OptimizeRasters

Copyright © 1995–2017 Esri.
All rights reserved.

User Documentation

Updated: April 20, 2017

Disclaimer: Applicable for OptimizeRasters Version 20161218 (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. Convert rasters from a variety of file formats to optimized Tiled TIF or MRF formats.
2. Transfer data to and from 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

What do you get?

This 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 at a later time.

What are the system requirements?

- Python 2.7 or greater (installed with ArcMap 10.4 and ArcGIS Pro 1.3).
- The OptimizeRasters geoprocessing toolbox requires ArcMap 10.4.1 or ArcGIS Pro 1.3 or higher.
- OptimizeRasters can be run from the command line even if ArcGIS is not installed.
- There are OptimizeRasters versions for both Windows and Linux.
- OptimizeRasters can be used with Amazon Web Services Lambda serverless compute service (with some restrictions on data size).

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 command line 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 Tiled TIF or MRF files, 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 Tiled TIF or MRF formats. The result is more efficient, scalable, and elastic data access with a lower storage cost.

File conversion to Tiled TIF or MRF 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 compression, which further reduces the amount of data stored and transmitted.

Files can be converted to either Tiled TIF or MRF format:

Tiled TIF
- Even popular TIF raster products, like those from DigitalGlobe and USGS, 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.

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.

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 both Amazon S3 and Microsoft Azure cloud storage services.

See Appendices B and C for more information on Amazon S3 and Microsoft Azure cloud storage.

Creating Raster Proxies to simplify data management

OptimizeRasters can also generate Raster Proxies to simplify access to raster data stored on cloud or network storage.

Raster Proxies are small files stored on local file systems that reference much larger raster data files stored remotely. A user can work efficiently with collections of small Raster Proxy files, which are accessed by ArcGIS like conventional raster files. At the same time, the application can 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.

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 the GDAL library. As a result, users can adapt and expand the code as new needs arise.
- The GDAL library enables OptimizeRasters to handle a wide variety of raster file formats.

Common OptimizeRasters Workflows

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

Copying raster data to enterprise storage

Example scenario 1: A satellite imagery vendor delivers imagery as TIF files, along with auxiliary metadata files, that are not tiled and do not have pyramids.

Workflow: Use OptimizeRasters to copy all raster files from one directory to another (e.g. from an external hard disk to your organization's shared file storage). During this process, OptimizeRasters will convert TIF files into Tiled TIF files with internal pyramids. The conversion is lossless and the filenames are unchanged, but the TIF files 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.

Example scenario 2: The 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 using Limited Error Raster Compression (LERC), which can be lossless or controlled lossy.

Optimizing raster data in enterprise storage

Example scenario 3: A vendor delivers the source raster data in a slow-to-read JP2 format.

Workflow: Use OptimizeRasters to copy all raster files from one directory to another, while converting the format to Tiled TIF (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.

Copying raster data to cloud storage

Example scenario 4: Your organization wants to take advantage of the flexible computing power and reduced cost of cloud-based storage.

Workflow: Use OptimizeRasters to copy 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 Tiled TIF) during the transfer (see example scenario 6).

Accessing raster data in cloud storage

Example scenario 5: Your organization needs to access large collections of rasters stored in the cloud.

Workflow: Use OptimizeRasters to improve efficiency by creating Raster Proxy files. Raster Proxies can be used as 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 that raster data 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. Files stored as Tiled TIF are also quite efficient.

Optimizing raster data in cloud storage

Example scenario 6: 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 TIF in the cloud. 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.

Accessing Landsat 8 data from Amazon Public Data Sets

Example scenario 7: You wish to use ArcGIS Desktop (ArcMap 10.4 or Pro 1.3) to access free Landsat 8 data from Amazon, but you do not want to download the full dataset.

Note: For more information about this dataset, see http://aws.amazon.com/public-data-sets/landsat.

Workflow: Use OptimizeRasters to create Raster Proxy files to access Landsat 8 data in ArcGIS. A configuration file (Landsat8_RasterProxy.xml) with all required parameters is provided with OptimizeRasters. The user must provide the input path, the Landsat path/row, and the scene ID of the desired Landsat 8 scene.

The resulting output directory will contain the Landsat scene as eight Raster Proxy TIF files (and a corresponding .met file), which will then reference the source TIF files 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:\Python27\ArcGIS10.4\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters\OptimizeRasters.py -config=c:\Image_Mgmt_Workflows\OptimizeRasters\Templates\Landsat8_RasterProxy.xml -input=L8/160/043/LC81600432015109LGN00 -clouddownload=true -inputbucket=landsat-pds -clouddownloadtype=amazon -output=c:\temp\landsatpdsdata\L8\160\043\LC81600432015109LGN00

Installing OptimizeRasters

GUI installation

1. Ensure **Python 2.7 or greater** is installed. If ArcGIS Desktop or Server has been installed, Python was included. If not, install Python from https://www.python.org/downloads/release/python-2712/.
2. 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.
3. **Double click** the downloaded OptimizeRastersToolsSetup.exe file and **step through the wizard** to install.

   Notes:

   - 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.
   - If you have difficulty, right-click the OptimizeRastersToolSetup.exe file and select "**Run compatibility troubleshooter**."

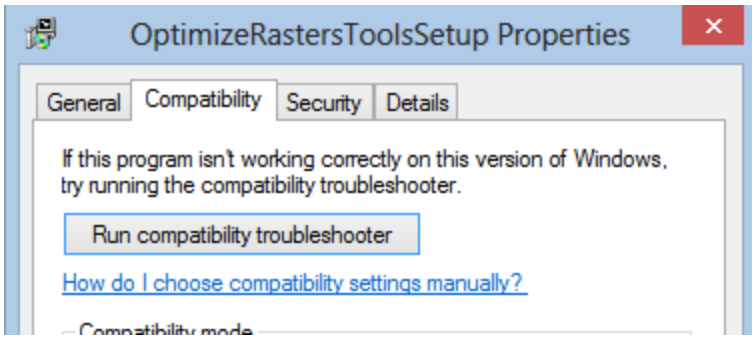

Figure 1: Compatibility Troubleshooter

Installing third-party packages for Python

If you are using OptimizeRasters to read or write from either Amazon S3 or Microsoft Azure cloud storage, you will need to install additional packages for Python. PIP is a tool used to install the boto and Azure Python packages, which are required to upload to Amazon S3 or Microsoft Azure, respectively.

Note: If Azure or boto is already installed, it is important to ensure you are running the most up-to-date version. To update these packages, open a **Command Prompt** and run the relevant command:

To upgrade Azure:	**pip.exe install --upgrade azure**
To upgrade boto (for Amazon S3):	**pip.exe install --upgrade boto**

Installing PIP (helpful for installing boto and Azure)

1. Open https://pip.pypa.io/en/latest/installing/ in a browser.
2. To download the **PIP installation** file, look under the heading **Installing with get-pip.py**, right click on **get-pip.py**, select **Save Link As…**, **navigate** to the folder you wish to use, and select **Save**.
3. Open a **Command Prompt**. Using full file locations, run the command "**python get-pip.py**"
   EXAMPLE: C:/Python27/ArcGIS10.4/python.exe C:/temp/download/get-pip.py

Installing boto (required to use Amazon S3)

1. Open a C**ommand Prompt**. Navigate to the **Scripts** folder in the Python folder.
   EXAMPLE: cd C:/Python27/ArcGIS10.4/Scripts
2. Type the command, "**pip install boto**"

Installing Azure (required to use Microsoft Azure)

1. Open a **Command Prompt**. Navigate to the **Scripts** folder in the Python folder.
   EXAMPLE: cd C:/Python27/ArcGIS10.4/Scripts
2. Type the command, "**pip install azure**"

Using OptimizeRasters: Geoprocessing Toolbox

OptimizeRasters comes with three geoprocessing tools:

- The **OptimizeRasters tool** allows you to convert most raster formats to either Tiled TIF 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).
- The **Profile Editor tool** allows you to store credential profiles for Amazon S3 or Microsoft Azure cloud storage. These simplify 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.

Note: For information about using these tools from the command line, see Using OptimizeRasters: Command Line below.

To get started using any of these tools, follow these steps:

1. Start **ArcMap**.
2. In the **Catalog** window, browse to the folder where you installed OptimizeRasters. The default location is **C:\Image_Mgmt_Workflows\OptimizeRasters**.
3. Open the toolbox by clicking the **+ sign** next to OptimizeRasters.pyt.

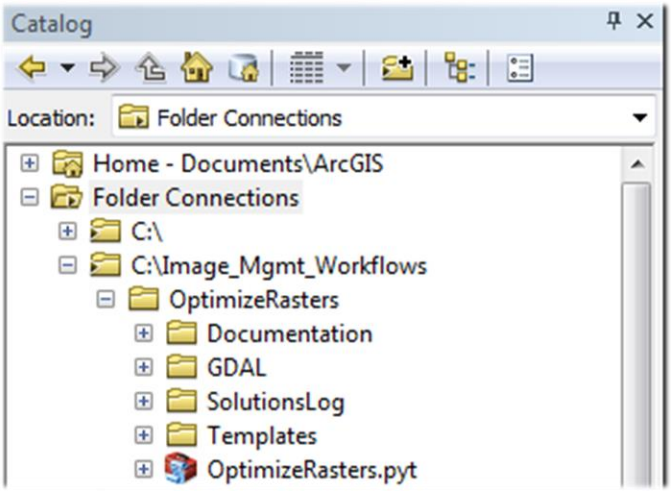

Figure 2: OptimizeRasters Catalog view

OptimizeRasters Tool

The OptimizeRasters tool allows you to convert most raster formats to either Tiled TIF or MRF file formats, transfer data to and from the cloud, and create Raster Proxies.

How to use OptimizeRasters

1. If needed, create the required **Amazon S3 or Microsoft Azure profiles** using the Profile Editor tool (see Profile Editor Tool, below).
2. Double-click to open the **OptimizeRasters tool** from the OptimizeRasters Toolbox.
3. Complete the **OptimizeRasters dialog** (Figure 3) to perform jobs. A brief description of each parameter can be found in Table 1.

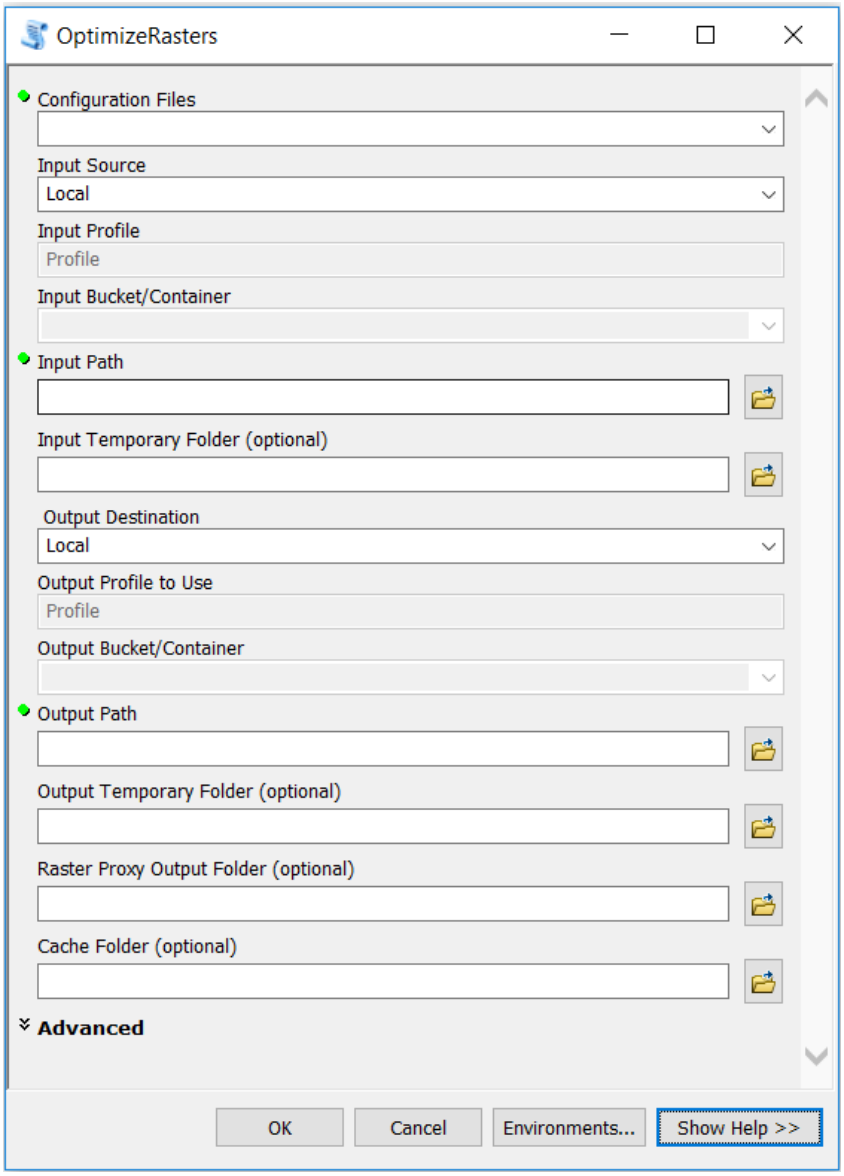

Figure 3: OptimizeRasters dialog

Table 1: Description of OptimizeRasters dialog options

Dialog parameter	Description
Configuration Files	(**Required**) These preset templates are used to drive the data conversion process. Use the pull down list to select a template that suits your needs. See Default configuration files, below, for a more detailed description.
Input Source	(**Required**) The input storage type. Options include: Local Disk: If input is from a local or network file system Cloud Storage: If input is from AWS S3 or Azure Blob storage
Input Profile	(Input from secured cloud storage only) The cloud storage user profile for your input data.
Input Bucket / Container	(Input from cloud storage only) The name of the Amazon S3 bucket or Azure container holding the input files. 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 of the form C:\MyData\Samples. For input from cloud storage, this will be the path without the bucket name or container name. See Accessing Cloud Storage, below. Note: The browse button will only work with local storage.
Input Temporary Folder	(Optional) A temporary folder used to 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 Disk: If output should be to a local or network file system Cloud Storage: If output is to AWS S3 or Azure Blob storage
Output Profile	(Output to secured cloud storage only) The cloud storage user profile for your output data.
Output Bucket/Container	(Output to cloud storage only) The name of the Amazon S3 bucket or Azure container holding the output files. 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 of the form C:\MyData\Samples. For output to cloud storage, this will be the path without the bucket name or container name. See Accessing Cloud Storage, below. Note: The browse button will only work with local storage.
Output Temporary Folder	(Optional) A temporary folder used to hold output files while uploading to cloud storage. The output will be first written to this directory and then transferred to the cloud (or slower enterprise storage). Notes: • Recommended if storage space is restricted in the default system temp location.

Dialog parameter	Description
	• Recommended 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) An output folder where cache files for Raster Proxies will be stored, which are created locally at the same time data is transferred into the cloud. Note: Option is disabled if the MRF mode is rasterproxy.
Cache Folder	(Optional) An output folder for cache files used with Raster Proxies. Default is z:\mrfcache\. See Appendix A: Working with Raster Proxy Files: Cache Management, below, for more information. Note: If left blank, caches of the Raster Proxies 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 **bucket/container name** is: landsat-pds
and the **input path** is: L8/139/045/LC81390452014295LGN00

Then the **complete paths** would be:
S3: s3://landsat-pds/L8/139/045/LC81390452014295LGN00/
Azure: http://landsat-pds.s3.amazonaws.com/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/en-us/rest/api/storageservices/fileservices/naming-and-referencing-containers--blobs--and-metadata#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 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 Command line arguments and Configuration file parameters, below, 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.
CreateRasterProxy	Creates Raster Proxy files from various raster file formats.
Landsat8_RasterProxy	Creates Raster Proxy files from Landsat 8 data stored in Amazon S3 Public Data Sets.
DG_SatelliteProduct_to_MRF_LERC	Converts Digital Globe Satellite imagery to MRF using LERC compression.
Imagery_to_MRF_JPEG	Converts imagery to MRF using JPEG compression.
Imagery_to_MRF_LERC	Converts imagery to MRF using LERC compression.
Imagery_to_TIF_JPEG	Converts imagery to TIF using JPEG compression.
Imagery_to_TIF_LZW	Converts imagery to TIF using LZW compression.
Landsat_to_MRF_LERC	Converts Landsat imagery to MRF using LERC compression.
Overviews_to_MRF_LERC	Converts overview imagery to MRF using LERC compression.
Overviews_to_MRF_JPEG	Converts overview imagery to MRF using JPEG compression.
Sentinel2_to_MRF	Converts Sentinel 2 imagery to MRF using LERC compression.
CopyFilesOnly	Copies between different storage systems (S3, Azure, and local storage) with no file conversion.

Advanced options

Advanced options in the OptimizeRasters dialog are for users who understand the OptimizeRasters parameters well enough to manually edit them.

Generally, the user will start with one of the predefined configuration files, edit parameters as necessary, and then save the selected configuration settings as a new configuration file. To edit configuration file parameters, complete the following steps:

1. Confirm you have selected the configuration file you wish to modify in the **Standard Options** section.
2. To edit values, 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.

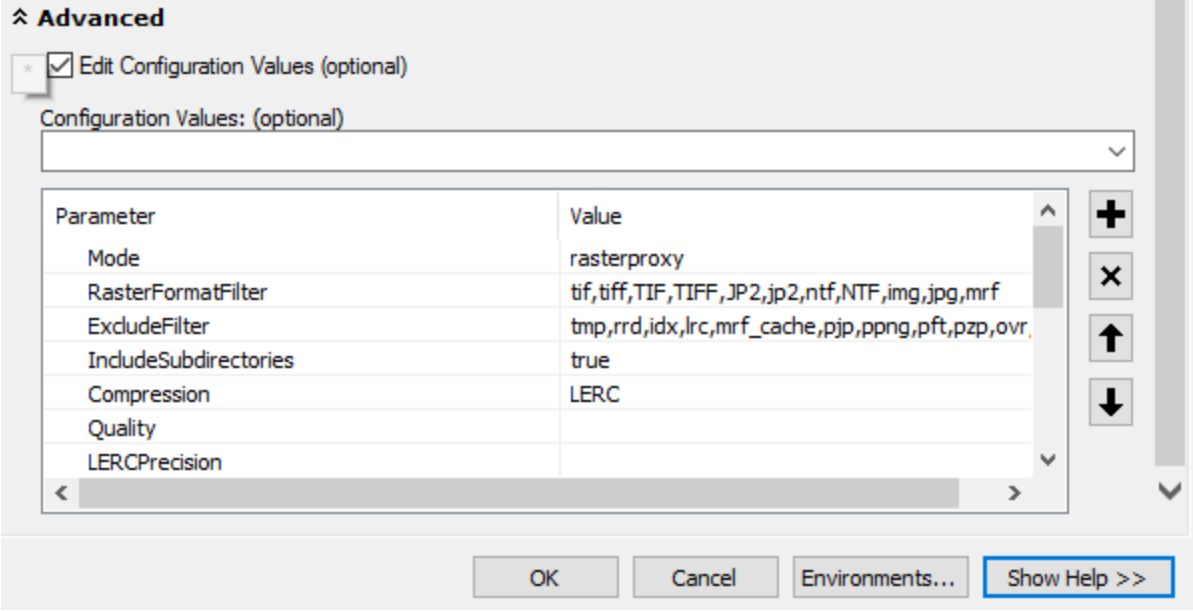

Figure 4: OptimizeRasters advanced options

3. After editing the desired parameters, click **OK** 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.

Profile Editor Tool

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

Both Amazon S3 and Azure Blob can be set up to require 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.

Note: When using storage that is public or using VPC on AWS, these credentials are not required.

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 5).
3. Enter the **profile nam**e, **Access ID**, and **Secret Access Key**.
4. Click **OK** to save.

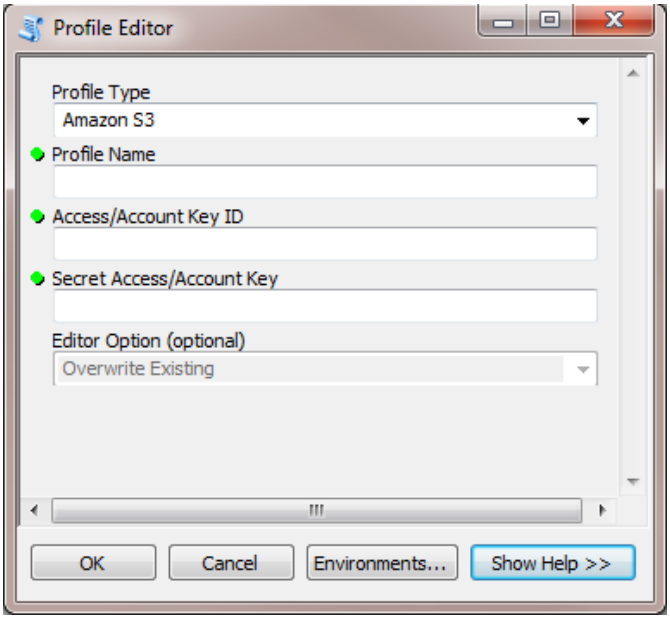

Figure 5: Profile Editor dialog

How to edit an existing profile

1. Double-click the **Profile Editor** in the OptimizeRasters Toolbox.
2. Select the **profile type** and enter the **profile name** to edit or delete.
3. Enter the new **Access ID** and **Secret Access Key**
4. Select the **Editor Option** '**Overwrite Existing**,' then click **OK**.

How to delete an existing profile

5. Double-click the **Profile Editor** in the OptimizeRasters Toolbox.
6. Select the **profile type** and enter the **profile name** to edit or delete.
7. 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. For more information on how this tool works, see Working with the Resume Jobs Tool, below.

How to resume unfinished jobs

1. Double-click **Resume Jobs** in the OptimizeRasters Toolbox to see a dropdown list of pending jobs (Figure 6).
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.

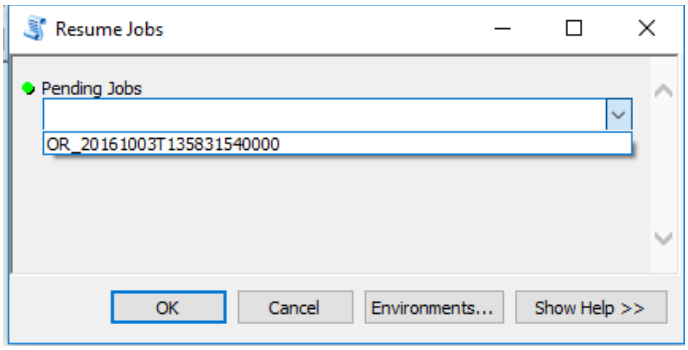

Figure 6: Resume Jobs dialog

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, 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 along with other parameters required to process a workflow. 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 still has not been resolved, the failed 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 the -input command line flag pointing at the job file, i.e.:

```
c:\Python27\ArcGIS10.4\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters -input=OR_20151211T024329630000.orjob
```

The user may manually inspect the job file to see raster files that have reported errors and the processing stage at which each 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.

Working with OptimizeRasters job files

While not recommended, OptimizeRasters job files can be created or edited manually using the following guidelines and examples.

OptimizeRasters job file guidelines:

- An OptimizeRasters job file is a text file with the extension '.orjob' that can be edited via a standard text editor.

- Job files must begin with valid header entries, which are key-value pairs prefixed with the '#' symbol, separated by line breaks, and grouped together at the beginning of the job file.
- Any valid command line argument can be a valid header key (leave off the prefix '**-**').
- [*Optional*] After the header has been defined, status column names [SOURCE, COPIED, PROCESSED, UPLOADED] can be defined separated by the (TAB) character on a single line. If omitted, OptimizeRasters will add the column header.
- The line with the status column names or the last key-value header pair is considered the end of the header section.
- After the header, a list of user-defined raster file paths or URLs can be added, one per line, all beginning with the 'input raster files' directory.
- Empty lines and lines beginning with '**##**' (i.e. comments) are ignored.

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**
   Note: 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
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
# cloduploadtype=azure
# outputprofile=azure_credential_profile_name
# outputbucket=your_azure_container_or_bucket_name
your_raster/files/path/world.tif
your_raster/files/path/subfolder0/air.tif
```

Using OptimizeRasters: Command Line

OptimizeRasters, Profile Editor, and Resume Jobs can be called as command line tools, even if ArcMap is not installed. (See also Working with the Resume Jobs tool, above.)

Standard command line usage

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

Command line help

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

Command line arguments

The following parameters and switches can be used with OptimizeRasters. They can be entered at the command line, or they can be specified in the configuration file (See Configuration file parameters, below.)

Table 3: 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 output directory.
-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=	Processing mode/output format. Typical values are tif, mrf, rasterproxy. Note: Cachingmrf and clonemrf modes have been phased out, but are supported for backwards compatibility.
-cache=	Path to cache output directory, where cache files for Raster Proxies will be stored. Default is z:\mrfcache\. See Appendix A: Working with Raster Proxy Files: Cache Management, below, for more information. Note: If left blank, caches of the Raster Proxies will be stored in the same directory as the Raster Proxy files. A separate directory (z:\mrfcache\) is recommended to make it easier to empty the cache periodically.

Command Line Argument	Description
-quality=	JPEG quality, if the compression is JPEG.
-prec=	LERC precision (the maximum amount that a value may change from its original value). The default value is 0.001, which is lossless for integers.
-pyramids=	Flag to generate pyramids. Enter true or false. 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. Values include: Noconvert: will copy raster files between different storage types without performing any file conversion Upload: will back up a local -input=C:\folder to S3, Azure, or local storage.
-subs=	Flag to let the program know whether or not to include subdirectories. Enter true or false. If true, all subdirectories of the input directory will also be processed.
-tempinput=	Path to a directory where input rasters will be copied temporarily before processing. Note: If the input is from Amazon S3 then this is a required parameter.
-tempoutput=	Path to a directory where converted rasters will be stored temporarily before moving them to output. Ideally, this will be on a faster drive as pyramids will be generated by reading and writing to this location. Note: This is a required parameter if the output will be stored on AWS S3 or Azure Blob.
-clouddownload=	Flag to let the program know whether the input location is cloud storage or not. Enter true or false.
-cloudupload=	Flag to let the program know whether the output location is cloud storage or not. Enter true or false.
-cloduploadtype=	Parameter linked to -cloudupload. If you are uploading to the cloud, enter Amazon or Azure.
-clouddownloadtype=	Parameter linked to -clouddownload. If you are downloading from the cloud, enter Amazon or Azure.
-inputprofile=	Cloud profile name that corresponds with the input cloud storage. The profile name will be used to pick up the relevant credentials.
-outputprofile=	Cloud profile name that corresponds with the output cloud storage. The profile name will be used to pick up the relevant credentials.
-inputbucket=	Input cloud bucket/container name.
-outputbucket=	Output cloud bucket/container name.
-rasterproxypath=	Path to a directory where Raster Proxy files are generated during the conversion process. Note: This is equivalent to the Raster Proxy Output folder in the UI, and will be ignored if the mode is rasterproxy.
-job=	User-defined name of job file.

Configuration file parameters

The configuration file (discussed in Default configuration files, above) can be used to define parameters for OptimizeRasters, minimizing the need for command line parameters. In addition to the command line arguments listed above, the configuration file contains the following parameters:

Table 4: OptimizeRasters configuration file parameters

Configuration File Parameter	Description
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).
KeepExtension	Specifies whether output raster file 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).
PyramidFactor	Defines the factors to build. Typically, this is 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 (e.g. average or nearest). The default is average sampling, with a few exceptions (e.g. quality files for Landsat 8).
Interleave	Defines the interleave setting for JPEG compression (Band or Pixel). Note: compression is improved when the output is 3band and the interleave is Pixel.
NoDataValue	Specifies the NoDataValue in the source (if applicable). 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.
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.

Appendix A: Working with Raster Proxy Files

Using Raster Proxy Files with ArcGIS

What are Raster Proxy files?

In ArcGIS ArcMap 10.4.1, Pro 1.3, and Server 10.4.1 and above, Raster Proxy files can be used to access raster data. 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. 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 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.

Common uses of 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.

Note: More information on mosaic datasets can be found at http://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/what-is-a-mosaic-dataset.htm.

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 TIF 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 also 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 TIF files work well, Raster Proxies created from MRF files work better for two reasons: (1) MRF files are structured to minimize the 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

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, perormance 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 cach 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 network-attached 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-dataset-overviews.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://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/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.

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 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.

Embedding Raster Proxies can be achieved two ways: (1) using specialized Python Raster Types (available with Arcmap 10.5 and ArcGIS Pro 1.4), which are specialized scripts that copy the Raster Proxy into the items of the mosaic dataset, or (2) using the Table Raster Type, which defines the properties of a raster and stores the content of the Raster Proxy in the Raster Field.

Note: For more information about Table Raster Type, see http://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/files-tables-and-web-services-raster-types.htm#ESRI_SECTION1_D8E60C757CA04174BED580F2101443BD.

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.

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.

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:\Python27\ArcGIS10.4\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache -ext=txt,mrf_cache -size=1

c:\Python27\ArcGIS10.4\python.exe c:\Image_Mgmt_Workflows\OptimizeRasters CleanMRFCache.py -input=z:\mrfcache

Optional arguments include:

Table 5: 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.

Parameter	Description
-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: Working with Amazon S3

Caution on case sensitivity

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.

AWS standards to manage 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 three ways:

1. The user can use the default environment variables (AWS_ACCESS_KEY_ID, AWS_ACCESS_KEY_ID)
2. The user can use the default AWS credential file located at %USERPROFILE%\.aws\credentials. Note: credential files are text/INI, without a file extension.
3. The user can set AWS credentials using the 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

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. This ensures that if the OptimizeRasters package is copied onto another machine, credentials available in the OptimizeRasters configuration file are not unintentionally exposed.

Reading from public AWS buckets

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

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

Overriding AWS credentials

To bypass the default AWS credentials, the user can edit the OptimizeRasters configuration file to include the information shown below. The credentials will be added to S3 input and S3 output storage respectively.

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

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

Note: The keys in the configuration file will take precedence over the AWS standard credential manager.

AWS credentials usage

If you're using the AWS credential manager for S3 bucket authentication, the AWS profile parameters in the OptimizeRasters configuration file need to be updated to match the profile names in the AWS credentials file.

For example, in the OptimizeRasters configuration file you can update parameters related to the AWS manager to match the AWS profile names.

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

For the above entries to work, the AWS profile file must be updated to reflect the profile names in the OptimizeRasters configuration file, as shown here:

[OptimizeRaster_S3In]
aws_access_key_id=XXX_YOUR_ACCESS_KEY_ID_XXX
aws_secret_access_key = XXX_SECRET_ACCESS_KEY_XXX

[OptimizeRaster_S3Out]
aws_access_key_id = XXX_YOUR_ACCESS_KEY_ID_XXX
aws_secret_access_key = XXX_SECRET_ACCESS_KEY_XXX

Setting parameters to read and write from Amazon S3

Various parameters need to be set in the OptimizeRasters configuration file in order to upload data to an S3 bucket.

For example, if you want to upload data to cloud storage here: http://mydata.s3.amazonaws.com/abc/pqr/t, then the following changes must be made to the configuration file:

Table 6: OptimizeRasters parameters for Amazon S3

Parameter	Change
Out_S3_Upload	Define as true.
Out_S3_Bucket	Specify the S3 bucket name where the data should be uploaded (e.g., mydata).

Parameter	Change
Out_S3_ParentFolder	Specify the S3 folder where the data should be uploaded (exclude the bucket name). The outputfolder path in the above example would be abc/pqr/t.
Out_S3_ID	Define the Access ID that will be used to make a connection.
Out_S3_Secret	Define the secretkey required to access the bucket.
Out_S3_DeleteAfterUpload	Define as true (the process will create intermediate data files that should be deleted after uploading).

The command line would then read as follows:

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

Setting access control on Amazon S3

Access to data on 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 node:

Table 7: Defining Amazon S3 Access Control Lists

Parameter	Change
Out_S3_ACL	Defines the canned ACL to apply to uploaded files. Acceptable values are: private, public-read, public-read-write, authenticated-read, bucket-owner-read, bucket-owner-full-control

To avoid making the data public, set the ACL as private and apply S3 bucket policy. Set up the bucket policy to allow only a specific machine IP to access the data, or to allow the data to be accessed from the machine where the server or desktop is running.

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 Raster Proxies from S3

To access rasters stored in S3, the user should use rasterproxy mode (See Appendix A: Working with Raster Proxy Files for more information on rasterproxy mode).

For example, if the input MRF files are in http://mydata.s3.amazonaws.com/abc/pqr/t, the user should make the following configuration changes:

Table 8: Configuration changes needed when using rasterproxy mode with Amazon S3

Parameter	Change
In_S3_Bucket	Specify the S3 bucket name where the input data is stored (e.g., mydata).
In_S3_ParentFolder	Specify the S3 folder where the input data is stored (exclude the bucket name). (E.g., abc/pqr/t)
In_S3_ID	Define the Access ID that will be used to make a connection.
In_S3_Secret	Define the secretkey required to access the bucket.

The command line would then read:

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

Appendix C: Working with Microsoft Azure

Appropriate configuration parameters in the parameter file must be set for OptimizeRasters to work with Azure Block storage.

Note: See also Using Obfuscation for Access Control in the Cloud, below, for additional help controlling access to rasters stored in the cloud.

Table 9: Configuration changes needed when using rasterproxy mode with Azure Block storage

Parameter	Change
<Out_Cloud_Type>azure</Out_Cloud_Type>	(Required) Defines cloud type. Valid values are [azure, amazon].
<Out_Azure_ParentFolder>folder/output</Out_Azure_ParentFolder>	Identifies the destination folder where files will be uploaded.
<Out_Azure_AccountName></Out_Azure_AccountName>	Identifies the Azure account name.
<Out_Azure_AccountKey></Out_Azure_AccountKey>	Identifies the Azure account key.
<Out_Azure_Container>tiffs</Out_Azure_Container>	Identifies the Azure container name.
<Out_Azure_Access>blob</Out_Azure_Access>	Identifies the access type. Valid values are [private, blob, container].**

****Azure access type definitions:**

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.

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

Azure command line usage

The command line would read as follows:

<path to python.exe> <path to optimizerasters.py> -input=c:/date/rasters -clouduploadtype=azure -cloudupload=true -tempoutput=c:/temp/tempoutput

Appendix D: Using Obfuscation for Access Control in the Cloud

OptimizeRasters has a feature to help with file obfuscation, which is a simpler alternative to common access control methods used in cloud environments.

Frequently, a user may need to make imagery accessible to only a select group. In Amazon S3, 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, once a user has downloaded a file from a secure location, access to the original file is irrelevant. As a result, sharing the short URL to the raster should be considered equivalent to sharing the large data files. Ultimately, though, obfuscation usually provides sufficient access security for large datasets, with advantages in terms of simplicity, performance, and interoperability.

OptimizeRasters has a feature to aid in the obfuscation of 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.

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. Please note the following syntax usage:

Table 10: 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 11: 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:

   Example 2: \ParentFolder\Scene1**RFd8GFf5_@**\a.tif

Note: 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 12: 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

Note: 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.
- The output hash is generated by concatenating the individual cloud output paths with the user-entered -hashkey 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 only 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 E: Working with ArcGIS Server and ArcGIS Desktop

The caching Raster Proxy 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.