relevant command:

To upgrade azure: **pip install --upgrade azure.storage.blob**

To upgrade boto3 (for Amazon S3):   **pip install --upgrade boto3**
To upgrade google cloud: **pip install --upgrade google-cloud-storage**

Linux/Unix installation

1. Download the OptimizeRasters ZIP file. To do this, navigate to the OptimizeRasters GitHub repo (https://github.com/Esri/OptimizeRasters), then select **Clone or Download > Download ZIP**
2. Unzip the OptimizeRasters-master.zip file and do the following:
   a. Delete all the files in the GDAL/bin folder of OptimizeRasters
   b. Copy your flavor of Linux GDAL binaries and the shared libraries into the GDAL/bin folder.
      Note: If you already have GDAL installed on your Linux box, only included the following binaries into GDAL/bin: gdalinfo gdaladdo gdalbuildvrt gdal_translate gdalinfo
3. Set the Linux environment variable LD_LIBRARY_PATH to point at the GDAL shared library path (e.g. export LD_LIBRARY_PATH=~/OptimizeRasters-master/GDAL/bin or export LD_LIBRARY_PATH=~/path/to/your/shared/GDAL/library)
4. Re-zip the updated OptimizeRasters-master file, transfer to Linux, unzip in your desired location

At this point, you can open a terminal and type **python OptimizeRasters.py**, followed by the CLI flags described in the documentation, to begin processing.

Installing third-party Python packages for cloud storage

If you will be transferring data in and out of cloud storage, you'll need to install the appropriate Python package:

Cloud platform	Python package needed
Amazon S3	boto3 (boto is no longer supported)
Microsoft Azure	azure
Google Cloud Storage	google-cloud-storage (google-cloud is no longer supported)

To install via pip at the console, enter the relevant command:
   sudo pip install boto3
   sudo pip install azure.storage.blob
   sudo pip install google loud
Credentials needed to access cloud storage should also be placed in the appropriate location:

Cloud platform	Credential location
Amazon S3	AWS credential file should be placed in the ~/.aws folder
Microsoft Azure	Azure credentials should be placed in the file ~/.OptimizeRasters/Microsoft/azure_credentials
Google Cloud Storage	Google service account JSON files should be placed in the ~/.OptimizeRasters/Google folder

Common OptimizeRasters Workflows

OptimizeRasters assists with both image management and sharing. Below are example scenarios illustrating the types of situations in which a user might use common OptimizeRasters workflows.

Optimizing raster data for enterprise storage

A satellite imagery vendor delivers imagery as TIFF files, along with auxiliary metadata files, that are not tiled and do not have pyramids.

Workflow: Use OptimizeRasters to transfer all raster files from one directory to another (e.g. from an external hard disk to your organization's shared file storage). Simultaneously, OptimizeRasters will convert the TIFFs into tiled TIFFs with internal pyramids. The conversion can be lossless (depending on compression method) and the filenames are unchanged, but the TIFFs will be faster to access.

Note: This is similar to the ArcGIS Copy Rasters command, but preserves all the data files (not just the raster files) and handles nested directories.

Configuration templates to use: Imagery_to_TIF_JPEG or Imagery_to_TIF_LZW

Source raster data is taking up too much storage space.

Workflow: Use OptimizeRasters to convert files using a compression option to reduce storage space. Compression can be lossless (e.g. using Deflate) or lossy (e.g. using JPEG). The user can also convert to MRF and use Limited Error Raster Compression (LERC), which can be lossless or controlled lossy.

Configuration templates to use: Imagery_to_MRF_JPEG, Imagery_to_MRF_LERC, Imagery_to_TIF_JPEG or Imagery_to_TIF_LZW

A vendor delivers the source raster data in a slow-to-read JP2 format.

Workflow: Use OptimizeRasters to transfer all raster files from one directory to another while converting the format to tiled TIFF (or MRF), which is faster to read (though the file size may be larger).

Note: It is possible to convert the files while keeping the original filenames, which may be required when a metadata file references the raster data file by name. OptimizeRasters also includes a custom extension naming option.

Configuration templates to use: Imagery_to_MRF_JPEG, Imagery_to_MRF_LERC, Imagery_to_TIF_JPEG or Imagery_to_TIF_LZW

Optimizing raster data for cloud storage

Your organization wants to move existing raster data into cloud storage.

Workflow: Use OptimizeRasters to transfer raster data to cloud storage by defining the output directory as cloud storage (such as S3). The user will usually convert the raster format to MRF (or Cloud Optimized GeoTIFF) during the transfer.

Configuration templates to use: CopyFilesOnly, Imagery_to_COG_DEF, Imagery_to_COG_JPG Imagery_to_MRF_JPEG, Imagery_to_MRF_LERC, Imagery_to_TIF_JPEG or Imagery_to_TIF_LZW

Your organization needs to efficiently access large collections of rasters stored in the cloud.

Workflow: Use OptimizeRasters to create raster proxy files. Raster proxies (small, local pointer files) can be used as if they were standard raster datasets and raster products in ArcGIS, but take only a small fraction of the original disk space. Only the required pixels will be read from raster data files in cloud storage, and those pixels can be locally cached so subsequent requests to the same area are very fast.

Note: OptimizeRasters can create raster proxy files that access raster data in a variety of formats as long as the rasters are tiled or have pyramids. Raster proxies are most efficient when accessing rasters stored as MRF files. If the imagery is not stored in a public bucket/container, the machine with the raster proxies must have access to the bucket/container.

Configuration templates to use: CreateRasterProxy

Your organization has stored non-optimized raster files in the cloud, and wishes to convert them to an optimized raster format in the cloud.

Workflow: Use OptimizeRasters to convert the original raster files to either MRF or Cloud Optimized GeoTIFF in the cloud. For best results, run OptimizeRasters in the cloud close to where the input and/or output data will be stored, on infrastructure such as Amazon EC2 instances or Azure compute. Alternatively, OptimizeRasters can be run using Lambda serverless compute.

Note: During the conversion process, it is standard to create raster proxy files to reference the new rasters. The raster proxy files can then be transferred to local storage, enabling more efficient, virtual access to the data stored in the cloud.

Configuration templates to use: Imagery_to_COG_DEF, Imagery_to_MRF_JPEG, Imagery_to_MRF_LERC

Your organization wants to embed raster proxy files in the mosaic dataset to eliminate dependency on local files.

Workflow: Use OptimizeRasters to generate raster proxies, but point to a CSV file instead of a folder location. A table of raster proxies will be generated instead of individual files. The raster proxies can be embedded in a mosaic dataset, bypassing any local files and making it simpler to transfer the mosaic dataset to another machine.

Configuration templates to use: CreateRasterProxy, Imagery_to_COG_DEF, Imagery_to_MRF_JPEG, Imagery_to_MRF_LERC, Imagery_to_TIF_JPEG or Imagery_to_TIF_LZW

Accessing Landsat 8 data from Amazon Public Data Sets

You wish to use ArcGIS Desktop to access free Landsat 8 data from Amazon, but you do not want to download the full dataset (more info at http://aws.amazon.com/public-datasets/landsat).

Workflow: Use OptimizeRasters to create raster proxy files to access Landsat 8 data in ArcGIS.

The resulting output directory will contain the specified Landsat scene as eight raster proxy TIFF files (and a corresponding .met file), which will then reference the source TIFFs stored in the cloud.

These raster proxy files can be used as a raster product in ArcGIS, but the size of the files will only be a few KB.

A sample command line entry for accessing this data (using the provided OptimizeRasters configuration file) would be:

"C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\python.exe"
C:\Image_Mgmt_Workflows\OptimizeRasters\OptimizeRasters.py -
***config**=C:\Image_Mgmt_Workflows\OptimizeRasters\Templates\Landsat8_RasterProxy.xml -*
***input**=c1/L8/160/043/LC08_L1TP_160043_20180326_20180404_01_T1 -**clouddownload**=true -*
***inputbucket**=landsat-pds -**clouddownloadtype**=amazon -*
***output**=c:\temp\landsatpdsdat\c1\L8\160\043\LC08_L1TP_160043_20180326_20180404_01_T1*

Configuration templates to use: Landsat8_RasterProxy (The user must provide the input path, the Landsat path/row, and the scene ID of the desired Landsat 8 scene)

Using OptimizeRasters in ArcGIS Pro

The OptimizeRasters tool allows you to convert most raster formats to tiled TIFF, Cloud Optimized GeoTIFF, or MRF file formats. It includes options to transfer data into or out of cloud storage and enables the creation of raster proxies (either during the conversion process or as a separate step).

Most parameters for common operations will be defined in the provided configuration files. Other parameters will need to be selected in the tool dialog box or specified at the command line.

Note: OptimizeRasters also includes two additional tools, described in Appendix B. The **Profile Editor tool** allows you to store credential profiles for Amazon S3 or Microsoft Azure cloud storage, simplifying access to secured cloud storage. The **Resume Jobs tool** will show a list of pending jobs, and allow you to resume any processes that failed to complete. This is helpful when working with large projects where interruptions and some failures can occur.

How to use the OptimizeRasters tool

To access the OptimizeRasters Python toolbox, follow these steps:

1. Start ArcGIS Pro and create a new project.

2. In the Catalog Pane, right click Toolboxes and select **Add Toolbox**. Navigate to the OptimizeRasters directory (C:\Image_Mgmt_Workflows\OptimizeRasters), select OptimizeRasters.pyt, and click **OK**.

3. Open the toolbox by clicking the arrow next to OptimizeRasters.pyt (see Figure 1).
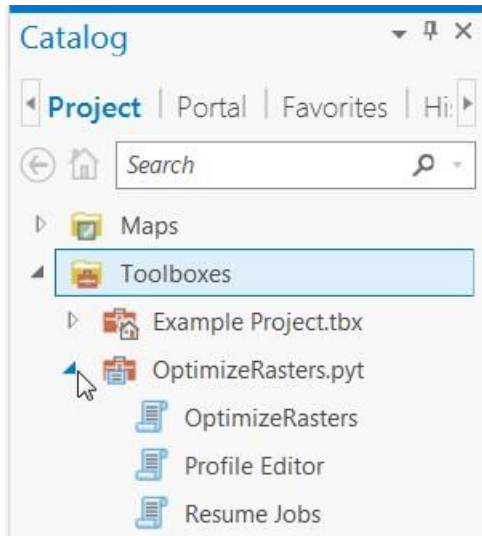
Figure 1: OptimizeRasters Catalog Pane

To use the OptimizeRasters geoprocessing tool, follow these steps:

1. If you will read or write from cloud storage, create the required Amazon S3, Microsoft Azure, or Google Cloud Storage profile. See Using OptimizeRasters with Cloud Storage, below.

2. Double-click to open the **OptimizeRasters tool** from the OptimizeRasters Toolbox.

3. Complete the OptimizeRasters dialog (Figure 2) to perform jobs. A brief description of each parameter can be found in Table 1.
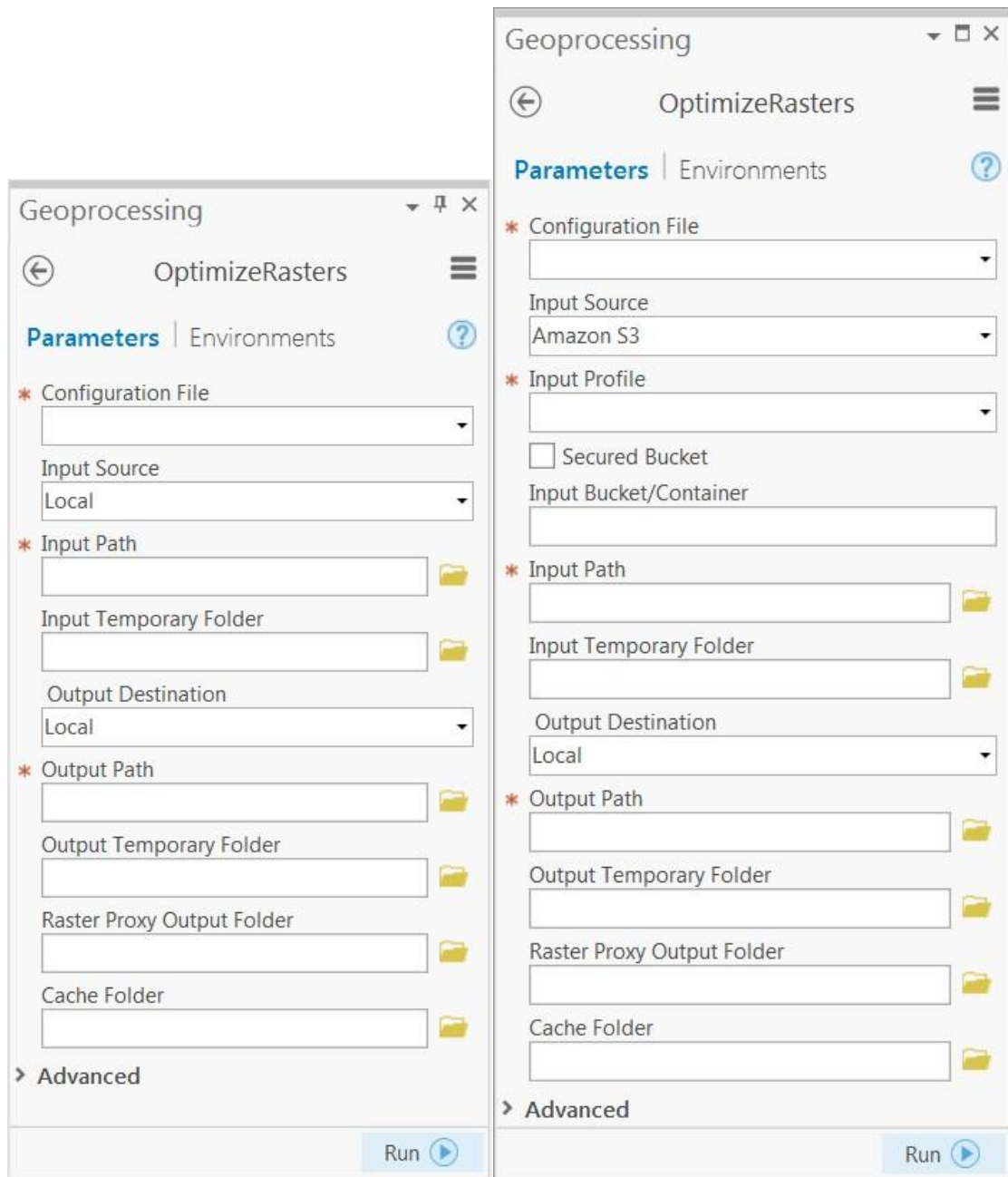
Figure 2: OptimizeRasters dialog for local and cloud input data

Table 1: Description of OptimizeRasters dialog options

Dialog parameter	Description
Configuration File	(**Required**) These templates set the appropriate default parameters for a given task (like converting file formats or creating raster proxies). Use the dropdown list to select a template that suits your needs. See <u>Configuration file templates</u>, below, for a more detailed description.

Input Source	(**Required**) The input storage type. Options include: Local, Amazon S3, Microsoft Azure, and Google Cloud.

Dialog parameter	Description
Input Profile	(**Required for input from cloud storage)** The cloud storage user profile for your input data. To add or edit a profile, use the Profile Editor tool (discussed below). Note: If the input source is Amazon S3, "Using_EC2 Instance_with_IAM_Role" will be an option. Assuming you are working from an EC2 instance with IAM role access to the input S3 bucket, this selection allows secure access to AWS resources without requiring credential keys.
Secured Bucket	(Optional) Check on if your bucket is secure (you'll need credentials to access it). If it's public, leave this unchecked. If checked on, vsis3 will be used to access the data instead of vsicurl.
Input Bucket / Container	(**Required for input from cloud storage)** The name of the Amazon S3 bucket, Google bucket, or Azure container holding the input files. This will be automatically populated based on your Input Profile. See Accessing Cloud Storage, below.
Input Path	(**Required**) The local or cloud storage folder where input data is located. For input from local storage, this will be a directory name, like C:\MyData\Samples. For input from cloud storage, this will be the folder path without the bucket/container name. See Accessing Cloud Storage, below. Note: The browse button will only work with local storage.
Input Temporary Folder	(Optional) A local folder used to temporarily hold input files while downloading from cloud storage. Data will be first downloaded to this location and then converted. Note: Recommended if space is limited in the default system temp location.
Output Destination	(**Required**) The output storage type. Options include: Local, Amazon S3, Microsoft Azure, and Google Cloud.
Output Profile	(**Required for output to cloud storage**) The cloud storage user profile for your output data. To add or edit a profile, use the Profile Editor tool (discussed below). Note: If the output source is Amazon S3, "Using_EC2 Instance_with_IAM_Role" will be an option. Assuming you are working from an EC2 instance with IAM role access to the output S3 bucket, this selection allows secure access to AWS resources without requiring credential keys.

Output Bucket/Container	(**Required for output to cloud storage**) The name of the Amazon S3 bucket, Google bucket, or Azure container holding the output files. This will be automatically populated based on your Output Profile. See Accessing Cloud Storage, below.
Output Path	(**Required**) The local or cloud storage folder where output data will be stored. For output to local storage, this will be a directory name, like C:\MyData\Samples. For output to cloud storage, this will be the path without the bucket/container name. (The browse button will only work with local storage.) See Accessing Cloud Storage, below.

Dialog parameter	Description
	Note: If the Mode is RasterProxy (in the CreateRasterProxy config file, for example), the output path can also point to a .csv file (existing or not). The contents of the raster proxies will be saved to the .csv file as rows, and individual raster proxies will not be generated.
Output Temporary Folder	(**Required for output to cloud storage**) A local folder used to temporarily hold output files before they're uploaded to a final location. The output will be first written to this directory, then transferred to the cloud or enterprise storage. For locally stored output, this is recommended if storage space is restricted in the default system temp location, or when the output location is a network drive. Using a faster temporary drive will speed processing, since the full resolution data is written first and then read again to create pyramids.
Raster Proxy Folder	(Optional) A local output folder where raster proxies, which are created at the same time data is transferred, will be stored. Alternatively, the Raster Proxy Folder can also point to a .csv file (existing or not). In this case, the contents of the raster proxies will be saved to the .csv file as rows, and individual raster proxies will not be generated.
Cache Folder	(Optional) An output folder for cache files used with raster proxies. Recommended cache path is Z:\mrfcache\. See Appendix A: Working with Raster Proxy Files: Cache Management, below, for more information. Note: If left blank, the cache will be stored in the same directory as the raster proxy files. A separate directory is recommended to make it easier to empty the cache periodically.

Accessing cloud storage

Example cloud storage path: If

    the **complete path** is…

    http://landsat-pds.s3.amazonaws.com/L8/139/045/LC81390452014295LGN00

    Then…

    The **bucket/container name** is: landsat-pds

The **input path** is: L8/139/045/LC81390452014295LGN00

For more information on Amazon S3, refer to
http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html#access-bucket-intro.

For information on Microsoft Azure, refer to
https://docs.microsoft.com/enus/rest/api/storageservices/fileservices/naming-and-referencing-containers--blobs--andmetadata#resource-uri-syntax.

Configuration file templates

OptimizeRasters comes with configuration template files that set OptimizeRasters parameters for a variety of common workflows and data types (see Table 2).

Once a configuration template file is selected in the OptimizeRasters dialog, the parameters set by that file can be viewed and edited in the Advanced section of the OptimizeRasters tool (or with a text editor). The configuration template files are located in the Templates folder in the OptimizeRasters directory location.

See Configuration file parameters (advanced options) for more detailed descriptions of the parameters defined in each file.

Table 2: Configuration File Templates

Default configuration files	Description
Airbus_SatelliteProduct_to_MRF_LERC	Converts Airbus Satellite Product data to MRF using LERC compression.
CopyFilesOnly	Copies between different storage systems (S3, Azure, Google Cloud, and local storage) with no file conversion and without creating raster proxies.
CreateRasterProxy	Creates raster proxy files from various raster file formats.
DG_SatelliteProduct_to_MRF_LERC	Converts Digital Globe Satellite imagery to MRF using LERC compression.
Imagery_to_COG_DEF	Converts imagery to Cloud Optimized GeoTIFF format using deflate compression.
Imagery_to_COG_JPEG	Converts imagery to Cloud Optimized GeoTIFF format using JPEG compression.
Imagery_to_MRF_JPEG	Converts imagery to MRF using JPEG compression.
Imagery_to_MRF_LERC	Converts imagery to MRF using LERC compression.
Imagery_to_MRF_ZLERC	Converts imagery to MRF using LERC compression and additional Deflate. Can be beneficial for sparse datasets (from small sensors or non-IT devices)
Imagery_to_TIF_JPEG	Converts imagery to TIF using JPEG compression.
Imagery_to_TIF_LZW	Converts imagery to TIF using LZW compression.

Landsat8_RasterProxy	Creates raster proxy files from Landsat 8 data stored in Amazon S3 Public Data Sets.
Landsat_to_MRF_LERC	Converts Landsat imagery to MRF using LERC compression.
Overviews_to_MRF_JPEG	Converts overview imagery to MRF using JPEG compression.
Overviews_to_MRF_LERC	Converts overview imagery to MRF using LERC compression.
Sentinel2_to_MRF	Converts Sentinel-2 imagery to MRF using LERC compression.

Configuration file parameters (advanced options)

For most users, one of the existing configuration files should be sufficient without any changes.

If customization is necessary, generally the user will start with one of the predefined configuration files, edit parameters under advanced options, then run the process. (The selected configuration settings will be saved as a new, timestamped configuration file.)

To edit configuration file parameters, complete the following steps:

1. In the OptimizeRasters tool dialog, select the configuration file you wish to modify.

2. To edit values, expand the Advanced section and select the check box labeled **Edit Configuration Values**.
Note: Select **Show Help** at the bottom of the dialog box to view a list of options for each
   parameter.
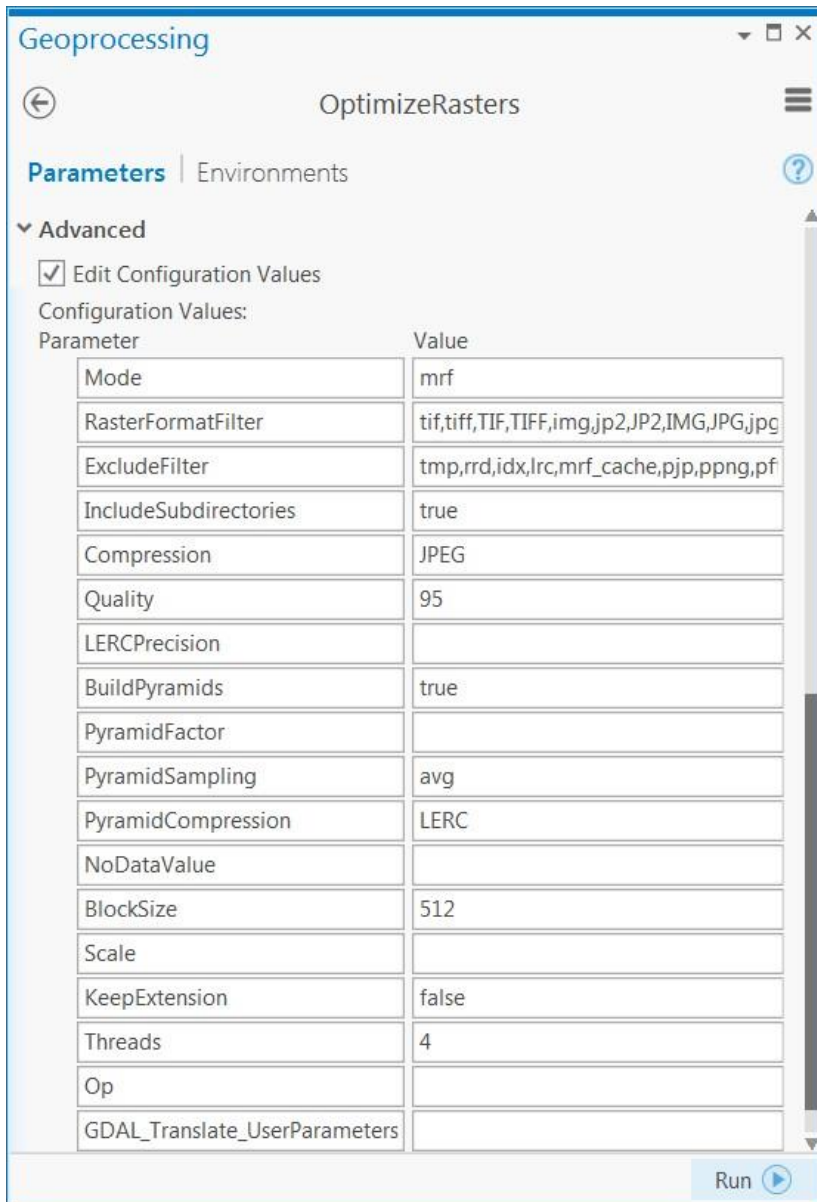
Figure 3: Example OptimizeRasters advanced options for Imagery_to_MRF_JPEG config file

3. After editing the desired parameters, click **Run** to implement them. The new values will be saved to a new configuration file, with the timestamp appended to the original filename.

   Note: If the user-created configuration file is edited again, the new changes will be saved to the same file.

Table 3 contains a list of possible configuration file parameters (for parameters related to cloud storage, see the Using OptimizeRasters with Cloud Storage).

Table 3: OptimizeRasters configuration file parameters

Configuration File Parameter	Description

Mode	Defines the output file format. Options include: • mrf (used to convert input data to MRF) • tif (used to convert input data to tiled TIFF) • tif_cog (used to convert input data into Cloud Optimized GeoTIFF) • rasterproxy (used to generate raster proxies without copying or converting the original data)
RasterFormatFilter	Defines the file extension of rasters that should be converted. Typically this is set to "tif,TIF,mrf,tiff"—rasters with these extensions will be converted. All other files (e.g. a JPEG file containing a logo) are simply copied from source to destination. Note: File extensions are case sensitive.
ExcludeFilter	Defines the file extensions that will not be copied. Examples include files that may not be needed once the rasters are converted (e.g. ovr, rrd, tfw, or aux.xml files) or files that would typically be replaced (e.g. idx, lrc, mrf_cache, pjp, ppng, pft, or pzp files).
IncludeSubdirectories	If true, OR will scan for rasters in subdirectories. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Compression	Compression type to use on output rasters. Acceptable values are: LERC, JPEG, LZW, DEFLATE.
Interleave	(Optional) Defines the interleave setting for compression. Acceptable values are: Band, Pixel. (Default is Pixel) Note: Compression is improved when the output is 3band and the interleave is Pixel.
Quality	JPEG compression quality (default is 85)
LERCPrecision	LERC compression precision (Default is 0.5 for integer data and 0.001 for float data)
BuildPyramids	Determines if pyramids should be built. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n. Note: setting -pyramids=external will create external pyramids.
PyramidFactor	Pyramid levels to create. The default value is a factor of 2, resulting in pyramids as follows: 2 4 8 16 32 64. It can also be set to a factor of 3 (e.g., 3 9 27 81).
PyramidSampling	Defines sampling to use for pyramids. The default is: average sampling. Acceptable values are: average, nearest, gauss, cubic, cubicspline, lanczos, average_mp, average_magphase, mode, AVG (a form of average where each pixel is the average of 4. Only available for MRF)
PyramidCompression	Compression type to use for pyramids (default is jpeg). Acceptable values are: jpeg, lzw, deflate.

Configuration File Parameter	Description
NoDataValue	Specifies the NoDataValue in the source (if applicable). The default is: undefined. Note: If rasters have 0 as NoData, the NoDataValue parameter must be set so pyramid generation will not include this value.
BlockSize	Defines tile size in the output image. A value of 512 is recommended for most datasets. Note: Smaller values that are powers of 2 (e.g. 128, 256) may be better for datasets used to generate temporal profiles.
KeepExtension	Specifies whether output raster file extensions (and accompanying raster proxy extensions) should be changed. If true, the extension remains the same as the input, even if the format was changed (e.g. to MRF). If false, the extension will be based on the output format (typically .tif or .mrf). Note: This parameter will apply to the raster proxies, as well.
Threads	Defines the number of simultaneous threads used for parallel processing (default is 10).
LogPath	Defines the location for log files. By default, these XML files are created in the Logs directory in the same location as OptimizeRasters.py. An alternative location must be a network path; it cannot be on cloud storage.
GDAL_Translate_UserParameters	User-defined GDAL_Translate values. Values will be passed on without any modification. Refer to http://www.gdal.org/gdal_translate.html for more options. Examples: (Refer to the link above for full explanations.) • -scale 200 255 0 255 (Rescale the input pixel values) • -outsize 10% 10% (Set the size of the output file) • -Scale 0 65536 1 256 (scale from 16 to 8 bit and leave 0) • -b 1 -b 2 -b 3 (Extract 3 bands) • -co NBITS=8 (Output should be 8bits) • -ot Byte (Set output to byte) • -scale 0 65535 0 4096 -co NBITS=12 -ot UInt16 (rescales input pixel values, converts to 12-bit, and uses 12-bit JPEG compression)

	Flag used for operational settings. Acceptable values include:
Op	• Noconvert (will copy raster files between different storage types without performing any file conversion) • Upload (will back up a local -input to S3, Azure, Google, or local storage) • copyonly (will copy files without conversion or creating raster proxies) • lambda (should be used when implementing OptimizeRasters as an Amazon Web Services lambda function)

Note that for local or network input/output, the following parameters can't be set in the configuration file. They must be specified either in the tool dialog discussed above, or using command line flags:

- Input path   • Output temporary folder
- Input temporary folder   • Raster proxy folder
- Output path   • Cache folder

Using OptimizeRasters at the Command Line

OptimizeRasters can be called as a command line tool, even if ArcGIS Desktop is not installed.

Standard command line usage

The standard command line usage uses the following syntax:

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

For help with OptimizeRasters command line, enter the following:

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

Command line arguments

The following parameters and switches can be used with OptimizeRasters at the command line, for local, enterprise, or cloud-based input and output.

Note: Some below parameters can also be specified in the configuration file. Command line parameters will override the configuration file.

Table 4: OptimizeRasters command line arguments

Command Line Argument	Description
-input=	Path to input directory or job file. Note: All files in a directory will be processed. If the input is a job file, no other parameters are needed—this is the equivalent of running the Resume Geoprocessing tool.

-output=	Path to a local or enterprise output directory. If the mode is rasterproxy, this can also point to a .csv file (existing or not). In this case, the contents of the raster proxies will be saved to the .csv file as rows, and individual raster proxies will not be generated.
-config=	Path to the configuration file with default settings used to set any parameters not defined on the command line. If undefined, then the OptimizeRasters.xml file, stored in the same location as OptimizeRasters.py, will be used.
-mode=	Defines the output file format. Options include: • mrf (used to convert input data to MRF) • tif (used to convert input data to tiled TIFF) • tif_cog (used to convert input data into Cloud Optimized GeoTIFF) • rasterproxy (used to generate raster proxies without copying or converting the original data)

Command Line Argument	Description
	• createjob (generates an OptimizeRasters .orjob file, but won't automatically start processing the file list. This is useful if users need to see the generated file list before processing.)
-cache=	(Optional) An output folder for cache files used with raster proxies. Recommended cache path is Z:\mrfcache\. See Appendix A: Working with Raster Proxy Files: Cache Management, below, for more information. Note: If left blank, the cache will be stored in the same directory as the raster proxy files. A separate directory is recommended to make it easier to empty the cache periodically.
-quality=	JPEG compression quality (default is 85)
-prec=	LERC compression precision (Default is 0.5 for integer data and 0.001 for float data)
-pyramids=	Determines if pyramids should be built. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n. Pyramids are created by default, but this flag will override the BuildPyramids parameter in config file. Note: setting -pyramids=external will create external pyramids.
-op=	Operation parameter. Acceptable values include: • Noconvert (will copy raster files between different storage types without performing any file conversion) • Upload (will back up a local -input to S3, Azure, Google, or local storage) • copyonly (will copy files without conversion or creating raster proxies) • lambda (should be used when implementing OptimizeRasters as an Amazon Web Services lambda function)

Command Line Argument	Description
-subs=	If true, OR will scan for rasters in subdirectories. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n. If true, all subdirectories of the input directory will also be processed.
-job=	User-defined name for the job file generated when OptimizeRasters is run.
-tempinput=	A local folder used to temporarily hold input files while downloading from cloud storage. Data will be first downloaded to this location and then converted. Note: Recommended if space is limited in the default system temp location.
-tempoutput=	(**Required for cloud output**) A local folder used to temporarily hold output files before they're uploaded to a final location. The output will be first written to this directory, then transferred to the cloud or enterprise storage. For locally stored output, this is recommended if storage space is restricted in the default system temp location, or when the output location is a network drive. Using a faster temporary drive will speed processing, since the full resolution data is written first and then read again to create pyramids.
-clouddownload=	Flag to let the program know whether the input location is cloud storage or not. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Command Line Argument	**Description**
-clouddownloadtype=	Parameter linked to -clouddownload (default is Amazon if -clouddownload is true and -clouddownloadtype is unspecified). Acceptable values are: Amazon, Azure, Google.
-inputprofile=	Cloud profile name that corresponds with the credential file for the input cloud storage type. The profile name will be used to pick up the relevant credentials. Note: If Google is selected as the cloud download type and you're using this parameter at the command line, this parameter must point to a JSON service account key file on the local machine.
-inputbucket=	Input cloud bucket/container name. The specified credentials must have access to this location.
-usetoken	Set to true if the input and/or output bucket is private. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
-cloudupload=	Flag to let the program know whether the output location is cloud storage or not. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
-clouduploadtype=	Parameter linked to -cloudupload (Default is Amazon if -cloudupload is true and -clouduploadtype is unspecified). Acceptable values are: Amazon, Azure, Google.

-outputprofile=	Cloud profile name that corresponds with the credential file for the output cloud storage type. The profile name will be used to pick up the relevant credentials. Note: If Google is selected as the cloud upload type and you're using this parameter at the command line, this parameter must point to a JSON service account key file on the local machine.
-outputbucket=	Output cloud bucket/container name. The specified credentials must have access to this location.
-rasterproxypath=	Path to a local directory where raster proxy files are generated during the conversion process. This can also point to an XML file, in which case the raster proxies will be added as rows in the file (individual raster proxy files won't be created). Note: This is equivalent to the Raster Proxy Output folder in the UI, and will be ignored if the mode is rasterproxy.

Note: If you want to set non-default values for any of the following, you will need to use a configuration file (see: OptimizeRasters Tool: Configuration file parameters, above). If there is a default value, it's provided in parentheses below. If you're using a provided configuration file, confirm the value set there.

RasterFormatFilter	PyramidCompression	DeleteAfterUpload (true)
ExcludeFilter	NoDataValue	Out_S3_ACL
Compression	BlockSize (512)	LogPath (OptimizeRasters logs
Interleave	KeepExtension (false)	folder)
PyramidFactor	Threads (4)	
PyramidSampling	GDAL_Translate_UserParameters	

Using OptimizeRasters with Cloud Storage

OptimizeRasters supports three cloud storage providers: Amazon S3, Microsoft Azure, and Google Cloud Storage. This section covers general guidelines for transferring data into and out of these three storage options using OptimizeRasters, including valid configuration file parameters for each. For more information about working with cloud storage at the command line, see Using OptimizeRasters: Command Line.

Working with Amazon S3

To work with Amazon S3, you must first install the boto3 third-party Python package. See OptimizeRasters installation, above.

Note: File names in S3 buckets are case sensitive, so it is very important not to change the case of the file names or their extensions when copying data to S3, or when referencing these files.

Managing S3 credentials

OptimizeRasters supports the AWS standards to manage credentials (more information can be found at http://docs.aws.amazon.com/cli/latest/reference/configure/index.html).

This means credentials can be set up the following ways:

1. (**Recommended**) The user can use the AWS credential file located at %USERPROFILE%\.aws\credentials. This can be managed with the Profile Editor tool.

    Note: If you are using the OptimizeRasters GP tool, this is required.

2. The user can use environment variables to set credentials on their machine (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)

3. The user can set AWS credentials using Amazon command line tools. If you are using an EC2 instance, these tools are already set up on your system. If the tools are not already installed, they can be downloaded at http://aws.amazon.com/cli. You can use the following command to set up the credentials:

    $ aws configure

4. (**Not recommended**) The user can set AWS credentials in the OptimizeRasters configuration file or using OptimizeRasters command line.

Security is the primary advantage of using an AWS credential file—the S3 keys for the default user will be stored using the access security offered by the OS. The credential file will only be accessible to the user who already has the read/write access to the profile location. If the OptimizeRasters package is copied onto another machine, credentials available in the OptimizeRasters configuration file will not be unintentionally exposed.

Configuration file parameters to read/write to S3

The following table lists all the S3-specific configuration file parameters and acceptable values. Values added to the parameter files will be used unless they aren't overridden at the command line. (See Using OptimizeRasters: Command Line for equivalent command line parameters where they exist.)

Table 3: Configuration file parameters for S3 storage

Configuration File Parameter: S3	Description
CloudDownload	Set to true if the input will be from the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
In_Cloud_Type	Specifies the cloud storage provided for the input data. Valid options are: amazon, azure, google. Note: For S3 storage, enter **amazon**.
In_S3_AWS_Profile_Name	Use this parameter to select a profile name for the AWS security credentials you wish to use to access the input location. Note: This must match a profile name in the AWS credential file.
In_S3_ID In_S3_Secret	(**Not recommended**) Use these parameters to set the AWS access key and secret key needed to access the input location. The keys in the configuration file will take precedence over the AWS standard credential manager. Note: If these keys are embedded in the configuration file, it is more likely that they will accidentally be shared.
UseToken	Set to true if the input and/or output bucket is private. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.

In_S3_Bucket	The name of the bucket holding the input files. The specified input credentials must provide access to this folder.
In_S3_ParentFolder	The input folder path where the input data is located. Exclude the bucket name.
CloudUpload	Set to true if the file destination is in the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Out_Cloud_Type	Specifies the cloud storage to be used for the output data. Valid options are: amazon, azure, google. Note: For S3 storage, enter **amazon**.
Out_S3_AWS_Profile_Name	Use this parameter to select a profile name for the AWS security credentials you wish to use to access the output location. This must match a profile name in the AWS credential file.
Out_S3_ID Out_S3_Secret	(**Not recommended**) Use these parameters to set the AWS access key and secret key needed to access the output location. The keys in the configuration file will take precedence over the AWS standard credential manager Note: If these keys are embedded in the configuration file, it is more likely that they will accidentally be shared.
Out_S3_Bucket	The name of the bucket where the output files will be located. The specified output credentials must provide access to this folder.
Out_S3_ParentFolder	The output folder path where the output data will be stored. (Exclude the bucket name)
DeleteAfterUpload	Set to true to delete temporary rasters/files after processing is complete. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Configuration File Parameter: S3	**Description**
Out_S3_ACL	Defines the canned Access Control List to apply to uploaded files. Default is public-read. For secure buckets, the bucket policy takes priority. Acceptable values are: private, public-read, public-readwrite, authenticated-read, bucket-owner-read, bucket-ownerfull-control

For cloud input/output, the following parameters can't be set in the configuration file. They must be specified either in the OptimizeRasters tool dialog discussed above, or at the command line:

- Input temporary folder
- Output temporary folder
- Raster proxy folder
- Cache folder

Example S3 workflows

Following are example configuration file parameters needed for common S3 workflows. These all assume that the relevant AWS credential file contains the following profile:

```
[OptimizeRasters_S3Out] aws_access_key_id =
XXX_YOUR_ACCESS_KEY_ID_XXX
aws_secret_access_key =
XXX_SECRET_ACCESS_KEY_XXX
```

Writing files to S3 storage

You want to upload data to cloud storage located at **http://mydata.s3.amazonaws.com/abc/pqr/t**.

To accomplish this, the following parameters should be added to the configuration file:

```
<CloudUpload>true</CloudUpload>
<Out_Cloud_Type>amazon</Out_Cloud_Type>
<Out_S3_Bucket>mydata</Out_S3_Bucket>
<Out_S3_ParentFolder>abc/pqr/t</Out_S3_ParentFolder>
<Out_S3_AWS_ProfileName>OptimizeRasters_S3Out</Out_S3_AWS_ProfileName>
<Out_S3_DeleteAfterUpload>true</Out_S3_DeleteAfterUpload> Alternatively,
```

you could enter the following at the command line:

python OptimizeRasters.py -cloudupload=true -clouduploadtype=amazon -outputbucket=mydata
output=abc/pqr/t -outputprofile=OptimizeRasters_S3Out -tempoutput=c:/temp/mydata

Reading from public S3 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>
```

Note: See the Landsat8_RasterProxy configuration file as an example.

Generating raster proxies from S3

If you are copying or converting data, you can create raster proxies at the same time. To do this, specify a Raster Proxy Folder in the tool dialog, or set -rasterproxypath at the command line, and the raster proxy files will automatically be generated.

Some users already have rasters in the cloud, and only want to generate raster proxy files. To do this, use the CreateRasterProxy configuration file, which uses rasterproxy mode. (See Appendix A: Working with Raster Proxy Files for more information on rasterproxy mode).

In this case, you can do one of three things:

1. Use the OptimizeRasters tool UI to specify the S3 input parameters and the output raster proxy folder location
2. Edit the configuration file parameters and add command line flags. For example, if the input raster files are in http://mydata.s3.amazonaws.com/abc/pqr/t, the user should make the following changes to the configuration file:

```
<mode>rasterproxy</mode>
<CloudDownload>true</CloudDownload>
<In_Cloud_Type>amazon</In_Cloud_Type>
<In_S3_Bucket>mydata</Out_S3_Bucket>
<In_S3_ParentFolder>abc/pqr/t</Out_S3_ParentFolder>
```

24

<In_S3_AWS_ProfileName>OptimizeRasters_S3In</In_S3_AWS_ProfileName>

And specify the local output location at the command line: -output=<path to outputfolder> or using the OptimizeRasters tool dialog

3. Set all parameters at the command line:

   <path to python.exe> <path to optimizerasters.py> -clouddownload=true -
   clouddownloadtype=Amazon -inputbucket=<S3 bucket name> -input=<path to s3 input folder> output=<path to rasterproxyfolder> -mode=rasterproxy

   Note: -clouddownloadtype is optional for Amazon (the default).

Setting access control on Amazon S3

Access to data in S3 buckets is defined using the Amazon S3 Access Control Lists (ACLs). Find more information at http://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#CannedACL.

By default, OptimizeRasters will use the public-read ACL, which enables all users to read the data. The ACL can also be defined manually in the configuration file by using the Out_S3_ACL parameter. For secure buckets, the bucket policy will take priority.

To avoid making your data public, set the ACL as private and apply a secure S3 bucket policy.

For more information on this, 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
See also Using Obfuscation for Access Control in the Cloud, below.

Working with Microsoft Azure

To work with Azure, you must first install the azure third-party Python package. See OptimizeRasters installation, above.

Managing Azure credentials

If you're using the Profile Editor to manage Azure credentials, they will be saved in the file C:\Users\%username%\.optimizerasters\Microsoft\azure_credentials.

Configuration file parameters to read/write to Azure

The following table lists all the Azure-specific configuration file parameters and acceptable values. Values added to the parameter files will be used unless they aren't overridden at the command line. (See Using OptimizeRasters: Command Line for equivalent command line parameters where they exist.)

Table 6: Configuration file parameters for Microsoft Azure storage

Configuration File Parameter: Azure	Description
CloudDownload	Set to true if the input will be from the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
In_Cloud_Type	Specifies the cloud storage provided for the input data. Valid options are: amazon, azure, google. Note: For Azure storage, enter **azure**.

In_Azure_ProfileName	Use this parameter to select a profile name for the Azure security credentials you wish to use to access the input location. Note: This must match a profile name in the Azure credential file.
In_Azure_AccountName In_Azure_AccountKey	(**Not recommended**) Use these parameters to set the Azure account name and account key needed to access the input location. The keys in the configuration file will take precedence. Note: If these values are embedded in the configuration file, it is more likely that they will accidentally be shared.
In_Azure_Container	The name of the container holding the input files. The specified input credentials must provide access to this folder.
In_Azure_ParentFolder	The input folder path where the input data is located. Exclude the container name.
CloudUpload	Set to true if the file destination is in the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Out_Cloud_Type	Specifies the cloud storage provided for the output data. Valid options are: amazon, azure, google. Note: For Azure storage, enter **azure**.
Out_Azure_ProfileName	Use this parameter to select a profile name for the Azure security credentials you wish to use to access the output location. Note: This must match a profile name in the Azure credential file.
Out_Azure_AccountName Out_Azure_AccountKey	(**Not recommended**) Use these parameters to set the Azure account name and account key needed to access the output location. The keys in the configuration file will take precedence. Note: If these values are embedded in the configuration file, it is more likely that they will accidentally be shared.
Out_Azure_Access	Identifies the access type. Valid values are: • private (the container is accessible only to the user) • blob (files within the container are publicly accessible) • container (files within the container and container metadata are publicly accessible)
Out_Azure_Container	The name of the container that will hold the output files. The specified output credentials must provide access to this folder.
Configuration File Parameter: Azure	**Description**
Out_Azure_ParentFolder	The output folder path where the output data will be copied. Exclude the container name.
DeleteAfterUpload	Set to true to delete temporary rasters/files after processing is complete. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.

Azure upload specifications

Specifications for uploading files to Azure include:

Parallel upload support: Yes

Number of parallel threads per file: 20
Azure blob type: Block blob
Upload payload size per thread: 4 MB

Working with Google Cloud

To use OptimizeRasters with Google Cloud Storage, you will need to install the google-cloud-storage third-party Python package. See OptimizeRasters installation, above.

Managing Google credentials

To use Google Cloud Storage, you will need to generate a Google service account credential key file (profiles can't be created with the Profile Editor tool).

Note: OptimizeRasters only supports Google service accounts for validation. This is the recommended approach for authentication, since it is application-specific rather than user-specific. For more info on service accounts, see Creating a service account from Google help documentation.

Generating Google service account credential key files
Before creating a new service account key, the owner of the Google account must give the necessary permissions using the existing datastore and storage roles.

1. To generate a service account key, log in to your https://console.cloud.google.com account, select **API Manager → Credentials → Create credentials**, and then select the **service account key item** from the list. The downloaded JSON service account key file will be used to gain access to read/write to buckets.
2. Save the downloaded service account key in the root folder at **C:\Users\%username%\.optimizerasters\google**. Multiple service keys (and JSON files) can exist at this location.

Accessing public Google buckets
To access publicly available buckets, you must still authenticate with a valid service account using the following steps:

1. Create and download a separate service account key without any security roles attached.
2. Save it (alongside the other key files, if present) at the Google credential file location (**C:\Users\%username%\.OptimizeRasters\Google)**.
3. Rename the file to **public-buckets.json** so it can be easily identified.

This service account will show up in the list of profiles when using OptimizeRasters, and can be used to access any publicly available Google bucket.

Known limitations
Access Control List (ACL) settings can't be set while uploading files to Google Cloud Storage. As a result, if you're creating raster proxy files using rasters already present on Google Cloud Storage, you'll have to make those files publicly readable temporarily through your Google Cloud Platform account. To do this, check the **Share publicly** check box next to each file on cloud storage.

Configuration file parameters

The following table lists all the Google-specific configuration file parameters and acceptable values. Values added to the parameter files will be used unless they aren't overridden at the command line.

(See <u>Using OptimizeRasters: Command Line</u> for equivalent command line parameters where they exist.)

Table 7: Configuration file parameters for Google Cloud Storage

Configuration File Parameter: Google	Description
CloudDownload	Set to true if the input will be from the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
In_Cloud_Type	Specifies the cloud storage provided for the input data. Valid options are: amazon, azure, google. <u>Note</u>: For Google storage, enter **google**.
In_Google_ProfileName	Use this parameter to select a profile name for the security credentials you wish to use to access the input location. <u>Note</u>: This must match a profile name in the Google credential file.
In_Google_Bucket	The name of the bucket holding the input files. The specified input credentials must provide access to this folder.
In_Google_ParentFolder	The input folder path where the input data is located. Exclude the bucket name.
CloudUpload	Set to true if the file destination is in the cloud. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.
Out_Cloud_Type	Specifies the cloud storage provided for the output data. Valid options are: amazon, azure, google. <u>Note</u>: For Google storage, enter **google**.
Out_Google_ProfileName	Use this parameter to select a profile name for the security credentials you wish to use to access the output location. <u>Note</u>: This must match a profile name in the Google credential file.
Out_Google_Bucket	The name of the bucket that will hold the output files. The specified output credentials must provide access to this folder.
Out_Google_ParentFolder	The output folder path where the output data will be copied. Exclude the bucket name.
Configuration File Parameter: Google	Description
DeleteAfterUpload	Set to true to delete temporary rasters/files after processing is complete. Acceptable values are: true, yes, t, 1, y, false, no, f, 0, n.

Using OptimizeRasters: Sample Python scripts

OptimizeRasters can be incorporated directly into custom client code. This is likely the preferred way for developers to interact with OptimizeRasters in an automated batch processing environment (no ArcGIS required). Included with OptimizeRasters (found in the …/OptimizeRasters/CodeSamples/ directory) are three source files that provide sample scripts for common OptimizeRasters tasks: *Table 8: Sample Python scripts*

Script filename	Description
processUsingAListOfFiles.py	Demonstrates how to process rasters using a list of known files
processUsingAnInputFolder.py	Demonstrates how to process rasters using an input folder path and how to deal with errors
validatingCredentialsUsingUI.py	Demonstrates the OR validation process

To use these sample scripts, developers should note the following:

- The scripts include comments to guide developers.
- The preferred way to test these source files is to copy them to the same location as OptimizeRasters.py. The default path in the OR setup points to C:\Image_Mgmt_Workflows\OptimizeRasters.
- Temp values prefixed and suffixed with '!!' will need to be edited with actual values before the code will run successfully.

Appendix A: Working with Raster Proxy Files

Using raster proxy files with ArcGIS

What are raster proxy files?

A raster proxy file is a small file that has the same name (and valid extension) as a raster, but instead of storing the pixels, it stores only a reference to a raster (which is often located in cloud or enterprise storage). In ArcGIS ArcMap 10.4.1+, Pro 1.3+, and Server 10.4.1+, raster proxy files can be used to access raster data. The user works directly with the locally stored raster proxy files, which access the large raster data files as needed. As a result, data management is improved: the large raster datasets can be stored in slower, inexpensive storage (like cloud storage or a storage area network), while users can quickly work with and transfer the smaller raster proxies and associated auxiliary files.

ArcGIS treats raster proxy files the same as a full raster data file. When ArcGIS attempts to read the pixels, it identifies the file as a raster proxy. It then reads and provides back primary metadata including extent, number of bits, and number of bands. If pixel data is required, it accesses the slower raster data storage, reading only the necessary tiles of raster data. Initially, such access will be slower than accessing the data locally. However, there are two benefits: the data can be accessed without downloading it locally, and raster proxies have the option of caching the accessed pixels (stored as highly optimized MRF files with the extension .mrf_cache). If a cache is created, subsequent access to the same areas occurs very quickly.

As a result, raster proxies can speed up and simplify the process of managing and accessing collections of rasters, while minimizing storage costs and requirements.

Note: Raster proxies are sometimes confused with the MRF file format. While raster proxy files are generated as a part of the MRF file format, they can also be created separately to point to any raster file. They can also take any raster file extension. For example, a user might use OptimizeRasters to transfer imagery into the cloud as Cloud Optimized GeoTIFF files, and generate local raster proxy files that maintain the .tif file extension.

Typical workflow for using raster proxy files

Most commonly, raster proxies are used to create mosaic datasets from collections of rasters stored in cloud storage.

The typical workflow for using raster proxies follows three steps:

1. Use OptimizeRasters to create small raster proxies from the remotely stored raster data files and copy them to a machine used for image management.

2. Create mosaic datasets from the raster proxies using standard image management workflows. The mosaic datasets can be tested using ArcGIS desktop.

3. Serve the mosaic datasets as image services.

    Note: Prior to serving the imagery, the raster proxies (and any auxiliary files) may need to be copied to the server computer, but since these files are small the process is quick. If paths to the files change, the paths will need to be repaired in the mosaic dataset.

Access to the remotely stored raster data is an important consideration. The machine storing the raster proxies must be able to access the raster data. For data stored in secure buckets/containers in the cloud, this can be an issue. Some possible solutions include:
- Storing raster proxies on a virtual machine with access to the storage location (e.g. using an IAM role)
- Providing access to relevant IP addresses in the storage location's bucket/container policy
- Setting the raster proxy machine's environment variables for the key/secret key needed to access the storage location

Accessing Landsat with raster proxies

Raster proxies can be used to access free Landsat imagery from Amazon Web Services. Amazon stores each Landsat 8 scene on publicly accessible S3 storage, formatted as eight tiled TIFF files and an associated .met (metadata) file.

Traditionally, the user would download the files to local storage. Since one Landsat scene is about 1GB, copying full scenes is time consuming, uses significant bandwidth, and requires substantial storage.

A better option is to use OptimizeRasters to create a locally stored raster proxy for each of the TIF files, copying only the small .met file from the cloud. When ArcGIS views the local directory of raster proxies, it will treat the files as if they were a complete Landsat scene—the Landsat 8 raster product and raster types in ArcGIS will work as normal, even though the raster data remains in S3 storage.

Maximizing performance with raster proxy files

Initial access performance of raster proxies is dependent on two things:

1. **The structure of the data.** The performance of raster proxies is influenced by the structure of the source data, which should be tiled and contain pyramids. If the source data is not tiled, accessing areas of the raster at high resolution requires a large number of requests. Without pyramids, accessing a raster at low resolution requires the whole image be accessed, which is very slow. Using OptimizeRasters to create or transfer the rasters to cloud storage is the best way to ensure your rasters are tiled with pyramids.

    Additionally, although raster proxies created from tiled TIFF files work well, raster proxies created from MRF files work better for two reasons: (1) MRF files are structured to minimize the number of requests needed to extract any tile of data, and (2) MRF files offer LERC compression.

    With LERC, performance is improved because the raster proxy cache is stored using the same compression as the MRF files themselves, so no transcoding is required.

    LERC is especially well-suited for these data types:
    *   Higher bit-depth data (e.g., newer satellite imagery or elevation models)
    *   Categorical or lower-bit-depth data (e.g., classification results)
    *   8-bit/band images (e.g., orthoimagery), if lossless compression is required

    LERC also explicitly handles NoData values, which can reduce artifacts at the edges of some images.

2. **The connection between the machine and the storage system**. While a raster proxy can be used from any internet-accessible machine to access raster data stored on Amazon, for example, performance will be significantly better when accessing the raster data from an EC2 instance running in the same AWS region as the S3 storage. Once a mosaic dataset of raster proxies has been created and tested locally, the raster proxies and mosaic dataset can be copied to an EC2 instance on AWS and served as image services.

Additionally, properly managing the block size and cache will help maximize performance.

*   The block size defined in the raster proxy should be the same size as the input tile size.

*   Use a GDAL block cache that is large enough to hold all the blocks. If the remote file is pixel interleaved but the raster proxy file is band interleaved (as in the case of LERC compression), using a large enough block cache will prevent the remote page from being read and decompressed multiple times, once for each and every output band.

    Note: GDAL block cache size is set in the system environmental variables in MB (e.g. GDAL_CACHEMAX=64). In most cases the default values are sufficient, but if using data with large input tiles it may be advantageous to check this value.

Creating raster proxy files from network-attached storage

Raster proxies can be used to speed up access to slower enterprise storage area networks or networkattached storage, especially when network latency is affecting performance. Raster proxies improve performance by reducing the number of requests to the slower storage.

To accomplish this, (1) create raster proxies of the network rasters on a fast SSD or direct access drive on the server, and (2) define the cache to reside on the faster drive.

The command line for creating raster proxy files would read as follows:

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

Converting overviews to raster proxies

When transferring overviews to cloud storage, configuration files that do not create pyramids (included with OptimizeRasters) should be used.

Overviews are reduced-resolution datasets created from mosaic datasets. While pyramids are required for most rasters, they are not required for overviews, which are already created at appropriate resolutions. For more information on overviews, see http://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/mosaic-datasetoverviews.htm.

The following OptimizeRasters configuration files offer two different compression options:

> **Overviews_to_MRF_LERC**: Used with data that should be lossless or for floating point data.
> **Overviews_to_MRF_JPEG**: Used with data that can be losslessly compressed, like natural color imagery.

To create overviews for a mosaic dataset and store them in cloud storage, follow these steps:

1. Create the overviews using the ArcGIS "Define and Build Overviews" command, directing output to a local datastore.
2. Run OptimizeRasters (using one of the configuration files listed below) to transform and copy the overviews to cloud storage.
3. Repair mosaic dataset paths in ArGIS to point to the raster proxies. (See http://pro.arcgis.com/en/pro-app/help/data/imagery/repairing-paths-in-a-mosaic-dataset.htm.) The locally stored overviews are no longer required.

Serving imagery using raster proxies

To serve imagery, raster proxy files can be transferred to a server, which will then reference the original source data and create its own local data copies as needed.

If your imagery is not in a public bucket, the raster proxies must have access to the imagery storage location. Example implementations include:

- Server is on an EC2 instance with IAM role access to the S3 bucket with imagery
- S3 bucket policy includes access for the IP addresses of all federated servers
- Environment variables are set on all federated servers for the user role credentials used to access the S3 bucket

Note: To maximize performance, the server should be well-connected and located as close as possible to where the data is stored. In the Amazon Web Services cloud, for example, the user should put the server in the same region as the S3 storage.

Embedding raster proxies into mosaic datasets

If rasters are accessed using mosaic datasets, and access to the raster proxies does not require any auxiliary files, the raster proxies can be embedded directly into the mosaic dataset. Mosaic datasets with embedded raster proxies no longer reference external files, so they can be easily transferred between servers without also needing to copy additional files.

This is relevant either if the raster is a simple raster dataset, or if the raster type used to ingest the rasters into the mosaic dataset has copied the required auxiliary data.

Note: If the raster requires additional metadata, either embedded in the raster or as an .aux.xml file next to the raster, the raster proxy can't be embedded into the mosaic dataset without losing access to this metadata.

OptimizeRasters supports the creation of raster proxies as a CSV file to make it simpler to embed them in a mosaic dataset. This can be accomplished two ways:

- Using rasterproxy mode, the output location can be defined as a .csv file (not a folder location). This will generate a raster proxy table without creating additional raster datasets.
- Using mrf, tif, or tif_cog mode, the rasterproxypath can be defined as a .csv file (not a folder location). This will generate a raster proxy table at the same time raster files are being converted or transferred.

Either way, a CSV file will be generated with a header row and a single line for each raster containing the contents of the raster proxy. A mosaic dataset can take this CSV file as input, embedding the raster proxies in the mosaic dataset.

To embed the raster proxies after generating a CSV file, do the following:

1. Right click the mosaic dataset in the Catalog pane and select **Add Rasters…**.
2. For Raster Type, select **Table / Raster Catalog**
3. Click the Browse… button and select the CSV of raster proxies you created.
4. Click **Run**.

To verify that the raster proxy is embedded, do the following:

1. Open the mosaic dataset's attribute table.
2. Select an embedded raster, then click **Explore Raster Items** on the Data tab (be sure the mosaic dataset is still be selected in the Contents pane).
3. Click the **Item Functions** button, then hover the mouse over the blue input raster. Check to make sure the raster proxy string has been embedded properly.

It is also possible to embed raster proxies using a Python raster type (available with ArcMap 10.5+ and ArcGIS Pro 1.4+), which is a specialized script that copies the raster proxy into the items of the mosaic dataset.

Cache management

ArcGIS will only add to the cache as required; it will not automatically clear the cache. The cache for raster proxies is not managed because the appropriate amount of time to keep a cache is dependent on both the application and the size of the drive available. As a result, the user should follow the best practices outlined here to manage the cache.

Defining a cache directory

The user should define a directory specifically for storing cache created when raster proxies are accessed. It is recommended that z:\mrfcache\ is used as a standard. Although the cache for raster proxies can be stored in the same location as the raster proxies, defining a dedicated directory for the cache (ideally set as fast, accessible disk) makes cache management simpler.

When using Amazon Elastic Compute Cloud (EC2) or similar infrastructure, ephemeral disks should be used for the cache. Ephemeral disks are fast SSD drives that are directly connected to the machines. This means they are not limited by network bandwidths and can be easily assigned as the Z drive.

If a Z:\ drive is unavailable, create a virtual Z:\ drive that maps to any user-specified drive with these steps:

1. Share a folder with all users granting full permission to write to the directory.
2. In Windows Explorer, **click** on the dropdown menu on the home tab and choose "**map as drive**."
3. In the dialog box, select "**Z:**" as the **drive**, and enter the directory path for the **folder type**.

Alternatively, a command similar to the following can be used at the Windows Command Prompt: subst

    z: c:\temp

The cache should follow the directory structure and naming of the source data. Since cache files have the same name as the source raster, this will prevent errors resulting from duplicate file names.

Caches can only be safely shared by multiple processes (ArcSOC) on the same machine on a local drive. If you're using a non-local drive or different machines, cache corruption is likely.

How to stop caching

The user may want to stop caching temporarily under some circumstances. To stop caching temporarily, set the environment variable **MRF_BYPASSCACHING** to **TRUE**. This variable can also be set as a GDAL configuration option.

Following are three example scenarios when a user would turn off caching:

- Creating a high resolution tile cache of a mosaic datasets (for example, persisting a mosaic dataset of orthoimagery as a basemap). This will typically require all data in a mosaic dataset to be read only once. As a result, caching has little value, but could cause an overflow of the cache storage.
- Running a process that results in all rasters being read once (for example, generating statistics using a skip factor of one).
- Running types of raster analytics that access all the rasters once (for example, running segmentation on a large collection of rasters using raster analytics).

Note: All raster proxy files opened while this variable is set to true are affected. Additionally, data that has been cached already can still be read, making this suitable for some partially connected applications.

Working with CleanMRFCache to clear the cache

The user must periodically delete cached files. It is important that the files in the cache directories are deleted periodically, or when additional space is required. Care should be taken to ensure they are properly managed, since a full disk can lead to errors. The CleanMRFCache.py tool can help simplify this process (see Working with CleanMRF, below).

Cache created by proxy rasters will have the extension .mrfcache. These files can be deleted at any time and will not influence the running of ArcGIS.

Note: If a cache file is actively written in parallel to a request to delete it, then a file access error will result and that file should not be deleted.

The CleanMRFCache.py tool simplifies the process of clearing the cache. CleanMRFCache is a simple cache clean up tool that clears the cache based on amount of memory required. The CleanMRF script should be scheduled to run on a regular basis using the Windows task scheduler.

To run CleanMRFCache, the command line would read as follows:

<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>

Example commands:

C:\"Program Files"\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\python.exe C:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache ext=txt,mrf_cache -size=1

C:\"Program Files"\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\python.exe C:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache Optional arguments include:

Table 9: Optional arguments for the CleanMRFCache tool

Parameter	Description
-mode=	The mode used to clean up the cache. Options include: scan: (default) displays files found, their sizes, and last access date, starting with least-accessed files first. del: Deletes files until the target -size has been reached.
-ext=	Extension of files to delete. By default, this is mrf_cache.
-size=	Size in Gigabytes to remain on the disk. The default is 2 GB.

Cache management summary

- ArcGIS will not automatically clear the cache.
- The user should define a directory specifically for cache (**z:\mrfcache** is recommended).
- The cache should follow the directory structure and naming of the source data.
- Caching can be suspended temporarily, if needed.
- The user must periodically delete cached files.
- CleanMRF is a tool provided with OptimizeRasters to help simplify clearing your cache.

Appendix B: Additional OptimizeRasters GP Tools

Two additional tools are included with OptimizeRasters: The Profile Editor tool and the Resume Jobs tool.

To access the OptimizeRasters Python toolbox, follow these steps:

1. Start ArcGIS Pro and create a new project.

2. In the Catalog Pane, right click Folders and select **Add Folder Connection**. Navigate to the OptimizeRasters directory (C:\Image_Mgmt_Workflows\OptimizeRasters) and click **OK**.

3. Open the toolbox by clicking the **+ sign** next to the OptimizeRasters directory and OptimizeRasters.pyt (see Figure 1).

Profile Editor Tool

The Profile Editor tool allows you to store and edit credential profiles for Amazon S3 or Microsoft Azure cloud storage.

To use OptimizeRasters with either Amazon S3 or Azure Blob, you will need specific credentials to read or write to storage. The Profile Editor tool stores access keys and secret keys for any number of S3 or Azure profiles. Based on the profile selected, the stored keys are then used by OptimizeRasters to access protected cloud storage.

Profiles to use with Google Cloud Storage accounts can't be created with the Profile Editor tool. See Working with Google Cloud, above.

Note: When accessing S3 via an EC2 instance with IAM role access, these credentials are not required. Instead, you'll either check the Secured Bucket checkbox in the OptimizeRasters tool dialog, or set the command line flag -usetoken=true.

How to add a new profile

1. Double-click the Profile Editor in the OptimizeRasters Toolbox.

2. Select a **profile type** (Amazon S3 or Microsoft Azure) in the Profile Editor dialog (Figure 4).

3. Enter the **profile name**, **Access Key ID**, and **Secret Access Key**.

4. (Optional) Enter a custom **Endpoint URL** (the entry point for the data access storage), if different from the default endpoints:

   Microsoft Azure: http://blob.core.windows.net
   Amazon S3:   http://s3.amazonaws.com
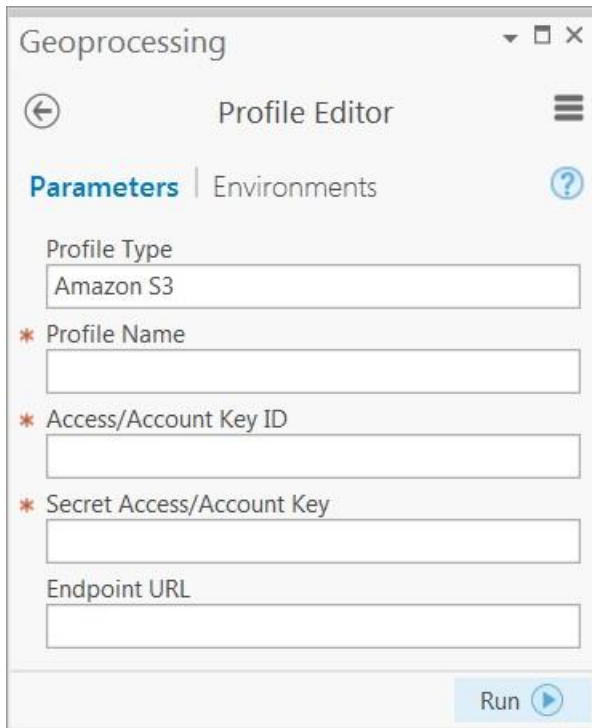
5. Click **OK** to save.

Figure 4: Profile Editor dialog

How to edit an existing profile

1. Double-click the **Profile Editor** in the OptimizeRasters Toolbox.

2. Select the **Profile Type** and enter the existing **Profile Name** you wish to edit.

3. Enter the new **Access ID** and **Secret Access Key**

4. (Optional) Enter a custom **Endpoint URL** (the entry point for the data access storage), if different from the default endpoints:

   Microsoft Azure: http://blob.core.windows.net
   Amazon S3:   http://s3.amazonaws.com

5. (Optional) Select the **Editor Option** '**Overwrite Existing**,' then click **OK**.

How to delete an existing profile

6. Double-click the **Profile Editor** in the OptimizeRasters Toolbox.

7. Select the **profile type** and enter the **profile name** you wish to delete.

8. Select the **Editor Option** '**Delete Existing**,' then click **OK**.

Resume Jobs Tool

The Resume Jobs tool is used to complete workflows that were accidentally stopped or failed to complete. Running the Resume Jobs tool will show a list of pending jobs, and allow you to resume any failed workflows.

How to resume unfinished jobs

1. Double-click Resume Jobs in the OptimizeRasters Toolbox to see a dropdown list of pending jobs (Figure 5).

2. To resume a pending job, select a job from the list and click **OK**.

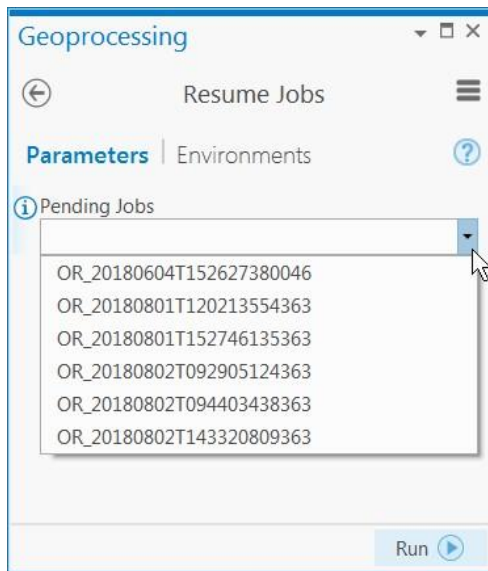   Note: Completed jobs will no longer show up in this list.

Figure 5: Resume Jobs dialog

Working with the Resume Jobs tool

When moving large collections of imagery to cloud storage, it is common for processing to be interrupted. To make it easier to complete interrupted transfers, OptimizeRasters automatically creates a job file each time it's run.

A job file is a text file with a header indicating the processes performed, followed by a list of the files remaining to be processed. The default job filename format is OR_YYYYMMDDTHHMMSSssssss.orjob, indicating the date the job was initiated.

Note: It's also possible to name the job files using the command-line flag -job. The .orjob extension will be added if omitted.

If an error occurs during a workflow:

OptimizeRasters will automatically retry processing the problem raster file. If unsuccessful, the issue will be flagged in the job file. Before completing the process entirely, OptimizeRasters will again attempt to process any flagged files. If the issue is still unresolved, the failed raster file will remain listed in the job file. After running OptimizeRasters, the resulting job file will list any files that were not fully processed.

Users can resume failed workflows using the Resume Jobs geoprocessing tool or by using OptimizeRasters with the -input command line flag pointing at the job file, i.e.:

<path to python.exe> <path to OptimizeRasters> -input=OR_20151211T024329630000.orjob

The user can manually inspect the job file to see the processing stage at which each raster file failed.

If a workflow is processed successfully:

Once a workflow is processed successfully, the job file will be archived in the Job folder; if there are errors, the file will remain at the OptimizeRasters.py location.

Example OptimizeRasters job files:

1. **Job file for working with user-defined Sentinel HTTP raster URL entries**

   \# input=http://sentinel-s2-l1c.s3.amazonaws.com/
   \# pyramids=false
   \# config=C:/Image_Mgmt_Workflows/OptimizeRasters/Templates/Sentinel2_to_MRF.xml #
   mode=mrf
   \# output=C:/processed/files
   SOURCE   COPIED PROCESSED   UPLOADED **http://sentinel-s2-l1c.s3.amazonaws.com**/tiles/10/R/EV/2016/1/13/0/B01.jp2 **http://sentinel-s2-l1c.s3.amazonaws.com**/tiles/10/R/EV/2016/1/13/0/B02.jp2

2. **Job file with entries pointing at a local resource**
   <u>Note</u>: The listed file below that starts with "##" is commented out and will not be processed.

   \# input=C:/your_raster/files/path
   \# config=C:/Image_Mgmt_Workflows/OptimizeRasters/Templates/Imagery_to_TIF_JPEG.xml #
   output=C:/processed/files
   SOURCE   COPIED PROCESSED   UPLOADED
   C:/your_raster/files/path/metadata.txt
   C:/your_raster/files/path/bahrain.TIF
   C:/your_raster/files/path/subfolder0/earth.TIF
   ## C:/your_raster/files/path/subfolder0/subfolder1/water.tif
   C:/your_raster/files/path/subfolder1/fire.TIF

3. **Job file for processing rasters in an S3 bucket and uploading the processed files to Azure storage**

   \# input=your_raster/files/path
   \# inputbucket=your_s3_bucket_name
   \# clouddownload=true
   \# clouddownloadtype=amazon
   \# inputprofile = amazon_credential_profile_name
   \# config=C:/Image_Mgmt_Workflows/OptimizeRasters/Templates/Imagery_to_TIF_JPEG.xml
   \# output=your_processed_amazon/files/on/azure
   \# tempoutput=C:/temp/storage/to/store/before/pushed/to_azure
   \# cloudupload=true
   \# clouduploadtype=azure
   \# outputprofile=azure_credential_profile_name
   \# outputbucket=your_azure_container_or_bucket_name SOURCE
   COPIED PROCESSED   UPLOADED
   your_raster/files/path/world.tif
   your_raster/files/path/subfolder0/air.tif

Appendix C: Using Obfuscation for Access Control in the Cloud

OptimizeRasters' -haskkey flag helps with file obfuscation, which is a simpler alternative to common access control methods used in cloud environments.

OptimizeRasters has a feature to help obfuscate directories by optionally appending an obfuscation key to a directory name for the output imagery. The obfuscation key is generated using a hash of the original directory and a hashkey, so the same key can be regenerated if additional data is added to the directory.

Frequently, a user may need to make imagery accessible to only a select group. Two common options for controlling access include ACL (Access Control List) and VPC (Virtual Private Cloud). However, these and other common access control options require the maintenance of multiple user names and passwords, and the additional security overhead can slow down access speed.

A simpler method is file obfuscation. File obfuscation locates files in public buckets that can be accessed by anyone, but (1) the names of the files are obfuscated so that it is not possible to guess the contents and (2) the bucket policy is set up so that users cannot query or get listings of the bucket content. Only users who know the URLs of the files can access them or share them (by email, for example).

The largest limitation to the file obfuscation method is that once a URL has been shared, access cannot be easily revoked for a single user. The file can be deleted from the bucket, but then all users lose access. However, even using secure storage, once a user has downloaded a file, access to the original file is irrelevant. As a result, sharing the short URL to the raster should be considered equivalent to sharing the data files. Ultimately, though, obfuscation usually provides sufficient access security for large datasets, with advantages in terms of simplicity, performance, and interoperability.

Using the -hashkey flag

The command line flag -hashkey can be set to a value that will be used to generate the hash output text. You can use your own custom key as hash text; you can insert that text at any folder position; or you can use randomly generated hash text:

Table 40: Hashkey flag syntax

Usage	Description
-hashkey=secret_key	Generates the hash text in the output path based on your <secret_key>.

Using your own private key is useful if you or anyone on your team will add data to the output cloud path in the future. Additionally, using the same <secret_key> will make sure data will be synced properly with existing folders and avoid creating new output folder hash paths each time OptimizeRasters is run.

Table 51: Hashkey flag syntax for custom hash positioning

Usage	Description
-hashkey=secret_key@2	The @ sign followed by a numeric value can be used to position the generated hash text in the output cloud path.

A. If the @ sign is omitted, or the number specified is less than 2, OptimizeRasters defaults to inserting the hash text as the second subfolder from the left:

Example 1:   \ParentFolder**RFd8GFf5_@**\Scene1\a.tif

B. If the specified value is larger than the maximum number of subfolders in the user-entered output path, OptimizeRasters automatically inserts the hash text just above the base filename: <u>Example 2</u>: \ParentFolder\Scene1**RFd8GFf5_@**\a.tif

<u>Note</u>: While Example 2 is also equivalent to -hashkey=secret_key@3, entering a very large number after the @ sign (e.g. -hashkey=secret_key@10000) will ensure that OptimizeRasters always inserts the hash text just above the base filename.

Using the # flag is useful if the user simply wants to hash out the output folder but still ensure that the output folder structure isn't easy to guess by anyone with access to the data.

Table 62: Syntax for hashing an individual output folder

Usage	Description
-hashkey=#	Creates random hash text for any individual cloud output folder.

Example usage

To help visualize hash text in cloud output folders, the same files are shown with and without obfuscation.

Without -hashkey enabled:

> \ParentFolder\Scene1\a.tif
> \ParentFolder\Scene1\b.tif
> \ParentFolder\Scene2\c.tif

With -hashkey set at the command line:

> \ParentFolder**RFd8GFf5_@**Scene1\a.tif
> \ParentFolder**RFd8GFf5_@**\Scene1\b.tif
> \ParentFolder**fgEfQRrq_@**\Scene2\c.tif

<u>Note</u>: The hashed folders above for the files a.tif and b.tif share the same hash text in the output folder path because their original folder path was the same.

Obfuscation with raster proxy files

Raster proxy files created during a file conversion won't include the hash keys. If the raster proxy files are created separately, using the rasterproxy mode, the hash key will be placed on the local disk.

Obfuscation highlights

- -hashkey is an optional flag at the command line.
- -hashkey is only effective if clubbed together with the -cloudupload=true.
- The output hash is always suffixed with '_@' for easy identification and easy removal, if needed.
- The generated hash text is 10 characters long, including the hash suffix.
- The hashing algorithm uses MD5 one-way hashing.
- The same -hashkey will result in the same hash text for the specific output path in order to simplify adding additional files to an existing folder at a later time.
- -hashkey usage does not affect -cache and -rasterproxy folder paths entered at the command line.

Appendix D: Working with ArcGIS Server and ArcGIS Desktop

The raster proxy caching support is designed to allow multiple processes to write to the same cache simultaneously without corrupting the cache. However, when using ArcGIS Desktop and Server on the same machine, some special considerations are required. If ArcGIS Server creates a file, ArcGIS Desktop (which is logged in as a different user) is not automatically able to read the file.

To resolve this problem, be sure to set permissions so that all users have read and write permission to the folder where the cache will be stored.

Appendix E: Common Raster File Formats

Below is a breakdown of common raster file formats referenced in this document.

TIFF (or GeoTIFF)	- Very popular format for imagery and rasters - Supports different bit depths and numbers of bands - Includes additional metadata in tags internal to the file - TIFF files from data providers are often in the simplest form and inefficient to access - Can include georeferencing information embedded as tags (sometimes called GeoTIFF)
Tiled TIFF	- Type of TIFF or GeoTIFF - Pixels are structured into tiles to optimize access, especially for large files. This minimizes the number of disk access requests to get a subset of pixels. - Tiling is done by including an index to the tiles as part of the metadata tags. - Optional JPEG or LZW/Deflate compression can reduce file sizes - Optional pyramids (sometimes referred to as reduced resolution datasets or overviews) increase access efficiency at smaller scales. These pyramids increase the file size by between 30% - 50% depending on the compression and type of data.
Cloud-Optimized GeoTIFF	- Two differences from tiled TIFF: Pyramids are required, and the index and pyramids are moved to the beginning of the file. - This file restructuring can provide a slight performance improvement in applications that only view the image at small scales or need to crawl for the metadata. - Creating COG files takes longer than tiled TIFF because the pyramids and tags are moved to the start of the files. - In ArcGIS Pro, performance improvements are negligible compared to tiled TIFF.