RU

Оглавление

1 Сборка GDAL из исходных текстов 1

2 Модель данных GDAL 3

2.1 Набор данных . . . 3

2.1.1 Система координат . . . 3

2.1.2 Аффинное преобразование геодезических координат . . . 4

2.1.3 Наземные контрольные точки (Ground Control Points, GCPs) . . . 4

2.1.4 База метаданных . . . 5

2.1.4.1 Домен вложенных наборов данных (SUBDATASETS) . . . 6

2.1.4.2 Домен структуры изображения (IMAGE_STRUCTURE) . . . 6

2.1.4.3 Домен RPC . . . 7

2.1.4.4 Домены "xml:" . . . 7

2.2 Растровый канал . . . 8

2.3 Таблица цветов . . . 9

2.4 Обзорные изображения . . . 9

3 Руководство по использованию GDAL 11

3.1 Открытие файла . . . 11

3.2 Чтение информации о наборе данных . . . 12

3.3 Чтение растрового канала . . . 13

3.4 Чтение растровых данных . . . 14

3.5 Закрытие набора данных . . . 15

3.6 Способы создания файлов . . . 15

3.7 Использование метода CreateCopy() . . . 16

3.8 Использование метода Create() . . . 18

4 GDAL - Geospatial Data Abstraction Library 21

4.1 Документация для пользователей . . . 21

4.2 Документация для разработчиков . . . 21

4.3 Конференция . . . 22

4.4 Список рассылки . . . 22

4.5 Отчёты об ошибках . . . 22

4.6 Интерфейсы для различных языков программирования 22

Глава 1

Сборка GDAL из исходных текстов

Этот раздел перенесён на страницы Вики: http://trac.osgeo.org/gdal/wiki/BuildHints

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 G↩ DALDataset::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 (dfGC↩ PX, 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 База метаданных

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

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

Некоторые форматы данных содержат собственную базу метаданных, драйверы других форматов могут отображать поля, специфичные для данного формата, в записи базы 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
```

Метаданные выделены в именованные группы, называемые доменами. Базовый домен не имеет имени (NULL или ""). Существует несколько специальных доменов, служащих определённым целям. Заметим, что в настоящий момент не существует способа перечислить все домены, доступные для данного объекта, однако приложение может проверить доступность любого домена, который оно умеет обрабатывать.

Следующие записи метаданных имеют чётко определённую семантику в базовом домене:

- AREA_OR_POINT: Принимает значение "Area" (по умолчанию) или "Point". Показывает, что значение пиксела либо соответствует среднему значению данной величины по всей области, занимаемой пикселем, либо представляет точечное значение в центре пиксела. Это не влияет на интерпретацию географической привязки, которая остаётся связанной с границами пиксела.
- NODATA_VALUES: Список разделённых пробелами значений обозначающих признак на отсутствия данных. Число пикселей в списке соответствует числу каналов в наборе данных. При таком способе указания отсутствия данных пиксель считается пустым тогда итолько тогда, когда если во всех каналах его значение совпадает соответствующему полю в списке NOD↩ ATA_VALUES. Эта запись пока не очень широко используется драйверами, алгоритмами и вспомогательными программами GDAL.
- MATRIX_REPRESENTATION: Показывает, в каком виде представлены матричные данные. Используется для наборов данных поляризационных радиолокаторов с синтезированной апертурой (Polarimetric SAR). Может принимать следующие значения:
  - SCATTERING
  - SYMMETRIZED_SCATTERING
  - COVARIANCE
  - SYMMETRIZED_COVARIANCE
  - COHERENCY
  - SYMMETRIZED_COHERENCY
  - KENNAUGH

  - SYMMETRIZED_KENNAUGH

- POLARMETRIC_INTERP: Эта запись определена для растровых каналов данных поляризационного радиолокатора с синтезированной апертурой (Polarimetric SAR). Показывает, какую позицию в заданном матричном представлении данных занимает данный канал. Например, для набора данных, представленных как матрица разброса, эта запись может принимать значения HH, HV, VH, VV. Если набор данных является матрицей ковариаций, эта запись будет принимать одно из значений: Covariance_11, Covariance_22, Covariance_33, Covariance_12, Covariance_13, Covariance_23 (поскольку сама матрица Эрмитова, то этих данных достаточно для её задания).

2.1.4.1 Домен вложенных наборов данных (SUBDATASETS)

Домен SUBDATASETS содержит список дочерних наборов данных. Обычно он используется для получения указателей на изображения, хранящиеся все вместе в едином файле (таком, как HDF или NITF). Например, файл формата NITF с четырьмя изображениями может иметь следующий список вложенных наборов данных:

```
SUBDATASET_1_NAME=NITF_IM:0:multi_1b.ntf
SUBDATASET_1_DESC=Image 1 of multi_1b.ntf
SUBDATASET_2_NAME=NITF_IM:1:multi_1b.ntf
SUBDATASET_2_DESC=Image 2 of multi_1b.ntf
SUBDATASET_3_NAME=NITF_IM:2:multi_1b.ntf
SUBDATASET_3_DESC=Image 3 of multi_1b.ntf
SUBDATASET_4_NAME=NITF_IM:3:multi_1b.ntf
SUBDATASET_4_DESC=Image 4 of multi_1b.ntf
SUBDATASET_5_NAME=NITF_IM:4:multi_1b.ntf
SUBDATASET_5_DESC=Image 5 of multi_1b.ntf
```

Значение записи _NAME — строка, которая может быть передана в функцию GDALOpen() для получения доступа к изображению. Запись _DESC предназначена для описания изображения в виде читаемой человеком строки и может быть показана пользователю для облегчения выбора.

2.1.4.2 Домен структуры изображения (IMAGE_STRUCTURE)

Метаданные в базовом домене содержат информацию, связанную с изображением, однако не содержат данных о том, как это изображение хранится на диске. Таким образом, эти метаданные могут быть спокойно скопированы вместе с остальным набором данными в новый формат. Тем не менее, некоторая важная информация тесно связана с конкретным форматом и способом хранения данных. Во избежание перенесения этой информации при копировании данных, она помещена в специальный домен, называемый IMAGE_STRUCTURE, который не должен слепо копироваться в другие форматы.

В настоящий момент спецификацией RFC 14 определены следующие специальные записи в домене IMAGE_STRUCTURE.

- COMPRESSION: Алгоритм сжатия, используемый для данного набора данных или канала. Не существует определённого каталога имён алгоритмов сжатия, однако если данный формат поддерживает параметр создания COMPRESSION, точно такой же список возможных значений должен быть использован и здесь.

- NBITS: Число бит на значение пиксела в канале или во всех каналах данного набора данных. Обычно присутствует только в случаях нестандартного числа бит для этого типа данных, например, если 1-битовый файл TIFF представлен в GDAL в виде GDT_Byte.

- INTERLEAVE: Применимо только к наборам данных и может принимать значения PIXEL, LINE или BAND. Может быть использовано в качестве подсказки для наиболее эффективного способа доступа к данным.

- PIXELTYPE: Может использоваться для каналов GDT_Byte (или соответсвующих наборов данных) и принимать значение SIGNEDBYTE, указывающее на то, что положительные значения от 128 до 255 должны интерпретироваться как отрицательные от -128 до -1 для приложений, умеющих обрабатывать байты со знаком.

2.1.4.3 Домен RPC

Домен RPC содержит метаданные, описывающие геометрическую модель изображения в виде полиномов с рациональными коэффициентами (Rational Polynomial Coefficient). Эта геометрическая модель может быть использована для пересчёта между координатами строка/столбец и географическими координатами. Модель определяется следующими записями:

- ERR_BIAS: Ошибка - уклон. Уклон среднеквадратичной ошибки в метрах по горизонтальной оси для всех точек изображения (-1.0 если неизвестно).
- ERR_RAND: Ошибка - случайная. Среднеквадратичная случайная ошибка в метрах по горизонтальной оси для каждой точки изображения (-1.0 если неизвестно).
- LINE_OFF: Смещение строки.
- SAMP_OFF: Смещение пиксела.
- LAT_OFF: Смещение геодезической широты.
- LONG_OFF: Смещение геодезической долготы.
- HEIGHT_OFF: Смещение геодезической высоты.
- LINE_SCALE: Масштаб строки.
- SAMP_SCALE: Масштаб пиксела.
- LAT_SCALE: Масштаб геодезической широты.
- LONG_SCALE: Масштаб геодезической долготы.
- HEIGHT_SCALE: Масштаб геодезической высоты.
- LINE_NUM_COEFF (1-20): Числители коэффициентов строк. Двадцать коэффициентов полинома в числителе уравнения rn (разделены пробелами).
- LINE_DEN_COEFF (1-20): Знаменатели коэффициентов строк. Двадцать коэффициентов полинома в знаменателе уравнения rn (разделены пробелами).
- SAMP_NUM_COEFF (1-20): Числители коэффициентов пикселей. Двадцать коэффициентов полинома в числителе уравнения cn (разделены пробелами).
- SAMP_DEN_COEFF (1-20): Знаменатели коэффициентов пикселей. Двадцать коэффициентов полинома в знаменателе уравнения cn (разделены пробелами).

Эти поля напрямую взяты из документа, предлагающего поддержку RPC в GeoTIFF (http↩ ://geotiff.maptools.org/rpc_prop.html), который в свою очередь следует определению NITF RP↩ C00B.

2.1.4.4 Домены "xml:"

Любой домен, чьё имя имеет префикс "xml:", является не обычной базой метаданных вида имя/значение, а единым документом XML представленным одной большой строкой.

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 маркер отсутсвия данных (см. также запись метаданных NODATA_VAL↩ UES для набора данных, содержащий маркеры отсутствия данных для всех каналов).
- Необязательный канал маски, маркирующий пикселы, в которых данные отсутствуют или прозрачны. См. обсуждение RFC 15: Band Masks.
- 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 GDALRaster↩ Band 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, n↩ XSize, 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(). В следующем примере мы запросим драйвер и проверим, поддерживает ли он методы Create() и/или CreateCopy().

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";
    GDALDriverH 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"
...
    char **papszOptions = NULL;

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

На языке C:

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

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

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 )
```

Глава 4

GDAL - Geospatial Data Abstraction Library

Select language: [English][Russian][Portuguese]

GDAL — это библиотека для работы с растровыми географическими форматами файлов данных. GDAL распространяется Open Source Geospatial Foundation на условиях лицензии X/MIT , то есть является проектом с открытым исходным кодом. Как библиотека GDAL предоставляет вызывающему приложению единую обобщённую модель данных (стр. 3) для всех поддерживаемых форматов файлов данных. Помимо этого в состав GDAL входит набор вспомогательных программ, вызываемых из командной строки, для преобразования и обработки данных. Описание выпуска GDAL/OGR 1.5.1 (март 2008) можно найти на странице новостей.

Связанная с GDAL библиотека OGR (исходные тексты которой поставляются вместе с GDAL) предоставляет сходные возможности для работы с векторными данными. На странице обновлений вы сможете найти описание версии 1.3.1, выпущенной в октябре 2005 г.

Главная страница: http://www.gdal.org
Загрузить: ftp at remotesensing.org, http at download.osgeo.org.

4.1 Документация для пользователей

- Вики - Документация и советы, предоставленные пользователями и разработчиками.
- Загрузить — Готовые исполняемые файлы.
- Поддерживаемые форматы.
- Вспомогательные программы GDAL.
- GDAL FAQ.
- Модель данных GDAL.
- Сообщество GDAL/OGR и управление проектом.
- Списки компаний, предоставляющих услуги с использованием GDAL.
- Благодарности.
- Программное обеспечение, использующее GDAL.

4.2 Документация для разработчиков

- Сборка GDAL из исходных текстов.
- Загрузить исходные тексты.

- Описание программного интерфейса
- Руководство по использованию GDAL
- Руководство по написанию драйверов GDAL
- Руководство по использованию интерфейса трансформации
- Руководство по OGRSpatialReference
- gdal.h - интерфейс GDAL для языка C.
- GDALDataset
- GDALRasterBand
- GDAL для Windows CE

4.3 Конференция

4.4 Список рассылки

Для отслеживания основных событий в разработке GDAL/OGR предназначен небольшой по объёму список рассылки gdal-announce.

Для обсуждения процесса разработки и вопросов по использованию GDAL и связанных с проектом технологий существует список рассылки gdal-dev@lists.osgeo.org. Подписаться и просмотреть архивы рассылки можно на специальной странице. Список рассылки также доступен в режиме только для чтения по протоколу NNTP news://news.gmane.org/gmane.comp.gis.gdal.devel и по HTTP http↩ ://news.gmane.org/gmane.comp.gis.gdal.devel.

Многих пользователей и разработчиков GDAL/OGR часто можно найти на IRC канале #gdal сервера irc.freenode.net.

4.5 Отчёты об ошибках

Об ошибках GDAL можно сообщить, и просмотреть при помощи системы Trac.

4.6 Интерфейсы для различных языков программирования

Вы можете использовать GDAL/OGR из следующих языков программирования:

- Perl
- Python
- VB6 Bindings (не использует SWIG)
- Интерфейс для R, созданный Timothy H. Keitt.
- Ruby
- Java
- C# / .Net