OGR API Tutorial

This document is intended to document using the OGR C++ classes to read and write data from a file. It is strongly advised that the read first review the OGR Architecture document describing the key classes and their roles in OGR.

It also includes code snippets for the corresponding functions in C and Python.

Reading From OGR

For purposes of demonstrating reading with OGR, we will construct a small utility for dumping point layers from an OGR data source to stdout in comma-delimited format.

Initially it is necessary to register all the format drivers that are desired. This is normally accomplished by calling **GDALAllRegister()** which registers all format drivers built into GDAL/OGR.

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();
```

In C :

```
#include "gdal.h"

int main()

{
    GDALAllRegister();
```

Next we need to open the input OGR datasource. Datasources can be files, RDBMSes, directories full of files, or even remote web services depending on the driver being used. However, the datasource name is always a single string. In this case we are hardcoded to open a particular shapefile. The second argument (GDAL_OF_VECTOR) tells the **OGROpen()** method that we want a vector driver to be use and that don't require update access. On failure NULL is returned, and we report an error.

In C++ :

```
GDALDataset       *poDS;

poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

In C :

```
GDALDatasetH hDS;

hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( hDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

A **GDALDataset** can potentially have many layers associated with it. The number of layers available can be queried with **GDALDataset::GetLayerCount()** and individual layers fetched by index using **GDALDataset::GetLayer()**. However, we will just fetch the layer by name.

In C++ :

```
OGRLayer  *poLayer;

poLayer = poDS->GetLayerByName( "point" );
```

In C :

```
OGRLayerH hLayer;

hLayer = GDALDatasetGetLayerByName( hDS, "point" );
```

Now we want to start reading features from the layer. Before we start we could assign an attribute or spatial filter to the layer to restrict the set of feature we get back, but for now we are interested in getting all features.

With GDAL 2.3 and C++11:

```
for( auto& poFeature: poLayer )
{
```

With GDAL 2.3 and C:

```
OGR_FOR_EACH_FEATURE_BEGIN(hFeature, hLayer)
{
```

If using older GDAL versions, while it isn't strictly necessary in this circumstance since we are starting fresh with the layer, it is often wise to call **OGRLayer::ResetReading()** to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using **OGRLayer::GetNextFeature()**. It will return NULL when we run out of features.

With GDAL < 2.3 and C++ :

```
OGRFeature *poFeature;

poLayer->ResetReading();
while( (poFeature = poLayer->GetNextFeature()) != NULL )
{
```

With GDAL < 2.3 and C :

```
OGRFeatureH hFeature;

OGR_L_ResetReading(hLayer);
while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
{
```

In order to dump all the attribute fields of the feature, it is helpful to get the **OGRFeatureDefn**. This is an object, associated with the layer, containing the definitions of all the fields. We loop over all the fields, and fetch and report the attributes based on their type.

With GDAL 2.3 and C++11:

```
for( auto&& oField: *poFeature )
{
    switch( oField.GetType() )
    {
        case OFTInteger:
            printf( "%d,", oField.GetInteger() );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ",", oField.GetInteger64() );
            break;
        case OFTReal:
            printf( "%.3f,", oField.GetDouble() );
            break;
        case OFTString:
            printf( "%s,", oField.GetString() );
            break;
        default:
            printf( "%s,", oField.GetAsString() );
            break;
    }
}
```

With GDAL < 2.3 and C++ :

```
OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
{
    OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

    switch( poFieldDefn->GetType() )
    {
        case OFTInteger:
            printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ",", poFeature->GetFieldAsInteger64( iField ) );
            break;
        case OFTReal:
            printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
            break;
        case OFTString:
            printf( "%s,", poFeature->GetFieldAsString(iField) );
            break;
        default:
            printf( "%s,", poFeature->GetFieldAsString(iField) );
            break;
    }
}
```

In C :

```
OGRFeatureDefnH hFDefn = OGR_L_GetLayerDefn(hLayer);
int iField;

for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
{
    OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

    switch( OGR_Fld_GetType(hFieldDefn) )
    {
        case OFTInteger:
            printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( hFeature, iField ) );
            break;
        case OFTReal:
            printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
            break;
        case OFTString:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
            break;
        default:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
            break;
    }
}
```

There are a few more field types than those explicitly handled above, but a reasonable representation of them can be fetched with the **OGRFeature::GetFieldAsString()** method. In fact we could shorten the above by using **OGRFeature::GetFieldAsString()** for all the types.

Next we want to extract the geometry from the feature, and write out the point geometry x and y. Geometries are returned as a generic **OGRGeometry** pointer. We then determine the specific geometry type, and if it is a point, we cast it to point and operate on it. If it is something else we write placeholders.

In C++ :

```
    OGRGeometry *poGeometry;

    poGeometry = poFeature->GetGeometryRef();
    if( poGeometry != NULL
            && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
    {
#if GDAL_VERSION_NUM >= GDAL_COMPUTE_VERSION(2,3,0)
        OGRPoint *poPoint = poGeometry->toPoint();
#else
        OGRPoint *poPoint = (OGRPoint *) poGeometry;
#endif

        printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
    }
    else
    {
        printf( "no point geometry\n" );
    }
```

In C :

```
OGRGeometryH hGeometry;

hGeometry = OGR_F_GetGeometryRef(hFeature);
if( hGeometry != NULL
        && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
{
    printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
}
else
{
    printf( "no point geometry\n" );
}
```

The **wkbFlatten()** macro is used above to convert the type for a wkbPoint25D (a point with a z coordinate) into the base 2D geometry type code (wkbPoint). For each 2D geometry type there is a corresponding 2.5D type code. The 2D and 2.5D geometry cases are handled by the same C++ class, so our code will handle 2D or 3D cases properly.

Starting with OGR 1.11, several geometry fields can be associated to a feature.

In C++ :

```
    OGRGeometry *poGeometry;
    int iGeomField;
    int nGeomFieldCount;

    nGeomFieldCount = poFeature->GetGeomFieldCount();
    for(iGeomField = 0; iGeomField < nGeomFieldCount; iGeomField ++ )
    {
        poGeometry = poFeature->GetGeomFieldRef(iGeomField);
        if( poGeometry != NULL
                && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
        {
#if GDAL_VERSION_NUM >= GDAL_COMPUTE_VERSION(2,3,0)
```

```
            OGRPoint *poPoint = poGeometry->toPoint();
#else
            OGRPoint *poPoint = (OGRPoint *) poGeometry;
#endif

            printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
        }
        else
        {
            printf( "no point geometry\n" );
        }
    }
```

In C :

```
OGRGeometryH hGeometry;
int iGeomField;
int nGeomFieldCount;

nGeomFieldCount = OGR_F_GetGeomFieldCount(hFeature);
for(iGeomField = 0; iGeomField < nGeomFieldCount; iGeomField ++ )
{
    hGeometry = OGR_F_GetGeomFieldRef(hFeature, iGeomField);
    if( hGeometry != NULL
            && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
    {
        printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0),
                OGR_G_GetY(hGeometry, 0) );
    }
    else
    {
        printf( "no point geometry\n" );
    }
}
```

In Python:

```
nGeomFieldCount = feat.GetGeomFieldCount()
for iGeomField in range(nGeomFieldCount):
    geom = feat.GetGeomFieldRef(iGeomField)
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print "%.3f, %.3f" % ( geom.GetX(), geom.GetY() )
    else:
        print "no point geometry\n"
```

Note that **OGRFeature::GetGeometryRef()** and **OGRFeature::GetGeomFieldRef()** return a pointer to the internal geometry owned by the **OGRFeature**. There we don't actually deleted the return geometry.

With GDAL 2.3 and C++11, the looping over features is simply terminated by a closing curly bracket.

```
}
```

With GDAL 2.3 and C, the looping over features is simply terminated by the following.

```
}
OGR_FOR_EACH_FEATURE_END(hFeature)
```

For GDAL < 2.3, as the **OGRLayer::GetNextFeature()** method returns a copy of the feature that is now owned by us. So at the end of use we must free the feature. We could just "delete" it, but this can cause problems in windows builds where the GDAL DLL has a different "heap" from the main program. To be on the safe side we use a GDAL function to delete the feature.

In C++ :

```
    OGRFeature::DestroyFeature( poFeature );
}
```

In C :

```
    OGR_F_Destroy( hFeature );
}
```

The **OGRLayer** returned by **GDALDataset::GetLayerByName()** is also a reference to an internal layer owned by the **GDALDataset** so we don't need to delete it. But we do need to delete the datasource in order to close the input file. Once again we do this with a custom delete method to avoid special win32 heap issues.

In C/C++ :

```
    GDALClose( poDS );
}
```

All together our program looks like this.

With GDAL 2.3 and C++11 :

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();

    GDALDatasetUniquePtr poDS(GDALDataset::Open( "point.shp", GDAL_OF_VECTOR));
    if( poDS == nullptr )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    for( const OGRLayer* poLayer: poDS->GetLayers() )
    {
        for( const auto& poFeature: *poLayer )
        {
            for( const auto& oField: *poFeature )
            {
                switch( oField.GetType() )
                {
                    case OFTInteger:
                        printf( "%d,", oField.GetInteger() );
                        break;
                    case OFTInteger64:
                        printf( CPL_FRMT_GIB ",", oField.GetInteger64() );
                        break;
                    case OFTReal:
                        printf( "%.3f,", oField.GetDouble() );
                        break;
                    case OFTString:
                        printf( "%s,", oField.GetString() );
                        break;
                    default:
                        printf( "%s,", oField.GetAsString() );
                        break;
                }
            }

            const OGRGeometry *poGeometry = poFeature->GetGeometryRef();
            if( poGeometry != nullptr
                    && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
            {
                const OGRPoint *poPoint = poGeometry->toPoint();

                printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
            }
            else
            {
                printf( "no point geometry\n" );
            }
        }
    }
    return 0;
}
```

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();

    GDALDataset *poDS = static_cast<GDALDataset*>(
        GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL ));
    if( poDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    OGRLayer  *poLayer = poDS->GetLayerByName( "point" );
    OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();

    poLayer->ResetReading();
    OGRFeature *poFeature;
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

            switch( poFieldDefn->GetType() )
            {
                case OFTInteger:
                    printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB ",", poFeature->GetFieldAsInteger64( iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
                    break;
                case OFTString:
                    printf( "%s,", poFeature->GetFieldAsString(iField) );
                    break;
                default:
                    printf( "%s,", poFeature->GetFieldAsString(iField) );
                    break;
            }
        }

        OGRGeometry *poGeometry = poFeature->GetGeometryRef();
        if( poGeometry != NULL
                && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
        {
            OGRPoint *poPoint = (OGRPoint *) poGeometry;

            printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
        }
        else
        {
            printf( "no point geometry\n" );
        }
        OGRFeature::DestroyFeature( poFeature );
    }

    GDALClose( poDS );
}
```

In C :

```
#include "gdal.h"

int main()

{
    GDALAllRegister();

    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFeatureH hFeature;
    OGRFeatureDefnH hFDefn;

    hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
    if( hDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetGetLayerByName( hDS, "point" );
    hFDefn = OGR_L_GetLayerDefn(hLayer);

    OGR_L_ResetReading(hLayer);
    while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
    {
        int iField;
        OGRGeometryH hGeometry;

        for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
        {
            OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

            switch( OGR_Fld_GetType(hFieldDefn) )
            {
                case OFTInteger:
                    printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( hFeature, iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
                    break;
                case OFTString:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
                    break;
                default:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
                    break;
            }
        }

        hGeometry = OGR_F_GetGeometryRef(hFeature);
        if( hGeometry != NULL
            && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
        {
            printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
        }
        else
        {
            printf( "no point geometry\n" );
        }

        OGR_F_Destroy( hFeature );
    }

    GDALClose( hDS );
}
```

In Python:

```
import sys
from osgeo import gdal

ds = gdal.OpenEx( "point.shp", gdal.OF_VECTOR )
if ds is None:
    print "Open failed.\n"
    sys.exit( 1 )

lyr = ds.GetLayerByName( "point" )

lyr.ResetReading()
```

```
for feat in lyr:

    feat_defn = lyr.GetLayerDefn()
    for i in range(feat_defn.GetFieldCount()):
        field_defn = feat_defn.GetFieldDefn(i)

        # Tests below can be simplified with just :
        # print feat.GetField(i)
        if field_defn.GetType() == ogr.OFTInteger or field_defn.GetType() == ogr.OFTInteger64:
            print "%d" % feat.GetFieldAsInteger64(i)
        elif field_defn.GetType() == ogr.OFTReal:
            print "%.3f" % feat.GetFieldAsDouble(i)
        elif field_defn.GetType() == ogr.OFTString:
            print "%s" % feat.GetFieldAsString(i)
        else:
            print "%s" % feat.GetFieldAsString(i)

    geom = feat.GetGeometryRef()
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print "%.3f, %.3f" % ( geom.GetX(), geom.GetY() )
    else:
        print "no point geometry\n"

ds = None
```

Writing To OGR

As an example of writing through OGR, we will do roughly the opposite of the above. A short program that reads comma separated values from input text will be written to a point shapefile via OGR.

As usual, we start by registering all the drivers, and then fetch the Shapefile driver as we will need it to create our output file.

In C++ :

```
#include "ogrsf_frmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = GetGDALDriverManager()->GetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
```

In C :

```
#include "ogr_api.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = (GDALDriver*) GDALGetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
```

Next we create the datasource. The ESRI Shapefile driver allows us to create a directory full of shapefiles, or a single shapefile as a datasource. In this case we will explicitly create a single file by including the extension in the name. Other drivers behave differently. The second, third, fourth and fifth argument are related to raster dimensions (in case the driver has raster capabilities). The last argument to the call is a list of option values, but we will just be using defaults in this case. Details of the options supported are also format specific.

In C ++ :

```
GDALDataset *poDS;

poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
if( poDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

In C :

```
GDALDatasetH hDS;

hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
if( hDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

Now we create the output layer. In this case since the datasource is a single file, we can only have one layer. We pass wkbPoint to specify the type of geometry supported by this layer. In this case we aren't passing any coordinate system information or other special layer creation options.

In C++ :

```
OGRLayer *poLayer;

poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
if( poLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRLayerH hLayer;

hLayer = GDALDatasetCreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
if( hLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

Now that the layer exists, we need to create any attribute fields that should appear on the layer. Fields must be added to the layer before any features are written. To create a field we initialize an **OGRField** object with the information about the field. In the case of Shapefiles, the field width and precision is significant in the creation of the output .dbf file, so we set it specifically, though generally the defaults are OK. For this example we will just have one attribute, a name string associated with the x,y point.

Note that the template **OGRField** we pass to CreateField() is copied internally. We retain ownership of the object.

In C++:

```
OGRFieldDefn oField( "Name", OFTString );

oField.SetWidth(32);

if( poLayer->CreateField( &oField ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}
```

In C:

```
OGRFieldDefnH hFieldDefn;

hFieldDefn = OGR_Fld_Create( "Name", OFTString );

OGR_Fld_SetWidth( hFieldDefn, 32);

if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}

OGR_Fld_Destroy(hFieldDefn);
```

The following snipping loops reading lines of the form "x,y,name" from stdin, and parsing them.

In C++ and in C :

```
double x, y;
char szName[33];

while( !feof(stdin)
       && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
{
```

To write a feature to disk, we must create a local **OGRFeature**, set attributes and attach geometry before trying to write it to the layer. It is imperative that this feature be instantiated from the **OGRFeatureDefn** associated with the layer it will be written to.

In C++ :

```
OGRFeature *poFeature;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
poFeature->SetField( "Name", szName );
```

In C :

```
OGRFeatureH hFeature;

hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );
```

We create a local geometry object, and assign its copy (indirectly) to the feature. The **OGRFeature::SetGeometryDirectly()** differs from **OGRFeature::SetGeometry()** in that the direct method gives ownership of the geometry to the feature. This is generally more efficient as it avoids an extra deep object copy of the geometry.

In C++ :

```
OGRPoint pt;
pt.setX( x );
pt.setY( y );

poFeature->SetGeometry( &pt );
```

In C :

```
OGRGeometryH hPt;
hPt = OGR_G_CreateGeometry(wkbPoint);
OGR_G_SetPoint_2D(hPt, 0, x, y);

OGR_F_SetGeometry( hFeature, hPt );
OGR_G_DestroyGeometry(hPt);
```

Now we create a feature in the file. The **OGRLayer::CreateFeature()** does not take ownership of our feature so we clean it up when done with it.

In C++ :

```
     if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
     {
         printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
     }

     OGRFeature::DestroyFeature( poFeature );
}
```

In C :

```
     if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
     {
         printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
     }

     OGR_F_Destroy( hFeature );
}
```

Finally we need to close down the datasource in order to ensure headers are written out in an orderly way and all resources are recovered.

In C/C++ :

```
    GDALClose( poDS );
}
```

The same program all in one block looks like this:

In C++ :

```
#include "ogrsf_frmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = GetGDALDriverManager()->GetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    GDALDataset *poDS;

    poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( poDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    OGRLayer *poLayer;

    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    OGRFieldDefn oField( "Name", OFTString );

    oField.SetWidth(32);

    if( poLayer->CreateField( &oField ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    double x, y;
    char szName[33];

    while( !feof(stdin)
           && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeature *poFeature;

        poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
        poFeature->SetField( "Name", szName );

        OGRPoint pt;

        pt.setX( x );
        pt.setY( y );

        poFeature->SetGeometry( &pt );

        if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
        {
            printf( "Failed to create feature in shapefile.\n" );
            exit( 1 );
        }

        OGRFeature::DestroyFeature( poFeature );
    }

    GDALClose( poDS );
}
```

In C :

```
#include "gdal.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriverH hDriver;
    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFieldDefnH hFieldDefn;
    double x, y;
    char szName[33];

    GDALAllRegister();

    hDriver = GDALGetDriverByName( pszDriverName );
    if( hDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( hDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetCreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
    if( hLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    hFieldDefn = OGR_Fld_Create( "Name", OFTString );

    OGR_Fld_SetWidth( hFieldDefn, 32);

    if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    OGR_Fld_Destroy(hFieldDefn);

    while( !feof(stdin)
           && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeatureH hFeature;
        OGRGeometryH hPt;

        hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
        OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );

        hPt = OGR_G_CreateGeometry(wkbPoint);
        OGR_G_SetPoint_2D(hPt, 0, x, y);

        OGR_F_SetGeometry( hFeature, hPt );
        OGR_G_DestroyGeometry(hPt);

        if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
        {
           printf( "Failed to create feature in shapefile.\n" );
           exit( 1 );
        }

        OGR_F_Destroy( hFeature );
   }

   GDALClose( hDS );
}
```

In Python :

```
import sys
from osgeo import gdal
from osgeo import ogr
import string

driverName = "ESRI Shapefile"
drv = gdal.GetDriverByName( driverName )
if drv is None:
    print "%s driver not available.\n" % driverName
    sys.exit( 1 )

ds = drv.Create( "point_out.shp", 0, 0, 0, gdal.GDT_Unknown )
if ds is None:
    print "Creation of output file failed.\n"
    sys.exit( 1 )

lyr = ds.CreateLayer( "point_out", None, ogr.wkbPoint )
```

```
if lyr is None:
    print "Layer creation failed.\n"
    sys.exit( 1 )

field_defn = ogr.FieldDefn( "Name", ogr.OFTString )
field_defn.SetWidth( 32 )

if lyr.CreateField ( field_defn ) != 0:
    print "Creating Name field failed.\n"
    sys.exit( 1 )

# Expected format of user input: x y name
linestring = raw_input()
linelist = string.split(linestring)

while len(linelist) == 3:
    x = float(linelist[0])
    y = float(linelist[1])
    name = linelist[2]

    feat = ogr.Feature( lyr.GetLayerDefn())
    feat.SetField( "Name", name )

    pt = ogr.Geometry(ogr.wkbPoint)
    pt.SetPoint_2D(0, x, y)

    feat.SetGeometry(pt)

    if lyr.CreateFeature(feat) != 0:
        print "Failed to create feature in shapefile.\n"
        sys.exit( 1 )

    feat.Destroy()

    linestring = raw_input()
    linelist = string.split(linestring)

ds = None
```

Starting with OGR 1.11, several geometry fields can be associated to a feature. This capability is just available for a few file formats, such as PostGIS.

To create such datasources, geometry fields must be first created. Spatial reference system objects can be associated to each geometry field.

In C++ :

```
OGRGeomFieldDefn oPointField( "PointField", wkbPoint );
OGRSpatialReference* poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(4326);
oPointField.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGRGeomFieldDefn oFieldPoint2( "PointField2", wkbPoint );
poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(32631);
oPointField2.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRGeomFieldDefnH hPointField;
OGRGeomFieldDefnH hPointField2;
OGRSpatialReferenceH hSRS;

hPointField = OGR_GFld_Create( "PointField", wkbPoint );
hSRS = OSRNewSpatialReference( NULL );
OSRImportFromEPSG(hSRS, 4326);
OGR_GFld_SetSpatialRef(hPointField, hSRS);
OSRRelease(hSRS);

if( OGR_L_CreateGeomField( hLayer, hPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField );

hPointField2 = OGR_GFld_Create( "PointField2", wkbPoint );
OSRImportFromEPSG(hSRS, 32631);
OGR_GFld_SetSpatialRef(hPointField2, hSRS);
OSRRelease(hSRS);

if( OGR_L_CreateGeomField( hLayer, hPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField2 );
```

To write a feature to disk, we must create a local **OGRFeature**, set attributes and attach geometries before trying to write it to the layer. It is imperative that this feature be instantiated from the **OGRFeatureDefn** associated with the layer it will be written to.

In C++ :

```
OGRFeature *poFeature;
OGRGeometry *poGeometry;
char* pszWKT;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );

pszWKT = (char*) "POINT (2 49)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField", poGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField2", poGeometry );

if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature.\n" );
    exit( 1 );
}

OGRFeature::DestroyFeature( poFeature );
```

In C :

```
OGRFeatureH hFeature;
OGRGeometryH hGeometry;
char* pszWKT;

poFeature = OGR_F_Create( OGR_L_GetLayerDefn(hLayer) );

pszWKT = (char*) "POINT (2 49)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField"), hGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField2"), hGeometry );
```

```
if( OGR_L_CreateFeature( hFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature.\n" );
    exit( 1 );
}

OGR_F_Destroy( hFeature );
```

In Python :

```
feat = ogr.Feature( lyr.GetLayerDefn() )

feat.SetGeomFieldDirectly( "PointField",
    ogr.CreateGeometryFromWkt( "POINT (2 49)" ) )
feat.SetGeomFieldDirectly( "PointField2",
    ogr.CreateGeometryFromWkt( "POINT (500000 4500000)" ) )

if lyr.CreateFeature( feat ) != 0 )
{
    print( "Failed to create feature.\n" );
    sys.exit( 1 );
}
```