GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...

Contents

Introduction

GDAL can access files located on "standard" file systems, ie in the / hierarchy on Unix-like systems or in C:\, D:\, etc... drives on Windows. But most GDAL raster and vector drivers use a GDAL specific abstraction to access files. This makes it possible to access less standard types of files, such as in-memory files, compressed files (.zip, .gz, .tar, .tar.gz archives), encrypted files, files stored on network (either publicly accessible, or in private buckets of commercial cloud storage services), etc.

Each special file system has a prefix, and the general syntax to name a file is /vsiPREFIX/...

Example:

```
gdalinfo /vsizip/my.zip/my.tif
```

Chaining

It is possible to chain multiple file system handlers.

ogrinfo a shapefile in a zip file on the internet:

```
ogrinfo -ro -al -so /vsizip//vsicurl/https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/ogr/data/poly.zip
```

ogrinfo a shapefile in a zip file on an ftp:

```
ogrinfo -ro -al -so /vsizip//vsicurl/ftp://user:password@example.com/foldername/file.zip/example.shp
```

Drivers supporting virtual file systems

Virtual file systems can only be used with GDAL or OGR drivers supporting the "large file API", which is now the vast majority of file based drivers. The full list of these formats can be obtained by looking at the driver marked with 'v' when running either gdalinfo –formats or ogrinfo –formats.

Notable exceptions are the netCDF, HDF4 and HDF5 drivers.

/vsizip/ (.zip archives)

/vsizip/ is a file handler that allows reading ZIP archives on-the-fly without decompressing them beforehand.

To point to a file inside a zip file, the filename must be of the form /vsizip/path/to/the/file.zip/path/inside/the/zip/file, where path/to/the/file.zip is relative or absolute and path/inside/the/zip/file is the relative path to the file inside the archive.

To use the .zip as a directory, you can use /vsizip/path/to/the/file.zip or /vsizip/path/to/the/file.zip/subdir. Directory listing is available with **VSIReadDir()**. A VSIStatL("/vsizip/...") call will return the uncompressed size of the file. Directories inside the ZIP file can be distinguished from regular files with the VSI_ISDIR(stat.st_mode) macro as for regular file systems. Getting directory listing and file statistics are fast operations.

Note: in the particular case where the .zip file contains a single file located at its root, just mentioning "/vsizip/path/to/the/file.zip" will work

Examples:

```
/vsizip/my.zip/my.tif  (relative path to the .zip)
/vsizip//home/even/my.zip/subdir/my.tif  (absolute path to the .zip)
/vsizip/c:\users\even\my.zip\subdir\my.tif
```

.kmz, .ods and .xlsx extensions are also detected as valid extensions for zip-compatible archives.

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .zip extension : /vsizip/{/path/to/the/archive}/path/inside/the/zip/file. Note that /path/to/the/archive may also itself use this alternate syntax.

Write capabilities are also available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file.

Creation of a new zip file:

```
fmain = VSIFOpenL("/vsizip/my.zip", "wb");
subfile = VSIFOpenL("/vsizip/my.zip/subfile", "wb");
VSIFWriteL("Hello World", 1, strlen("Hello world"), subfile);
VSIFCloseL(subfile);
VSIFCloseL(fmain);
```

Addition of a new file to an existing zip:

```
newfile = VSIFOpenL("/vsizip/my.zip/newfile", "wb");
VSIFWriteL("Hello World", 1, strlen("Hello world"), newfile);
VSIFCloseL(newfile);
```

Starting with GDAL 2.4, the GDAL_NUM_THREADS configuration option can be set to an integer or ALL_CPUS to enable multi-threaded compression of a single file. This is similar to the *pigz* utility in independent mode. By default the input stream is splitted by 1 MB chunks (can be tuned with the CPL_VSIL_DEFLATE_CHUNK_SIZE configuration option, with values like "x K" or "x M"), and each chunk is independently compressed (and terminated by a nine byte marker 0x00 0x00 0xFF 0xFF 0x00 0x00 0x00 0xFF 0xFF, signaling a full flush of the stream and dictionary, enabling potential independent decoding of each chunks). This slightly reduces the compression rate, so too small chunks should be avoided.

Read and write operations cannot be interleaved. The new zip must be closed before being re-opened for read.

/vsigzip/ (gzipped file)

/vsigzip/ is a file handler that allows reading on-the-fly in GZip (.gz) files without decompressing them priorly.

To view a gzipped file as uncompressed by GDAL, you must use the /vsigzip/path/to/the/file.gz syntax, where path/to/the/file.gz is relative or absolute

Examples:

```
/vsigzip/my.gz  (relative path to the .gz)
/vsigzip//home/even/my.gz  (absolute path to the .gz)
/vsigzip/c:\users\even\my.gz
```

VSIStatL() will return the uncompressed file size, but this is potentially a slow operation on large files, since it requires uncompressing the whole file. Seeking to the end of the file, or at random locations, is similarly slow. To speed up that process, "snapshots" are internally created in memory so as to be able being able to seek to part of the files already decompressed in a faster way. This mechanism of snapshots also apply to /vsizip/ files.

When the file is located in a writable location, a file with extension .gz.properties is created with an indication of the uncompressed file size (the creation of that file can be disabled by setting the CPL_VSIL_GZIP_WRITE_PROPERTIES configuration option to NO).

Write capabilities are also available, but read and write operations cannot be interleaved.

Starting with GDAL 2.4, the GDAL_NUM_THREADS configuration option can be set to an integer or ALL_CPUS to enable multi-threaded compression of a single file. This is similar to the *pigz* utility in independent mode. By default the input stream is splitted by 1 MB chunks (can be tuned with the CPL_VSIL_DEFLATE_CHUNK_SIZE configuration option, with values like "x K" or "x M"), and each chunk is independently compressed (and terminated by a nine byte marker 0x00 0x00 0xFF 0xFF 0x00 0x00 0x00 0xFF 0xFF, signaling a full flush of the stream and dictionary, enabling potential independent decoding of each chunks). This slightly reduces the compression rate, so too small chunks should be avoided.

/vsitar/ (.tar, .tgz archives)

/vsitar/ is a file handler that allows reading on-the-fly in regular uncompressed .tar or compressed .tgz or .tar.gz archives, without decompressing them priorly.

To point to a file inside a .tar, .tgz .tar.gz file, the filename must be of the form /vsitar/path/to/the/file.tar/path/inside/the/tar/file, where path/to/the/file.tar is relative or absolute and path/inside/the/tar/file is the relative path to the file inside the archive.

To use the .tar as a directory, you can use /vsizip/path/to/the/file.tar or /vsitar/path/to/the/file.tar/subdir. Directory listing is available with **VSIReadDir()**. A VSIStatL("/vsitar/...") call will return the uncompressed size of the file. Directories inside the TAR file can be distinguished from regular files with the VSI_ISDIR(stat.st_mode) macro as for regular file systems. Getting directory listing and file statistics are fast operations.

Note: in the particular case where the .tar file contains a single file located at its root, just mentioning "/vsitar/path/to/the/file.tar" will work

Examples:

```
/vsitar/my.tar/my.tif  (relative path to the .tar)
/vsitar//home/even/my.tar/subdir/my.tif  (absolute path to the .tar)
/vsitar/c:\users\even\my.tar\subdir\my.tif
```

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .tar extension : /vsitar/{/path/to/the/archive}/path/inside/the/tar/file. Note that /path/to/the/archive may also itself use this alternate syntax.

Network based file systems

A generic **/vsicurl/** file system handler exists for online resources that do not require particular signed authentication schemes. It is specialized into sub-filesystems for commercial cloud storage services, such as **/vsis3/**, **/vsigs/**, **/vsiaz/**, **/vsioss/** or **/vsiswift/**.

When reading of entire files in a streaming way is possible, prefer using the **/vsicurl_streaming/**, and its variants for the above cloud storage services, for more efficiency.

/vsicurl/ (http/https/ftp files: random access)

/vsicurl/ is a file system handler that allows on-the-fly random reading of files available through HTTP/FTP web protocols, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsicurl/http[s]://path/to/remote/resource or /vsicurl/ftp://path/to/remote/resource where path/to/remote/resource is the URL of a remote resource.

Example of ogrinfo a shapefile on the internet:

```
ogrinfo -ro -al -so /vsicurl/https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/ogr/data/poly.shp
```

Starting with GDAL 2.3, options can be passed in the filename with the following syntax: /vsicurl?[option_i=val_i&]*url=http://... where each option name and value (including the value of "url") is URL-encoded. Currently supported options are :

- use_head=yes/no: whether the HTTP HEAD request can be emitted. Default to YES. Setting this option overrides the behaviour of the CPL_VSIL_CURL_USE_HEAD configuration option.
- max_retry=number: default to 0. Setting this option overrides the behaviour of the GDAL_HTTP_MAX_RETRY configuration option.
- retry_delay=number_in_seconds: default to 30. Setting this option overrides the behaviour of the GDAL_HTTP_RETRY_DELAY configuration option.
- list_dir=yes/no: whether an attempt to read the file list of the directory where the file is located should be done. Default to YES.

Partial downloads (requires the HTTP server to support random reading) are done with a 16 KB granularity by default. Starting with GDAL 2.3, the chunk size can be configured with the CPL_VSIL_CURL_CHUNK_SIZE configuration option, with a value in bytes. If the driver detects sequential reading it will progressively increase the chunk size up to 2 MB to improve download performance. Starting with GDAL 2.3, the GDAL_INGESTED_BYTES_AT_OPEN configuration option can be set to impose the number of bytes read in one GET call at file opening (can help performance to read Cloud optimized geotiff with a large header).

The GDAL_HTTP_PROXY, GDAL_HTTP_PROXYUSERPWD and GDAL_PROXY_AUTH configuration options can be used to define a proxy server. The syntax to use is the one of Curl CURLOPT_PROXY, CURLOPT_PROXYUSERPWD and CURLOPT_PROXYAUTH options.

Starting with GDAL 2.1.3, the CURL_CA_BUNDLE or SSL_CERT_FILE configuration options can be used to set the path to the Certification Authority (CA) bundle file (if not specified, curl will use a file in a system location).

Starting with GDAL 2.3, additional HTTP headers can be sent by setting the GDAL_HTTP_HEADER_FILE configuration option to point to a filename of a text file with "key: value" HTTP headers.

Starting with GDAL 2.3, the GDAL_HTTP_MAX_RETRY (number of attempts) and GDAL_HTTP_RETRY_DELAY (in seconds) configuration option can be set, so that request retries are done in case of HTTP errors 429, 502, 503 or 504.

More generally options of **CPLHTTPFetch()** available through configuration options are available.

The file can be cached in RAM by setting the configuration option VSI_CACHE to TRUE. The cache size defaults to 25 MB, but can be modified by setting the configuration option VSI_CACHE_SIZE (in bytes). Content in that cache is discarded when the file handle is closed.

In addition, a global least-recently-used cache of 16 MB shared among all downloaded content is enabled by default, and content in it may be reused after a file handle has been closed and reopen, during the life-time of the process or until **VSICurlClearCache()** is called. Starting with GDAL 2.3, the size of this global LRU cache can be modified by setting the configuration option CPL_VSIL_CURL_CACHE_SIZE (in bytes).

Starting with GDAL 2.3, the CPL_VSIL_CURL_NON_CACHED configuration option can be set to values like "/vsicurl/http://example.com/foo.tif:/vsicurl/http://example.com/some_directory", so that at file handle closing, all cached content related to the mentioned file(s) is no longer cached. This can help when dealing with resources that can be modified during execution of GDAL related code. Alternatively, **VSICurlClearCache()** can be used.

Starting with GDAL 2.1, /vsicurl/ will try to query directly redirected URLs to Amazon S3 signed URLs during their validity period, so as to minimize round-trips. This behaviour can be disabled by setting the configuration option CPL_VSIL_CURL_USE_S3_REDIRECT to NO.

VSIStatL() will return the size in st_size member and file nature- file or directory - in st_mode member (the later only reliable with FTP resources for now).

VSIReadDir() should be able to parse the HTML directory listing returned by the most popular web servers, such as Apache or Microsoft IIS.

/vsicurl_streaming/ (http/https/ftp files: streaming)

/vsicurl_streaming/ is a file system handler that allows on-the-fly sequential reading of files streamed through HTTP/FTP web protocols, without prior download of the entire file. It requires GDAL to be built against libcurl.

Although this file handler is able seek to random offsets in the file, this will not be efficient. If you need efficient random access and that the server supports range downloading, you should use the /vsicurl/ file system handler instead.

Recognized filenames are of the form /vsicurl_streaming/http[s]://path/to/remote/resource or /vsicurl_streaming/ftp://path/to/remote/resource where path/to/remote/resource is the URL of a remote resource.

The GDAL_HTTP_PROXY, GDAL_HTTP_PROXYUSERPWD and GDAL_PROXY_AUTH configuration options can be used to define a proxy server. The syntax to use is the one of Curl CURLOPT_PROXY, CURLOPT_PROXYUSERPWD and CURLOPT_PROXYAUTH options.

Starting with GDAL 2.1.3, the CURL_CA_BUNDLE or SSL_CERT_FILE configuration options can be used to set the path to the Certification Authority (CA) bundle file (if not specified, curl will use a file in a system location).

The file can be cached in RAM by setting the configuration option VSI_CACHE to TRUE. The cache size defaults to 25 MB, but can be modified by setting the configuration option VSI_CACHE_SIZE (in bytes).

VSIStatL() will return the size in st_size member and file nature- file or directory - in st_mode member (the later only reliable with FTP resources for now).

/vsis3/ (AWS S3 files: random reading)

/vsis3/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in AWS S3 buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). Deletion of files with **VSIUnlink()** is also supported. Starting with GDAL 2.3, creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible.

Recognized filenames are of the form /vsis3/bucket/key where bucket is the name of the S3 bucket and key the S3 object "key", i.e. a filename potentially containing subdirectories.

The generalities of **/vsicurl/** apply.

Several authentication methods are possible. In order of priorities (first mentioned is the most prioritary)

1. If AWS_NO_SIGN_REQUEST=YES configuration option is set, request signing is disabled. This option might be used for buckets with public access rights. Available since GDAL 2.3
2. The AWS_SECRET_ACCESS_KEY and AWS_ACCESS_KEY_ID configuration options can be set. The AWS_SESSION_TOKEN configuration option must be set when temporary credentials are used.
3. Starting with GDAL 2.3, alternate ways of providing credentials similar to what the "aws" command line utility or Boto3 support can be used. If the above mentioned environment variables are not provided, the ~/.aws/credentials or UserProfile%/.aws/credentials file will be read (or the file pointed by CPL_AWS_CREDENTIALS_FILE). The profile may be specified with the AWS_PROFILE environment variable (the default profile is "default")
4. The ~/.aws/config or UserProfile%/.aws/config file may also be used (or the file pointer by AWS_CONFIG_FILE) to retrieve credentials and the AWS region.
5. If none of the above method succeeds, instance profile credentials will be retrieved when GDAL is used on EC2 instances.

The AWS_REGION (or AWS_DEFAULT_REGION starting with GDAL 2.3) configuration option may be set to one of the supported S3 regions and defaults to 'us-east-1'.

Starting with GDAL 2.2, the AWS_REQUEST_PAYER configuration option may be set to "requester" to facilitate use with Requester Pays buckets.

The AWS_S3_ENDPOINT configuration option defaults to s3.amazonaws.com.

The AWS_VIRTUAL_HOSTING configuration option defaults to TRUE. This allows you to configure the two ways to access the buckets, see Bucket and Host Name for more details.

- TRUE value, identifies the bucket via a virtual bucket host name, e.g.: mybucket.cname.domain.com
- FALSE value, identifies the bucket as the top-level directory in the URI, e.g.: cname.domain.com/mybucket

On writing, the file is uploaded using the S3 multipart upload API. The size of chunks is set to 50 MB by default, allowing creating files up to 500 GB (10000 parts of 50 MB each). If larger files are needed, then increase the value of the VSIS3_CHUNK_SIZE config option to a larger value (expressed in MB). In case the process is killed and the file not properly closed, the multipart upload will remain open, causing Amazon to charge you for the parts storage. You'll have to abort yourself with other means such "ghost" uploads (e.g. with the s3cmd utility) For files smaller than the chunk size, a simple PUT request is used instead of the multipart upload API.

Since GDAL 2.4, when listing a directory, files with GLACIER storage class are ignored unless the CPL_VSIL_CURL_IGNORE_GLACIER_STORAGE configuration option is set to NO.

Since

GDAL 2.1

/vsis3_streaming/ (AWS S3 files: streaming)

/vsis3_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in AWS S3 buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsis3_streaming/bucket/key where bucket is the name of the S3 bucket and resource the S3 object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to **/vsis3/**

Since

GDAL 2.1

/vsigs/ (Google Cloud Storage files: random reading)

/vsigs/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Google Cloud Storage buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Starting with GDAL 2.3, it also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). Deletion of files with **VSIUnlink()**, creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible.

Recognized filenames are of the form /vsigs/bucket/key where bucket is the name of the bucket and key the object "key", i.e. a filename potentially containing subdirectories.

The generalities of **/vsicurl/** apply.

Several authentication methods are possible. In order of priorities (first mentioned is the most prioritary)

1. The GS_SECRET_ACCESS_KEY and GS_ACCESS_KEY_ID configuration options can be set for AWS style authentication
2. The GDAL_HTTP_HEADER_FILE configuration option to point to a filename of a text file with "key: value" headers. Typically, it must contain a "Authorization: Bearer XXXXXXXXX" line.
3. (GDAL >= 2.3) The GS_OAUTH2_REFRESH_TOKEN configuration option can be set to use OAuth2 client authentication. See http://code.google.com/apis/accounts/docs/OAuth2.html This refresh token can be obtained with the "gdal_auth.py -s storage" or "gdal_auth.py -s storage-rw" script Note: instead of using the default GDAL application credentials, you may define the GS_OAUTH2_CLIENT_ID and GS_OAUTH2_CLIENT_SECRET configuration options (need to be defined both for gdal_auth.py and later execution of /vsigs)
4. (GDAL >= 2.3) The GOOGLE_APPLICATION_CREDENTIALS configuration option an be set to point to a JSon file containing OAuth2 service account credentials, in particular a private key and a client email. See https://developers.google.com/identity/protocols/OAuth2ServiceAccount for more details on this authentication method. The bucket must grant the "Storage Legacy Bucket Owner" or "Storage Legacy Bucket Reader" permissions to the service account. The GS_OAUTH2_SCOPE configuration option can be set to change the default permission scope from "https://www.googleapis.com/auth/devstorage.read_write" to "https://www.googleapis.com/auth/devstorage.read_only" if needed.
5. (GDAL >= 2.3) Variant of the previous method. The GS_OAUTH2_PRIVATE_KEY (or GS_OAUTH2_PRIVATE_KEY_FILE) and GS_OAUTH2_CLIENT_EMAIL can be set to use OAuth2 service account authentication. See https://developers.google.com/identity/protocols/OAuth2ServiceAccount for more details on this authentication method. The GS_OAUTH2_PRIVATE_KEY configuration option must contain the private key as a inline string, starting with "-----BEGIN PRIVATE KEY-----" Alternatively the GS_OAUTH2_PRIVATE_KEY_FILE configuration option can be set to indicate a filename that contains such a private key. The bucket must grant the "Storage Legacy Bucket Owner" or "Storage Legacy Bucket Reader" permissions to the service account. The GS_OAUTH2_SCOPE configuration option can be set to change the default permission scope from "https://www.googleapis.com/auth/devstorage.read_write" to "https://www.googleapis.com/auth/devstorage.read_only" if needed.
6. (GDAL >= 2.3) An alternate way of providing credentials similar to what the "gsutil" command line utility or Boto3 support can be used. If the above mentioned environment variables are not provided, the ~/.boto or UserProfile%/.boto file will be read (or the file pointed by CPL_GS_CREDENTIALS_FILE) for the

gs_secret_access_key and gs_access_key_id entries for AWS style authentication. If not found, it will look for the gs_oauth2_refresh_token (and optionally client_id and client_secret) entry for OAuth2 client authentication.

7. (GDAL >= 2.3) Finally if none of the above method succeeds, the code will check if the current machine is a Google Compute Engine instance, and if so will use the permissions associated to it (using the default service account associated with the VM). To force a machine to be detected as a GCE instance (for example for code running in a container with no access to the boot logs), you can set CPL_MACHINE_IS_GCE to YES.

Since

GDAL 2.2

/vsigs_streaming/ (Google Cloud Storage files: streaming)

/vsigs_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Google Cloud Storage buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsigs_streaming/bucket/key where bucket is the name of the bucket and key the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to **/vsigs/**

Since

GDAL 2.2

/vsiaz/ (Microsoft Azure Blob files: random reading)

/vsiaz/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Microsoft Azure Blob containers, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). A block blob will be created if the file size is below 4 MB. Beyond, an append blob will be created (with a maximum file size of 195 GB).

Deletion of files with **VSIUnlink()**, creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible. Note: when using **VSIMkdir()**, a special hidden .gdal_marker_for_dir empty file is created, since Azure Blob does not support natively empty directories. If that file is the last one remaining in a directory, **VSIRmdir()** will automatically remove it. This file will not be seen with **VSIReadDir()**. If removing files from directories not created with **VSIMkdir()**, when the last file is deleted, its directory is automatically removed by Azure, so the sequence VSIUnlink("/vsiaz/container/subdir/lastfile") followed by VSIRmdir("/vsiaz/container/subdir") will fail on the **VSIRmdir()** invocation.

Recognized filenames are of the form /vsiaz/container/key where container is the name of the container and key the object "key", i.e. a filename potentially containing subdirectories.

The generalities of **/vsicurl/** apply.

Several authentication methods are possible. In order of priorities (first mentioned is the most prioritary)

1. The AZURE_STORAGE_CONNECTION_STRING configuration option, given in the access key section of the administration interface. It contains both the account name and a secret key.
2. The AZURE_STORAGE_ACCOUNT and AZURE_STORAGE_ACCESS_KEY configuration options pointing respectively to the account name and a secret key.

Since

GDAL 2.3

/vsiaz_streaming/ (Microsoft Azure Blob files: streaming)

/vsiaz_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Microsoft Azure Blob containers, buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsiaz_streaming/container/key where container is the name of the container and key the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to **/vsiaz/**

Since

GDAL 2.3

/vsioss/ (Alibaba Cloud OSS files: random reading)

/vsioss/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Alibaba Cloud Object Storage Service (OSS) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). Deletion of files with **VSIUnlink()** is also supported. Creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible.

Recognized filenames are of the form /vsioss/bucket/key where bucket is the name of the OSS bucket and key the OSS object "key", i.e. a filename potentially containing subdirectories.

The generalities of **/vsicurl/** apply.

The OSS_SECRET_ACCESS_KEY and OSS_ACCESS_KEY_ID configuration options *must* be set. The OSS_ENDPOINT configuration option should normally be set to the appropriate value, which reflects the region attached to the bucket. The default is oss-us-east-1.aliyuncs.com. If the bucket is stored in another region than oss-us-east-1, the code logic will redirect to the appropriate endpoint.

On writing, the file is uploaded using the OSS multipart upload API. The size of chunks is set to 50 MB by default, allowing creating files up to 500 GB (10000 parts of 50 MB each). If larger files are needed, then increase the value of the VSIOSS_CHUNK_SIZE config option to a larger value (expressed in MB). In case the process is killed and the file not properly closed, the multipart upload will remain open, causing Alibaba to charge you for the parts storage. You'll have to abort yourself with other means. For files smaller than the chunk size, a simple PUT request is used instead of the multipart upload API.

Since

GDAL 2.3

/vsioss_streaming/ (Alibaba Cloud OSS files: streaming)

/vsioss_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Alibaba Cloud Object Storage Service (OSS) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsioss_streaming/bucket/key where bucket is the name of the bucket and key the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to **/vsioss/**

Since

GDAL 2.3

/vsiswift/ (OpenStack Swift Object Storage: random reading)

/vsiswift/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in OpenStack Swift Object Storage (swift) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). Deletion of files with **VSIUnlink()** is also supported. Creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible.

Recognized filenames are of the form /vsiswift/bucket/key where bucket is the name of the swift bucket and key the swift object "key", i.e. a filename potentially containing subdirectories.

The generalities of **/vsicurl/** apply.

Two authentication methods are possible. In order of priorities (first mentioned is the most prioritary)

1. The SWIFT_STORAGE_URL and SWIFT_AUTH_TOKEN configuration options are set respectively to the storage URL (e.g http://127.0.0.1:12345/v1/AUTH_something) and the value of the x-auth-token authorization token.
2. The SWIFT_AUTH_V1_URL, SWIFT_USER and SWIFT_KEY configuration options are set respectively to the endpoint of the Auth V1 authentication (e.g http://127.0.0.1:12345/auth/v1.0), the user name and the key/password. This authentication endpoint will be used to retrieve the storage URL and authorization token mentioned in the first authentication method.

This file system handler also allows sequential writing of files (no seeks or read operations are then allowed)

Since

GDAL 2.3

/vsiswift_streaming/ (OpenStack Swift Object Storage: streaming)

/vsiswift_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in OpenStack Swift Object Storage (swift) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form /vsiswift_streaming/bucket/key where bucket is the name of the bucket and key the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to **/vsiswift/**

Since

GDAL 2.3

/vsihdfs/ (Hadoop File System)

/vsihdfs/ is a file system handler that provides read access to HDFS. This handler requires GDAL to have been built with Java support (–with-java) and HDFS support (–with-hdfs). Support for this handler is currently only available on Unix-like systems. Note: support for the HTTP REST API (webHdfs) is also available with **/vsiwebhdfs/ (Web Hadoop File System REST API)**

Recognized filenames are of the form /vsihdfs/hdfsUri where hdfsUri is a valid HDFS URI.

Examples:

```
/vsihdfs/file:/tmp/my.tif  (a local file accessed through HDFS)
/vsihdfs/hdfs:/hadoop/my.tif  (a file stored in HDFS)
```

Since

GDAL 2.4

/vsiwebhdfs/ (Web Hadoop File System REST API)

/vsiwebhdfs/ is a file system handler that provides read and write access to HDFS through its HTTP REST API

Recognized filenames are of the form /vsiwebhdfs/http://hostname:port/webhdfs/v1/path/to/filename.

Examples:

```
/vsiwebhdfs/http://localhost:50070/webhdfs/v1/mydir/byte.tif
```

It also allows sequential writing of files (no seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported). Deletion of files with **VSIUnlink()** is also supported. Creation of directories with **VSIMkdir()** and deletion of (empty) directories with **VSIRmdir()** are also possible.

The generalities of **/vsicurl/** apply.

The following configuration options are available

- WEBHDFS_USERNAME=value: User name (when security is off).
- WEBHDFS_DELEGATION=value: Hadoop delegation token (when security is on).
- WEBHDFS_DATANODE_HOST=value: For APIs using redirect, substitute the redirection hostname with the one provided by this option (normally resolvable hostname should be rewritten by a proxy)
- WEBHDFS_REPLICATION=int_value: Replication value used when creating a file
- WEBHDFS_PERMISSION=int_value: Permission mask (to provide as decimal number) when creating a file or directory

This file system handler also allows sequential writing of files (no seeks or read operations are then allowed)

Since
GDAL 2.4

/vsistdin/ (standard input streaming)

/vsistdin/ is a file handler that allows reading from the standard input stream.

The filename syntax must be only "/vsistdin/"

The file operations available are of course limited to Read() and forward Seek(). Full seek in the first MB of a file is possible, and it is cached so that closing, re-opening /vsistdin/ and reading within thist first megabyte, is possible multiple times in the same process.

/vsistdout/ (standard output streaming)

/vsistdout/ is a file handler that allows writing into the standard output stream.

The filename syntax must be only "/vsistdout/"

The file operations available are of course limited to Write().

A variation of this file system exists as the /vsistdout_redirect/ file system handler, where the output function can be defined with **VSIStdoutSetRedirection()**.

/vsimem/ (in-memory files)

/vsimem/ is a file handler that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI*L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file. See **VSIFileFromMemBuffer()** and **VSIGetMemFileBuffer()**

Directory related functions are supported.

/vsimem/ files are visible within the same process. Multiple threads can access in reading to the same underlying file, provided they used different handles, but concurrent write and read operations on the same underlying file are not supported (locking is left to the responsibility of calling code)

/vsisubfile/ (portions of files)

The /vsisubfile/ virtual file system handler allows access to subregions of files, treating them as a file on their own to the virtual file system functions (**VSIFOpenL()**, etc).

A special form of the filename is used to indicate a subportion of another file: /vsisubfile/<offset>[_<size>], <filename>

The size parameter is optional. Without it the remainder of the file from the start offset as treated as part of the subfile. Otherwise only <size> bytes from <offset> are treated as part of the subfile. The <filename> portion may be a relative or absolute path using normal rules. The <offset> and <size> values are in bytes.

eg. /vsisubfile/1000_3000,/data/abc.ntf /vsisubfile/5000,../xyz/raw.dat

Unlike the /vsimem/ or conventional file system handlers, there is no meaningful support for filesystem operations for creating new files, traversing directories, and deleting files within the /vsisubfile/ area. Only the **VSIStatL()**, **VSIFOpenL()** and operations based on the file handle returned by **VSIFOpenL()** operate properly.

/vsisparse/ (sparse files)

The /vsisparse/ virtual file handler allows a virtual file to be composed from chunks of data in other files, potentially with large spaces in the virtual file set to a constant value. This can make it possible to test some sorts of operations on what seems to be a large file with image data set to a constant value. It is also helpful when wanting to add test files to the test suite that are too large, but for which most of the data can be ignored. It could, in theory, also be used to treat several files on different file systems as one large virtual file.

The file referenced by /vsisparse/ should be an XML control file formatted something like:

```
<VSISparseFile>
  <Length>87629264</Length>
  <SubfileRegion>  <!-- Stuff at start of file. -->
    <Filename relative="1">251_head.dat</Filename>
    <DestinationOffset>0</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>2768</RegionLength>
  </SubfileRegion>

  <SubfileRegion>  <!-- RasterDMS node. -->
    <Filename relative="1">251_rasterdms.dat</Filename>
    <DestinationOffset>87313104</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>160</RegionLength>
  </SubfileRegion>

  <SubfileRegion>  <!-- Stuff at end of file. -->
    <Filename relative="1">251_tail.dat</Filename>
    <DestinationOffset>87611924</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>17340</RegionLength>
  </SubfileRegion>

  <ConstantRegion>  <!-- Default for the rest of the file. -->
    <DestinationOffset>0</DestinationOffset>
    <RegionLength>87629264</RegionLength>
    <Value>0</Value>
```

```
  </ConstantRegion>
</VSISparseFile>
```

Hopefully the values and semantics are fairly obvious.

File caching

This is not a proper virtual file system handler, but a C function that takes a virtual file handle and returns a new handle that caches read-operations on the input file handle. The cache is RAM based and the content of the cache is discarded when the file handle is closed. The cache is a least-recently used lists of blocks of 32KB each.

The VSICachedFile class only handles read operations at that time, and will error out on write operations.

This is done with the VSICreateCachedFile() function, that is implictly used by a number of the above mentioned file systems (namely the default one for standard file system operations, and the /vsicurl/ and other related network file systems) if the VSI_CACHE configuration option is set to YES.

The default size of caching for each file is 25 MB (25 MB for each file that is cached), and can be controlled with the VSI_CACHE_SIZE configuration option (value in bytes).

/vsicrypt/ (encrypted files)

/vsicrypt/ is a special file handler is installed that allows reading/creating/update encrypted files on the fly, with random access capabilities.

Refert to **VSIInstallCryptFileHandler()** for more details.
