Оглавление

Оглавление

Глава 1

Сборка GDAL из исходных текстов

1.1 Сборка в среде Unix

Библиотека GDAL успешно собирается в системах Linux, IRIX, Solaris, BSD, и MacOS X. На Unix-платформах вы можете скомпилировать её следующим способом (предполагается, что исходные тексты распакованы или взяты из CVS модуля gdal):

```
% cd gdal
% ./configure
% make
% su
Password: ********
# make install
# exit
```

Для того, чтобы запустить GDAL после установки, необходимо сделать разделяемую библиотеку известной динамическому загрузчику. Обычно этого можно достичь, включив в значение переменной окружения LD_LIBRARY_PATH путь /usr/local/lib.

Обратите внимание на следующие особенности:

- Для сборки на платформе Unix необходима утилита GNU make. Скачайте и установите её, если она отсутствует в вашей системе.
- GDAL не требует большого числа дополнительных пакетов. Она включает в себя исходные тексты библиотек libz, libtiff, libgeotiff, libpng, libgif, и libjpeg, которые могут быть использованы, если отсутствуют их предустановленные версии (либо их использование нежелательно).
- Некоторые драйверы зависят от внешних библиотек. Команда configure –help выводит список параметров сборки, в том числе параметры для указания путей к внешним библиотекам, применяемым для работы с форматами GRASS, FITS, OGDI, HDF4, JPEG2000, ECW и т.п.
- Процесс сборки разделяемой библиотеки в сильной степени определяется набором инструментов GNU. Если вы используете что-то иное, чем GNU C++, то скорее всего окажется, что файлы .so не будут правильно собраны. В этом случае можно попробовать связывать приложение со статическими библиотеками (обычно gdal/gdal.a gdal/ogr/ogr.a gdal/frmts/o/ *.a gdal/gdal.a gdal/port/cpl.a), либо изменить команду

LD_SHARED в файле gdal/GDALmake.opt так, чтобы она работала на вашей платформе. Например, для платформ SGI обычно подходит команда c++ -shared -all.

- И GDAL, и включённые в поставку утилиты должны нормально собираться в среде Cygwin и на платформах, где разделяемые библиотеки не поддерживаются. Однако для сборки ваших собственных приложений требуется связывание с более широким набором библиотек. Команда gdal-config –libs выведет требуемый список библиотек.
- Логика "autoconf", проверяющая наличие libtiff, libpng и libjpeg недостаточно хорошо тестирует версии этих библиотек. Если вы используете предустановленные библиотеки и они не работают, перезапустите скрипт configure с параметрами "–with-png=internal", "–with-jpeg=internal", "–with-geotiff=internal" или "–with-libtiff=internal".
- Для сборки на IRIX скорее всего понадобится вручную исправить файл GDALmake.opt, получающийся в результате запуска configure и заменить "ld -shared" на "ld -shared -all".
- Если при сборке вы обнаружите проблемы с одним из драйверов для формата, который вам не нужен, то просто удалите его из списка GDAL_FORMATS в конце файла gdal/GDALmake.opt, сделайте make clean и затем make. Это исключит его из процесса компиляции и использования после запуска.
- При инсталляции без прав суперпользователя (в ваше собственный каталог, указанный параметром –prefix), то вы скорее всего получите проблемы с модулями для языка Python, поскольку они всегда пытаются установиться в каталог site-packages в дереве Python. Если поддержка Python вам не нужна, то вы можете отключить её на этапе конфигурирования с помощью параметра –without-python, либо указать доступный для записи каталог с помощью параметра –with-pymodir=<directory>. Модули Python будут установлены туда.

1.2 Сборка в среде Windows

В среде Windows GDAL можно собрать с помощью компиляторов MS VC++ 6.x или MS Visual Studio .NET (C++) из командной строки. Для этого вам обычно следует запустить скрипт VCVAR32.BAT, который устанавливается вместе в компилятором. Для MSVC 6.x он может быть расположен в

C:$\$*Program Files*$\$*Microsoft Visual Studio*$\$*VC*98$\$*bin*$\$*VCVARS*32.*BAT*

Как только все переменные окружения будут установлены, можно переходить в корневой каталог GDAL и запускать следующую команду:

C:$\$*GDAL* > *nmake /f makefile.vc*

После успешного завершения сборки можно установить все необходимые для функционирования GDAL файлы, используя цель install мейкфайла. Перед этим убедитесь, что переменные BINDIR и DATADIR в файле nmake.opt установлены в подходящие для вас значения:

C:$\$*GDAL* > *nmake /f makefile.vc install*

Если вы собираетесь применять GDAL в ваших собственных приложениях, то можно использовать следующую команду для установки всех необходимых библиотек и заголовочных файлов. Убедитесь, что в файле nmake.opt переменные LIBDIR и INCDIR установлены в подходящие для вас значения.

C:\$\$*GDAL* > *nmake /f makefile.vc devinstall*

Проекты, собираемые с GDAL, должны включать каталог, заданный в переменной INCDIR, в список путей для поиска заголовочных файлов, а каталог, заданный в LIBDIR в список /LIBPATH. Связываться следует с библиотекой импорта gdal_i.lib.

1.2.1 Основные параметры

Файл nmake.opt, расположенный в корневом каталоге GDAL, содержит большое количество параметров, которые могут быть настроены вручную:

- BINDIR: Каталог, в который будут установлены исполняемые файлы и разделяемые библиотеки при выполнение команды "nmake /f makefile.vc install".
- LIBDIR: Каталог, в который будет установлена библиотека импорта gdal_i.lib.
- INCDIR: Каталог, в который будут установлены заголовочные файлы GDAL/OGR.
- DATADIR: Каталог, в который будут установлены необходимые файлы данных.
- OPTFLAGS: Эта переменная должна содержать флаги компилятора, которые следует применить при сборке GDAL. По умолчанию используется сборка с отладочной информацией, а флаги для оптимизированной окончательной сборки закомментированы. Пожалуйста, обратите внимание на то, что если вы будете собирать GDAL с использованием дополнительных библиотек (установленных из бинарных пакетов, либо собранных вами ранее), вам необходимо использовать одинаковые параметры сборки для этих библиотек и для GDAL. Например, если вы используете флаг /MD в строке OPTFLAGS (связывание с многопоточной разделяемой стандартной библиотекой), то тот же самый флаг должен быть использован при компиляции других библиотек, таких, как HDF4 и JasPer. Если у вас нет исходных текстов дополнительных библиотек, то прочитайте документацию, поставляемую вместе с ними, для того, чтобы выяснить, как она была скомпилирована и какие флаги вам следует выставить в строке OPTFLAGS. Смешивание различных параметров компиляции или флагов оптимизации/отладки приведёт к ошибкам при сборке или к падениям после запуска на исполнение.

1.2.2 Дополнительные параметры

- PY_INST_DIR: Каталог, в который будут устанавливаться модули поддержки языка Python. Необходим только в случае, если эта поддержка включена.
- PYDIR: Каталог, в который установлено дерево Python. Используется для для поиска заголовочных файлов Python. Если эта переменная указывает на несуществующий каталог, то тогда поддержка Python будет выключена.
- DLLBUILD: Установите эту переменную в значение "1", чтобы заставить утилиты OGR связываться с разделяемой библиотекой GDAL, а не компилироваться статически.
- INCLUDE_OGR_FRMTS: Установите эту переменную в значение "YES", чтобы включить сборку библиотеки OGR, либо закомментируйте её, для исключения OGR из сборки.
- SETARGV: Путь к файлу setargv.obj из поставки Visual Studio. Включите эту переменную, если вы хотите получить раскрытие метасимволов в аргументах командной строки. Если вам это не требуется, оставьте параметр закомментированным.

- ECWDIR/ECWLIB: Раскомментируйте эти параметры для включения поддержки ECW. ECWDIR должна указывать на каталог, в котором установлены библиотеки ECW.
- OGDIDIR/OGDIVER/OGDILIB: Раскомментируйте эти параметры для включения поддержки OGDI, отредактировав OGDIDIR и OGDIVER в соответствии с вашей системой.
- HDF4_DIR: Раскомментируйте и отредактируйте эту переменную для включения поддержки NCSA HDF Release 4.
- JASPER_DIR/JASPER_INCLUDE/JASPER_LIB: Эти переменные указывают на каталоги, в которые установлены компоненты библиотеки JasPer. Эта библиотека необходима для поддержки формата JPEG2000.
- XERCES_DIR/XERCES_INCLUDE/XERCES_LIB: Раскомментируйте и отредактируйте переменную XERCES_DIR для включения поддержки XML-анализатора Xerces при чтении файлов в формате GML.
- FME_DIR: Раскомментируйте и отредактируйте эту переменную для включения поддержки векторного формата FMEObject.
- JPEG_EXTERNAL_LIB/JPEGDIR/JPEG_LIB: Эти переменные используются для связывания GDAL с внешней библиотекой JPEG (по умолчанию будет скомпилирована встроенная библиотека). Раскомментируйте их и отредактируйте пути.
- PNG_EXTERNAL_LIB/PNGDIR/PNG_LIB: Эти переменные используются для связывания GDAL с внешней библиотекой PNG (по умолчанию будет скомпилирована встроенная библиотека). Раскомментируйте их и отредактируйте пути.

При добавлении новых компонентов я часто забываю обновить мейкфайлы для Windows, поэтому если при сборке что-то не найдено, попробуйте сравнить списки файлов в соответствующем makefile.vc со списками в GNUmakefile, либо просто сообщите об этом мне.

1.3 Внешние библиотеки

Некоторые драйверы требуют установки на вашу систему следующих дополнительных библиотек:

- Библиотека NCSA HDF. Может быть скачана с домашней страницы NCSA HDF в Национальном центре суперкомпьютерных приложений.

  Если в поставку вашей операционной системы уже входит библиотека HDF, то вы можете использовать её.

  Обратите внимание на то, что библиотека NCSA HDF собрана с целым рядом параметров, определённых в файле hlimits.h. В частности,hlimits.h определяет максимальное число открытых файлов:

  ```
  #   define MAX_FILE   32
  ```

  Если вам нужно открывать одновременно большее число файлов в формате HDF4, то это значение следует изменить и перекомпилировать библиотеку HDF4 (при этом нет необходимости в пересборке GDAL, если она уже собрана с поддержкой разделяемой библиотеки HDF4).

- Поддержка JPEG2000 основывается на библиотеке JasPer, доступной на своей домашней странице.

  Сама по себе библиотека JasPer является транслятором между несколькими растровыми форматами файлов. GDAL использует только форматы JP2 и JPC.

  Если вы хотите получить поддержку расширения GeoJP2, то вам потребуется модифицированная библиотека JasPer, которая может быть скачана отсюда: ftp://ftp.remotesensing.org/gdal/jasper-1.701.0.uuid.tar.gz

- Драйвер формата MrSID требует декодирующую библиотеку LizardTech (DSDK). Это не свободное программное обеспечение, однако оно бесплатно доступно на сайте http://developer.lizardtech.com/. Если вы хотите создавать файлы в формате MrSID, вам потребуется кодирующая библиотека (ESDK). Свяжитесь с представителями компании LizardTech для выяснения условий её получения. Библиотеки MrSID распространяются в бинарном виде и, если вы используете GCC, убедитесь, что применяется тот же самый компилятор, который был использован для сборки SDK. Данная библиотека написана на C++, поэтому вы можете получить несовместимость в схеме именования символов при использовании различных версий GCC (2.95.x and 3.x).

- Поддержка формата NetCDF требует библиотеку netCDF. После сборки и установки этой библиотеки используйте параметр "–with-netcdf=<path to install tree>" для конфигурирования GDAL. При одновременном использовании библиотек netCDF и HDF могут возникнуть конфликты связывания, поэтому такое использование может оказаться невозможным. Поддержка NetCDF не собиралась и не тестировалась в среде Windows.

1.4 Поддержка больших файлов

GDAL поддерживает чтение и запись больших файлов (> 2GiB), если это возможно в вашей операционной системе. Информацию о поддержке больших файлов в системе Linux можно получить здесь: http://www.suse.de/~aj/linux_lfs.html. Вкратце: если вы работаете с ядром 2.4.x и glibc 2.2.x, то проблем быть не должно. Максимальный размер файла зависит от размера кластера файловой системы. Для файловой системы ext2 с кластером в 1 KiB это 16448 MiB, для ext2 с кластером 4 KiB это 2048 GiB. Другие файловые системы могут работать с файлами даже большего размера.

Информацию о поддержке больших файлов в Windows можно найти на страницах MSDN. Вкратце: максимальный размер файла на NTFS ограничен (2^64 - 1) байтами, на FAT32 и FAT16 это (2^32 - 1) байт. Поэтому не пытайтесь создавать файлы, большие, чем 4 GiB на FAT32. В некоторых случаях вы даже не получите сообщения об ошибке при переходе за барьер в 4GiB, и ваши данные будут потеряны. Это не ошибка в GDAL, это проблема Windows.

Глава 2

Модель данных GDAL

Настоящий документ описывает модель данных, применяемую библиотекой GDAL: разновидности информации, которая может содержаться в источниках данных GDAL, а также их семантика.

2.1 Набор данных

Набор данных (представляемый классом GDALDataset) состоит из связанных растровых каналов, а также некоторой дополнительной информации, общей для всего набора. В частности, набор данных имеет понятие размера (ширины и высоты), общем для всех каналов. Набор данных также отвечает за географическую привязку и указание координатной системы, также общими для всех каналов. Сам набор данных может иметь ассоциированный комплект метаданных: список пар ключ/значение в форме ASCIIZ строк

Заметим, что набор данных GDAL и модель растровых каналов изначально базируется на спецификации регулярных покрытий консорциума OpenGIS.

2.1.1 Система координат

Географические координатные системы представляются в виде строк OpenGIS WKT (Well Known Text). Такая строка может содержать:

- Общее название координатной системы.
- Название географической координатной системы.
- Идентификатор системы координат.
- Название эллипсоида, большая полуось, сжатие.
- Название начального меридиана и его смещение относительно Гринвичского.
- Название проекции (например, Transverse Mercator).
- Список параметров проекции (например, положение осевого меридиана).
- Название единиц измерения и множитель для перехода к метрам или радианам.
- Названия и порядок следования координатных осей.
- Коды для вышеперечисленных параметров по предопределённым таблицам, таким, как таблицы EPSG.

Дополнительные сведения об определениях координатных систем с помощью строк OpenGIS WKT и способах работы с ними можно найти в разделе osr_tutorial, а также в документации на класс OGRSpatialReference.

Координатная система, возвращаемая методом GDALDataset::GetProjectionRef() описывает геодезические координаты, определяемые с помощью матрицы аффинного преобразования, возвращаемой функцией GDALDataset::GetGeoTransform(). Координатная система, возвращаемая методом GDALDataset::GetGCPProjection() описывает геодезические координаты наземных контрольных точек, список которых даёт метод GDALDataset::GetGCPs().

Заметим, что пустая строка (""), возвращаемая в качестве определения координатной системы, означает отсутствие информации о координатной системе.

2.1.2 Аффинное преобразование геодезических координат

Существует два способа задать связь между точками растра (в терминах строка/столбец) и геодезическими координатами. Первый и наиболее часто используемый — это аффинное преобразование. Второй предполагает использование наземных контрольных точек.

Матрица аффинного преобразования состоит из шести коэффициентов, возвращаемых методом GDALDataset::GetGeoTransform(), которая отображает строку/столбец в пространство геодезических координат по следующему соотношению:

```
Xgeo = GT(0) + Xpixel*GT(1) + Yline*GT(2)
Ygeo = GT(3) + Xpixel*GT(4) + Yline*GT(5)
```

В случае изображений, верхняя рамка которых ориентирована на север, коэффициенты GT(2) и GT(4) равны нулю, GT(1) равен ширине пиксела, а GT(5) — его высоте. Координаты (GT(0),GT(3)) задают положение верхнего левого угла верхнего левого пиксела растра.

Заметим, что координаты строка/столбец могут принимать значения от (0.0,0.0) в верхнем левом углу верхнего левого пиксела до (ширина_в пикселах,высота_в_пикселах) в правом нижнем углу правого нижнего пиксела. Положение центра верхнего левого пиксела в терминах строка/столбец будет, таким образом, (0.5,0.5).

2.1.3 Наземные контрольные точки (Ground Control Points, GCPs)

Набор данных может иметь список контрольных точек, связывающих одну или несколько точек растра с их геодезическими координатами. Все контрольные точки заданы в одной и той же координатной системе, возвращаемой методом GDALDataset::GetGCPProjection(). Каждая контрольная точка (описываемая классом GDAL_GCP) содержит следующее:

```
typedef struct
{
    char *pszId;
    char *pszInfo;
    double  dfGCPPixel;
    double dfGCPLine;
    double dfGCPX;
    double dfGCPY;
    double dfGCPZ;
} GDAL_GCP;
```

Строка pszId должна быть уникальным (и, часто, но не всегда, числовым) идентификатором для контрольной точке в списке точек данного набора. pszInfo — это обычно пустая строка, но она также может содержать любой вспомогательный текст, относящийся к данной точке. Теоретически это поле может также содержать машинно читаемую информацию о статусе данной точки, однако в настоящий момент эта возможность не реализована.

Координаты (dfGCPPixel, dfGCPLine) задают положение точки на растре. Координаты (dfGCPX, dfGCPY, dfGCPZ) задают соответствующую привязку точки к геодезическим координатам (координата Z часто бывает нулём).

Модель данных GDAL не содержит механизма преобразования, получаемого из контрольных точек, — это оставлено для приложений более высокого уровня. Обычно для этого применяются полиномы от 1-го до 5-го порядка.

Обычно набор данных содержит либо аффинное преобразование, либо контрольные точки, либо ничего. В редких случаях может присутствовать и то, и другое, тогда не определено, какой из способов имеет преимущество.

2.1.4 Метаданные

Метаданные — это вспомогательные данные, хранящиеся в виде пар ключ/значение. Их состав определяется форматом хранения данных и приложением. Ключи должны быть "хорошими" лексемами (без пробельных и специальных символов). Значения могут иметь любую длину и содержать любые символы, за исключением нулевого символа ASCII.

Механизм управления метаданными хорошо оптимизирован для работы с очень большими блоками данных. Однако работа с метаданными, превышающими в размере 100KiB скорее всего приведёт к снижению производительности.

В будущем предполагается введение нескольких стандартных ключей с предопределённой семантикой, однако в настоящий момент таковых нет.

Некоторые форматы данных содержат собственные метаданные, в то время как драйверы для других форматов могут отображать поля, специфичные для данного формата, в записи метаданных. Например, драйвер TIFF возвращает содержимое некоторых информационных тегов в виде метаданных, включая поле дата/время, которое будет выглядеть как:

TIFFTAG_DATETIME=1999:05:11 11:29:56

2.2 Растровый канал

Растровый канал описывается в GDAL с помощью класса GDALRasterBand. Он не обязательно должен представлять всё изображение. Например, 24-битное RGB-изображение должно быть представлено как набор данных с тремя каналами, по одному для красной, зелёной и синей компоненты.

Растровый канал имеет следующие свойства:

- Ширина и высота в пикселах и строках. Они будут теми же самыми, что и для всего набора данных, если это канал в полном разрешении.
- Тип данных (GDALDataType). Один из вещественных (Byte, UInt16, Int16, UInt32, Int32, Float32, Float64), или комплексных типов CInt16, CInt32, CFloat32, and CFloat64.
- Размер блока. Предпочтительный (наиболее эффективный) размер блока данных для считывания. Для изображений, хранящихся построчно, это в большинстве случаев будет одна строка.
- Список метаданных в виде пар ключ/значение в том же формате, что и для всего набора данных, но содержащих информацию, специфичную для данного канала.
- Необязательная строка описания.
- Необязательный список категорий (например, названий классов на тематической карте).
- Необязательные минимальное и максимальное значение.
- Необязательные калибровочные коэффициенты для пересчёта значений растра в физические величины (например, перевод отсчётов высоты в метры).
- Необязательное название единиц измерения. Например, это поле может содержать единицы измерения высоты для модели рельефа.
- Цветовая интерпретация канала. Одна из:

  - GCI_Undefined: по умолчанию, не определено.
  - GCI_GrayIndex: одиночное изображение в оттенках серого.
  - GCI_PaletteIndex: изображение с цветовой палитрой.
  - GCI_RedBand: красная компонента RGB- или RGBA-изображения.
  - GCI_GreenBand: зелёная компонента RGB- или RGBA-изображения.
  - GCI_BlueBand: синяя компонента RGB- или RGBA-изображения.
  - GCI_AlphaBand: альфа-канал RGBA-изображения.
  - GCI_HueBand: компонента цвета HLS-изображения.
  - GCI_SaturationBand: компонента насыщенности HLS-изображения.
  - GCI_LightnessBand: компонента яркости HLS-изображения.
  - GCI_CyanBand: голубая компонента CMY- или CMYK-изображения.
  - GCI_MagentaBand: пурпурная компонента CMY- или CMYK-изображения.
  - GCI_YellowBand: жёлтая компонента CMY- или CMYK-изображения.
  - GCI_BlackBand: чёрная компонента CMY- или CMYK-изображения.

- Таблица цветов (палитра), которая будет подробно описана ниже.
- Информация об уменьшенных обзорных изображениях (пирамидах).

2.3 Таблица цветов

Таблица цветов состоит из нуля или нескольких записей, описываемых на языке C в виде следующей структуры:

```
typedef struct
{
    /- серый, красный, голубой или цвет -/
    short      c1;

    /- зелёный, пурпурный или яркость -/
    short      c2;

    /- синий, жёлтый или насыщенность -/
    short      c3;

    /- альфа-канал или чёрный -/
    short      c4;
} GDALColorEntry;
```

Таблица цветов также имеет индикатор интерпретации (GDALPaletteInterp), который указывает на то, как параметры c1/c2/c3/c4 должны быть проинтерпретированы приложением. Этот индикатор может принимать следующие значения:

- GPI_Gray: Считать c1 значением в градациях серого.
- GPI_RGB: Считать c1 красным, c2 зелёным, c3 синим, а c4 — альфа-каналом.
- GPI_CMYK: Считать c1 голубым, c2 пурпурным, c3 жёлтым, c4 чёрным.

- GPI_HLS: Считать c1 цветом, c2 яркостью, c3 насыщенностью.

Для связывания цвета с пикселем значение этого пиксела используется в качестве индекса в таблице цветов. Это значит, что цвета всегда располагаются в таблице начиная с нулевого индекса и далее по возрастанию. Не существует механизма для предварительного масштабирования значений, прежде, чем будет применена таблица цветов.

2.4 Обзорные изображения

Канал может содержать обзорные изображения. Каждое обзорное изображение представлено в виде отдельного канала GDALRasterBand. Размер обзорного изображения (в терминах строк и столбцов) будет отличаться от базового полноразмерного растра, однако географически они будут покрывать один и тот же регион.

Обзорные изображения применяются для быстрого отображения уменьшенных копий растра, вместо того, чтобы читать полноразмерное изображение с последующим масштабированием.

Канал также обладает свойством HasArbitraryOverviews, которое равно TRUE, если растр может быть эффективно прочитан в любом разрешении, но не имеет чётких пирамидальных слоёв. Такими свойствами обладают некоторые алгоритмы кодирования изображений с помощью БПФ и вейвлетов, а также изображения, получаемые из внешних источников (таких, как OGDI), когда масштабирование производится на удалённой стороне.

Глава 3

Руководство по использованию GDAL

3.1 Открытие файла

Перед тем, как открыть набор данных, поддерживаемый GDAL, необходимо зарегистрировать драйверы. Для каждого поддерживаемого формата существует отдельный драйвер. В большинстве случаев это можно сделать с помощью функции GDALAllRegister(), которая пытается зарегистрировать все известные драйверы, включая те, что загружены из динамически подгружаемых модулей .so, используя GDALDriverManager::AutoLoadDrivers(). Имеется возможность ограничить набор драйверов, доступных в приложении; примером может служить код модуля gdalallregister.cpp.

Как только драйверы зарегистрированы, приложение должно вызвать функцию GDALOpen() для открытия набора данных. В качестве параметров функция принимает название набора данных и режим доступа (GA_ReadOnly или GA_Update).

На языке C++:

```
#include "gdal_priv.h"

int main()
{
    GDALDataset  *poDataset;

    GDALAllRegister();

    poDataset = (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
    if( poDataset == NULL )
    {
        ...;
    }
```

На языке C:

```
#include "gdal.h"

int main()
{
    GDALDatasetH  hDataset;

    GDALAllRegister();

    hDataset = GDALOpen( pszFilename, GA_ReadOnly );
    if( hDataset == NULL )
    {
        ...;
    }
```

На языке Python:

```
    import gdal
    from gdalconst import *

    dataset = gdal.Open( filename, GA_ReadOnly )
    if dataset is None:
        ...
```

Если GDALOpen() возвращает NULL, это означает, что операция не удалась и что сообщения об ошибке были посланы с помощью функции CPLError(). Если вы хотите управлять процессом выдачи пользователю сообщений об ошибках, то обратитесь к документации на функцию CPLError(). Вообще говоря, CPLError() применяется во всех компонентах GDAL для выдачи сообщений об ошибках. Заметим также, что pszFilename не должна обязательно

быть именем файла на физическом носителе (хотя обычно это так). Интерпретация этого параметра зависит от драйвера, это может быть URL или имя файла с дополнительными параметрами, управляющими процессом чтения, либо чем-то иным. Пожалуйста, не ограничивайте диалоги выбора набора данных для открытия только лишь файлами на физических носителях.

3.2 Чтение информации о наборе данных

Как было описано в разделе Модель данных GDAL, набор данных GDALDataset содержит список растровых каналов, покрывающих одну и ту же территорию и имеющих одинаковое разрешение. Он также содержит метаданные, координатную систему, географическую привязку, размер растра и некоторую дополнительную информацию.

```
adfGeoTransform[0] /* координата x верхнего левого угла */
adfGeoTransform[1] /* ширина пиксела */
adfGeoTransform[2] /* поворот, 0, если изображение ориентировано на север */
adfGeoTransform[3] /* координата y верхнего левого угла */
adfGeoTransform[4] /* поворот, 0, если изображение ориентировано на север */
adfGeoTransform[5] /* высота пиксела */
```

Если мы хотим вывести некоторую общую информацию о наборе данных, то можно сделать следующее:

На языке C++:

```
double        adfGeoTransform[6];

printf( "Драйвер: %s/%s\n",
        poDataset->GetDriver()->GetDescription(),
        poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );

printf( "Размер %dx%dx%d\n",
        poDataset->GetRasterXSize(), poDataset->GetRasterYSize(),
        poDataset->GetRasterCount() );

if( poDataset->GetProjectionRef()  != NULL )
    printf( "Проекция \"%s\"\n", poDataset->GetProjectionRef() );

if( poDataset->GetGeoTransform( adfGeoTransform ) == CE_None )
{
    printf( "Начало координат (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Размер пиксела (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

На языке C:

```
GDALDriverH   hDriver;
double        adfGeoTransform[6];

hDriver = GDALGetDatasetDriver( hDataset );
printf( "Драйвер: %s/%s\n",
        GDALGetDriverShortName( hDriver ),
        GDALGetDriverLongName( hDriver ) );

printf( "Размер %dx%dx%d\n",
        GDALGetRasterXSize( hDataset ),
```

```
        GDALGetRasterYSize( hDataset ),
        GDALGetRasterCount( hDataset ) );

if( GDALGetProjectionRef( hDataset ) != NULL )
    printf( "Проекция \"%s\"\n", GDALGetProjectionRef( hDataset ) );

if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
{
    printf( "Начало координат (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Размер пиксела (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

На языке Python:

```
print 'Драйвер: ', dataset.GetDriver().ShortName,'/', \
      dataset.GetDriver().LongName
print 'Размер ',dataset.RasterXSize,'x',dataset.RasterYSize, \
      'x',dataset.RasterCount
print 'Проекция ',dataset.GetProjection()

geotransform = dataset.GetGeoTransform()
if not geotransform is None:
    print 'Начало координат (',geotransform[0], ',',geotransform[3],')'
    print 'Размер пиксела = (',geotransform[1], ',',geotransform[5],')'
```

3.3 Чтение растрового канала

Одним из способов чтения растровых данных с помощью GDAL является поканальный доступ. При этом при последовательном чтении каналов доступны метаданные, параметры блоков, а также различная другая информация. Далее приведены примеры кода, извлекающего объект GDALRasterBand из набора данных (каналы нумеруются от 1 и до GetRasterCount()) и выводящего некоторую информацию о канале.

На языке C++:

```
    GDALRasterBand  *poBand;
    int             nBlockXSize, nBlockYSize;
    int             bGotMin, bGotMax;
    double          adfMinMax[2];

    poBand = poDataset->GetRasterBand( 1 );
    poBand->GetBlockSize( &nBlockXSize, &nBlockYSize );
    printf( "Размер блока %dx%d, тип данных %s, ColorInterp=%s\n",
            nBlockXSize, nBlockYSize,
            GDALGetDataTypeName(poBand->GetRasterDataType()),
            GDALGetColorInterpretationName(
                poBand->GetColorInterpretation()) );

    adfMinMax[0] = poBand->GetMinimum( &bGotMin );
    adfMinMax[1] = poBand->GetMaximum( &bGotMax );
    if( ! (bGotMin && bGotMax) )
        GDALComputeRasterMinMax((GDALRasterBandH)poBand, TRUE, adfMinMax);

    printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

    if( poBand->GetOverviewCount() > 0 )
        printf( "Канал содержит %d обзорных изображений.\n",
                poBand->GetOverviewCount() );
```

```
if( poBand->GetColorTable() != NULL )
    printf( "Канал содержит таблицу цветов с %d записями.\n",
            poBand->GetColorTable()->GetColorEntryCount() );
```

In C:

```
GDALRasterBandH hBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];

hBand = GDALGetRasterBand( hDataset, 1 );
GDALGetBlockSize( hBand, &nBlockXSize, &nBlockYSize );
printf( "Размер блока %dx%d, тип данных %s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(GDALGetRasterDataType(hBand)),
        GDALGetColorInterpretationName(
            GDALGetRasterColorInterpretation(hBand)) );

adfMinMax[0] = GDALGetRasterMinimum( hBand, &bGotMin );
adfMinMax[1] = GDALGetRasterMaximum( hBand, &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax( hBand, TRUE, adfMinMax );

printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

if( GDALGetOverviewCount(hBand) > 0 )
    printf( "Канал содержит %d обзорных изображений.\n",
            GDALGetOverviewCount(hBand));

if( GDALGetRasterColorTable( hBand ) != NULL )
    printf( "Канал содержит таблицу цветов с %d записями.\n",
            GDALGetColorEntryCount(
                GDALGetRasterColorTable( hBand ) ) );
```

На языке Python:

```
band = dataset.GetRasterBand(1)

print 'Тип данных',gdal.GetDataTypeName(band.DataType)

min = band.GetMinimum()
max = band.GetMaximum()
if min is not None and max is not None:
    (min,max) = ComputeRasterMinMax(1)
print 'Min=%.3f, Max=%.3f' % (min,max)

if band.GetOverviewCount() > 0:
    print 'Канал содержит ', band.GetOverviewCount(), \
    ' обзорных изображений.'

if not band.GetRasterColorTable() is None:
    print 'Канал содержит таблицу цветов с ', \
    band.GetRasterColorTable().GetCount(), ' записями.'
```

3.4 Чтение растровых данных

Существует несколько способов чтения растровых данных, однако наиболее общим является использование метод GDALRasterBand::RasterIO(). Этот метод автоматически производит конвертацию типов данных, масштабирование и вырезку области интереса. Следующий код читает первую строку данных в буфер соответствующего размера, преобразовывая их при этом в вещественный тип одинарной точности.

На языке C++:

```
float *pafScanline;
int   nXSize = poBand->GetXSize();

pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
poBand->RasterIO( GF_Read, 0, 0, nXSize, 1,
                  pafScanline, nXSize, 1, GDT_Float32,
                  0, 0 );
```

На языке C:

```
float *pafScanline;
int   nXSize = GDALGetRasterBandXSize( hBand );

pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
GDALRasterIO( hBand, GF_Read, 0, 0, nXSize, 1,
              pafScanline, nXSize, 1, GDT_Float32,
              0, 0 );
```

На языке Python:

```
scanline = band.ReadRaster( 0, 0, band.XSize, 1, \
                            band.XSize, 1, GDT_Float32 )
```

Здесь возвращаемая строка имеет тип string, и содержит xsize∗4 байт вещественных данных. Эта строка может быть преобразована в базовые типы языка Python с помощью модуля struct из стандартной библиотеки:

```
import struct

tuple_of_floats = struct.unpack('f' * b2.XSize, scanline)
```

Вызов функции The RasterIO производится со следующими аргументами:

```
CPLErr GDALRasterBand::RasterIO( GDALRWFlag eRWFlag,
                                 int nXOff, int nYOff, int nXSize, int nYSize,
                                 void * pData, int nBufXSize, int nBufYSize,
                                 GDALDataType eBufType,
                                 int nPixelSpace,
                                 int nLineSpace )
```

Заметим, что один и тот же вызов RasterIO() применяется как для чтения, так и для записи, в зависимости от значения флага eRWFlag (GF_Read или GF_Write). Аргументы nXOff, nYOff, nXSize, nYSize описывают окно растра для чтения (или записи). Это окно необязательно должно совпадать с границами смежных блоков, однако считывание может быть более эффективным, если границы совпадают.

pData — это указатель на буфер в памяти, куда должны быть прочитаны (или откуда записаны) данные. Фактический тип этого буфера должен совпадать с типом, передаваемым в параметре eBufType, например, GDT_Float32 или GDT_Byte. Функция RasterIO() возьмёт на себя преобразование между типом данных буфера и типом данных канала. Обратите внимание, что при преобразовании вещественных данных в целые RasterIO() округляет в меньшую сторону, а если значение выходит за рамки допустимого диапазона, оно преобразуется в ближайшее допустимое значение. Это, например, означает, что при чтении 16-битных данных в буфер типа GDT_Byte все значения, превышающие 255 будут отображены в значение 255, масштабирования данных не произойдёт!

Параметры nBufXSize и nBufYSize задают размер буфера. При загрузке данных в полном разрешении он будет совпадать с размером окна. Однако для загрузки уменьшенного обзорного изображения размер буфера можно установить меньшим, чем размер окна. В этом случае RasterIO() будет использовать подходящие обзорные изображения (пирамиду) для более эффективного ввода/вывода.

Параметры nPixelSpace и nLineSpace обычно равны нулю, что приводит к использованию значений по умолчанию. Однако они могут быть использованы для управления доступом к буферу данных, давая возможность читать в буфер, который уже содержит другие данные, чередуя пиксели или строки.

3.5 Закрытие набора данных

Пожалуйста, постоянно помните, что объекты GDALRasterBand принадлежат к своему набору данных и они никогда не должны удаляться с помощью оператора delete языка C++. Наборы данных GDALDataset могут быть закрыты либо с помощью вызова функции GDALClose(), либо с использованием оператора delete для объекта GDALDataset. Любой вариант приведёт к корректному освобождению памяти и сбросу на диск всех незаписанных данных.

3.6 Способы создания файлов

Новые файлы в форматах, поддерживаемых GDAL, могут быть созданы в том случае, если драйвер формата поддерживает создание. Существует два основных способа создать файл: CreateCopy() и Create().

Способ CreateCopy предполагает вызов функции CreateCopy() с указанием требуемого драйвера выходного формата и передачей исходного набора данных, копия которого должна быть создана. Способ Create предполагает вызов метода Create() с указанием необходимого драйвера, а затем непосредственной записью всех метаданных и изображения соответствующими отдельными вызовами. Все драйверы, которые могут создавать новые файлы, поддерживают метод CreateCopy(), однако не все поддерживают метод Create().

Для того, чтобы определить, какой метод поддерживает конкретный драйвер, можно проверить метаданные DCAP_CREATE и DCAP_CREATECOPY у объекта драйвера. Убедитесь, что функция GDALAllRegister() была вызвана прежде, чем вызывать функцию GetDriverByName(). На примере:

На языке C++:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriver *poDriver;
    char **papszMetadata;

    poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);

    if( poDriver == NULL )
        exit( 1 );

    papszMetadata = poDriver->GetMetadata();
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Драйвер %s поддерживает метод Create().\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Драйвер %s поддерживает метод CreateCopy().\n", pszFormat );
```

На языке C:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriver hDriver = GDALGetDriverByName( pszFormat );
    char **papszMetadata;

    if( hDriver == NULL )
        exit( 1 );

    papszMetadata = GDALGetMetadata( hDriver, NULL );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Драйвер %s поддерживает метод Create().\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Драйвер %s поддерживает метод CreateCopy().\n", pszFormat );
```

На языке Python:

```
    format = "GTiff"
    driver = gdal.GetDriverByName( format )
    metadata = driver.GetMetadata()
    if metadata.has_key(gdal.DCAP_CREATE) \
       and metadata[gdal.DCAP_CREATE] == 'YES':
        print 'Драйвер %s поддерживает метод Create().' % format
    if metadata.has_key(gdal.DCAP_CREATECOPY) \
       and metadata[gdal.DCAP_CREATECOPY] == 'YES':
        print 'Драйвер %s поддерживает метод CreateCopy().' % format
```

Заметим, что некоторые драйверы могут только читать данные и не поддерживают ни метод Create(), ни CreateCopy().

3.7 Использование метода CreateCopy()

Использование метода GDALDriver::CreateCopy() тривиально, поскольку большая часть информации читается из входного набора данных. Тем не менее, Метод позволяет передавать параметры, специфичные для создаваемого выходного формата, а также имеет возможность отображать ход процесса копирования пользователю. Простейшая операция копирования из файла с именем pszSrcFilename в новый файл pszDstFilename с параметрами по умолчанию и в формате, драйвер которого был предварительно выбран, может выглядеть следующим образом:

На языке C++:

```
    GDALDataset *poSrcDS =
      (GDALDataset *) GDALOpen( pszSrcFilename, GA_ReadOnly );
    GDALDataset *poDstDS;

    poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                    NULL, NULL, NULL );
    if( poDstDS != NULL )
        delete poDstDS;
```

На языке C:

```
    GDALDatasetH hSrcDS = GDALOpen( pszSrcFilename, GA_ReadOnly );
    GDALDatasetH hDstDS;
```

```
    hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                             NULL, NULL, NULL );
    if( hDstDS != NULL )
        GDALClose( hDstDS );
```

На языке Python:

```
    src_ds = gdal.Open( src_filename )
    dst_ds = driver.CreateCopy( dst_filename, src_ds, 0 )
```

Заметим, что метод CreateCopy() возвращает набор данных, пригодный для записи и он должен быть соответствующим образом закрыт для завершения записи и сброса данных на диск. В случае языка Python это произойдёт автоматически, когда "dst_ds" выйдет из области видимости. Значение FALSE (или 0), используемое для параметра bStrict, следующего сразу за именем выходного набора данных в вызове CreateCopy(), показывает, что CreateCopy() должен завершиться без фатальной ошибки даже в случае, если создаваемый набор данных не может быть идентичен входному набору. Такое может произойти, например, поскольку выходной формат не поддерживает тип данных входного формата, или потому, что выходной формат не поддерживает географическую привязку.

Более сложный случай может включать указание параметров для создания выходного файла и использование индикатора хода работы:

На языке C++:

```
#include "cpl_string.h"
...
    const char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                    papzOptions, GDALTermProgress, NULL );
    if( poDstDS != NULL )
        delete poDstDS;
```

На языке C:

```
#include "cpl_string.h"
...
    const char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                             papzOptions, GDALTermProgres, NULL );
    if( hDstDS != NULL )
        GDALClose( hDstDS );
```

На языке Python:

```
    src_ds = gdal.Open( src_filename )
    dst_ds = driver.CreateCopy( dst_filename, src_ds, 0,
                                [ 'TILED=YES', 'COMPRESS=PACKBITS' ] )
```

3.8 Использование метода Create()

В тех случаях, когда вы не просто экспортируете существующий файл в новый формат, может быть необходимо применить метод GDALDriver::Create() (кроме этого несколько интересных вариантов возможны при использовании виртуальных файлов или файлов в памяти).

Метод Create() принимает список параметров, похожий на такой же для CreateCopy(), однако размеры изображения, число каналов и тип данных должен быть задан непосредственно.

На языке C++:

```
GDALDataset *poDstDS;
char **papszOptions = NULL;

poDstDS = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_Byte,
                            papszOptions );
```

На языке C:

```
GDALDatasetH hDstDS;
char **papszOptions = NULL;

hDstDS = GDALCreate( hDriver, pszDstFilename, 512, 512, 1, GDT_Byte,
                     papszOptions );
```

На языке Python:

```
dst_ds = driver.Create( dst_filename, 512, 512, 1, gdal.GDT_Byte )
```

Как только набор данных будет успешно создан, все необходимые метаданные и собственно изображение должны быть записаны в файл. Конкретная реализация очень сильно зависит от задачи, но в простейшем случае, включающем запись проекции, географической привязки и растрового изображения, может выглядеть так:

На языке C++:

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReference oSRS;
char *pszSRS_WKT = NULL;
GDALRasterBand *poBand;
GByte abyRaster[512*512];

poDstDS->SetGeoTransform( adfGeoTransform );

oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "NAD27" );
oSRS.exportToWkt( &pszSRS_WKT );
poDstDS->SetProjection( pszSRS_WKT );
CPLFree( pszSRS_WKT );

poBand = poDstDS->GetRasterBand(1);
poBand->RasterIO( GF_Write, 0, 0, 512, 512,
                  abyRaster, 512, 512, GDT_Byte, 0, 0 );

delete poDstDS;
```

На языке C:

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReferenceH hSRS;
char *pszSRS_WKT = NULL;
GDALRasterBandH hBand;
GByte abyRaster[512*512];

GDALSetGeoTransform( hDstDS, adfGeoTransform );

hSRS = OSRNewSpatialReference( NULL );
```

```
OSRSetUTM( hSRS, 11, TRUE );
OSRSetWellKnownGeogCS( hSRS, "NAD27" );
OSRExportToWkt( hSRS, &pszSRS_WKT );
OSRDestroySpatialReference( hSRS );

GDALSetProjection( hDstDS, pszSRS_WKT );
CPLFree( pszSRS_WKT );

hBand = GDALGetRasterBand( hDstDS, 1 );
GDALRasterIO( hBand, GF_Write, 0, 0, 512, 512,
              abyRaster, 512, 512, GDT_Byte, 0, 0 );

GDALClose( hDstDS );
```

На языке Python:

```
import Numeric, osr

dst_ds.SetGeoTransform( [ 444720, 30, 0, 3751320, 0, -30 ] )

srs = osr.SpatialReference()
srs.SetUTM( 11, 1 )
srs.SetWellKnownGeogCS( 'NAD27' )
dst_ds.SetProjection( srs.ExportToWkt() )

raster = Numeric.zeros( (512, 512) )
dst_ds.GetRasterBand(1).WriteArray( raster )
```