OGR

Contents

1 OGR Simple Feature Library **1**

2 OGR API Tutorial **3**

2.1 Reading From OGR . . . 3

2.2 Writing To OGR . . . 8

3 OGR Architecture **15**

3.1 Class Overview . . . 15

3.2 Geometry . . . 15

3.3 Spatial Reference . . . 16

3.4 Feature / Feature Definition . . . 16

3.5 Layer . . . 17

3.6 Data Source . . . 17

3.7 Drivers . . . 17

4 OGR Driver Implementation Tutorial **19**

4.1 Overall Approach . . . 19

4.2 Contents . . . 19

4.3 Implementing OGRSFDriver . . . 19

4.4 Basic Read Only Data Source . . . 21

4.5 Read Only Layer . . . 22

5 OGR SQL **25**

5.1 SELECT . . . 25

5.1.1 Field List Operators . . . 26

5.1.1.1 Functions . . . 26

5.1.1.2 Using the field name alias . . . 27

5.1.1.3 Changing the type of the fields . . . 27

5.1.2 WHERE . . . 27

5.1.3 WHERE Limitations . . . 28

5.1.4 ORDER BY . . . 28

5.1.5 JOINs . . . 29

5.1.6 JOIN Limitations . . . 30

5.2 SPECIAL FIELDS . . . 30
5.2.1 FID . . . 30
5.2.2 OGR_GEOMETRY . . . 30
5.2.3 OGR_GEOM_WKT . . . 30
5.2.4 OGR_GEOM_AREA . . . 31
5.2.5 OGR_STYLE . . . 31
5.3 CREATE INDEX . . . 31
5.3.1 Index Limitations . . . 31
5.4 DROP INDEX . . . 31
5.5 ALTER TABLE . . . 31
5.6 DROP TABLE . . . 32
5.7 ExecuteSQL() . . . 32
5.8 Non-OGR SQL . . . 32

6 OGR Projections Tutorial **33**
6.1 Introduction . . . 33
6.2 Defining a Geographic Coordinate System . . . 33
6.3 Defining a Projected Coordinate System . . . 34
6.4 Querying Coordinate System . . . 35
6.5 Coordinate Transformation . . . 36
6.6 Alternate Interfaces . . . 36
6.7 Internal Implementation . . . 38

7 Deprecated List **39**

8 Class Index **41**
8.1 Class Hierarchy . . . 41

9 Class Index **45**
9.1 Class List . . . 45

10 File Index **49**
10.1 File List . . . 49

11 Class Documentation **51**
11.1 _CPLHashSet Struct Reference . . . 51
11.2 _CPLList Struct Reference . . . 51
11.2.1 Detailed Description . . . 51
11.2.2 Member Data Documentation . . . 51
11.2.2.1 pData . . . 51
11.2.2.2 psNext . . . 51
11.3 _CPLQuadTree Struct Reference . . . 52

11.4 _QuadTreeNode Struct Reference . . . 52
11.5 _sPolyExtended Struct Reference . . . 52
11.6 CachedConnection Struct Reference . . . 52
11.7 CachedDirList Struct Reference . . . 52
11.8 CachedFileProp Struct Reference . . . 52
11.9 CachedRegion Struct Reference . . . 52
11.10 CPLErrorContext Struct Reference . . . 53
11.11 CPLHTTPResult Struct Reference . . . 53
11.11.1 Detailed Description . . . 53
11.11.2 Member Data Documentation . . . 53
11.11.2.1 nDataLen . . . 53
11.11.2.2 nMimePartCount . . . 53
11.11.2.3 nStatus . . . 53
11.11.2.4 pabyData . . . 53
11.11.2.5 papszHeaders . . . 54
11.11.2.6 pasMimePart . . . 54
11.11.2.7 pszContentType . . . 54
11.11.2.8 pszErrBuf . . . 54
11.12 CPLKeywordParser Class Reference . . . 54
11.13 CPLLocaleC Class Reference . . . 54
11.14 CPLMimePart Struct Reference . . . 54
11.14.1 Detailed Description . . . 55
11.14.2 Member Data Documentation . . . 55
11.14.2.1 nDataLen . . . 55
11.14.2.2 pabyData . . . 55
11.14.2.3 papszHeaders . . . 55
11.15 CPLMutexHolder Class Reference . . . 55
11.16 CPLODBCDriverInstaller Class Reference . . . 55
11.16.1 Detailed Description . . . 55
11.16.2 Member Function Documentation . . . 56
11.16.2.1 InstallDriver . . . 56
11.16.2.2 RemoveDriver . . . 56
11.17 CPLODBCSession Class Reference . . . 56
11.17.1 Detailed Description . . . 56
11.17.2 Member Function Documentation . . . 57
11.17.2.1 EstablishSession . . . 57
11.17.2.2 GetLastError . . . 57
11.18 CPLODBCStatement Class Reference . . . 57
11.18.1 Detailed Description . . . 58
11.18.2 Member Function Documentation . . . 58

11.18.2.1 Append . . . 58
11.18.2.2 Append . . . 58
11.18.2.3 Append . . . 58
11.18.2.4 AppendEscaped . . . 59
11.18.2.5 Appendf . . . 59
11.18.2.6 Clear . . . 59
11.18.2.7 DumpResult . . . 59
11.18.2.8 ExecuteSQL . . . 60
11.18.2.9 Fetch . . . 60
11.18.2.10 GetColCount . . . 60
11.18.2.11 GetColData . . . 60
11.18.2.12 GetColData . . . 61
11.18.2.13 GetColId . . . 61
11.18.2.14 GetColName . . . 61
11.18.2.15 GetColNullable . . . 62
11.18.2.16 GetColPrecision . . . 62
11.18.2.17 GetColSize . . . 62
11.18.2.18 GetColType . . . 62
11.18.2.19 GetColTypeName . . . 63
11.18.2.20 GetColumns . . . 63
11.18.2.21 GetPrimaryKeys . . . 63
11.18.2.22 GetTables . . . 64
11.18.2.23 GetTypeMapping . . . 64
11.18.2.24 GetTypeName . . . 64
11.19 CPLRectObj Struct Reference . . . 65
11.20 CPLSharedFileInfo Struct Reference . . . 65
11.21 CPLStdCallThreadInfo Struct Reference . . . 65
11.22 CPLString Class Reference . . . 65
11.22.1 Detailed Description . . . 65
11.22.2 Member Function Documentation . . . 65
11.22.2.1 FormatC . . . 65
11.22.2.2 ifind . . . 66
11.22.2.3 ifind . . . 66
11.22.2.4 tolower . . . 66
11.22.2.5 toupper . . . 67
11.22.2.6 Trim . . . 67
11.23 CPLStringList Class Reference . . . 67
11.23.1 Detailed Description . . . 67
11.23.2 Constructor & Destructor Documentation . . . 68
11.23.2.1 CPLStringList . . . 68

11.23.3 Member Function Documentation . . . 68
11.23.3.1 AddNameValue . . . 68
11.23.3.2 AddString . . . 68
11.23.3.3 AddStringDirectly . . . 68
11.23.3.4 Assign . . . 69
11.23.3.5 Clear . . . 69
11.23.3.6 Count . . . 69
11.23.3.7 FetchBoolean . . . 69
11.23.3.8 FetchNameValue . . . 69
11.23.3.9 FetchNameValueDef . . . 70
11.23.3.10 FindName . . . 70
11.23.3.11 InsertString . . . 70
11.23.3.12 InsertStringDirectly . . . 71
11.23.3.13 operator[] . . . 71
11.23.3.14 SetNameValue . . . 71
11.23.3.15 Sort . . . 71
11.23.3.16 StealList . . . 72
11.24 CPLXMLNode Struct Reference . . . 72
11.24.1 Detailed Description . . . 72
11.24.2 Member Data Documentation . . . 72
11.24.2.1 eType . . . 72
11.24.2.2 psChild . . . 73
11.24.2.3 psNext . . . 73
11.24.2.4 pszValue . . . 73
11.25 ctb Struct Reference . . . 73
11.26 curfile_info Struct Reference . . . 73
11.27 DefaultCSVFileNameTLS Struct Reference . . . 74
11.28 errHandler Struct Reference . . . 74
11.29 file_in_zip_read_info_s Struct Reference . . . 74
11.30 FindFileTLS Struct Reference . . . 74
11.31 GZipSnapshot Struct Reference . . . 74
11.32 linkedlist_data_s Struct Reference . . . 74
11.33 linkedlist_datablock_internal_s Struct Reference . . . 74
11.34 OGR_SRSNode Class Reference . . . 75
11.34.1 Detailed Description . . . 75
11.34.2 Constructor & Destructor Documentation . . . 75
11.34.2.1 OGR_SRSNode . . . 75
11.34.3 Member Function Documentation . . . 75
11.34.3.1 AddChild . . . 75
11.34.3.2 applyRemapper . . . 76

11.34.3.3 Clone . . . 76
11.34.3.4 DestroyChild . . . 77
11.34.3.5 exportToWkt . . . 77
11.34.3.6 FindChild . . . 77
11.34.3.7 GetChild . . . 77
11.34.3.8 GetChildCount . . . 78
11.34.3.9 GetNode . . . 78
11.34.3.10 GetValue . . . 79
11.34.3.11 importFromWkt . . . 79
11.34.3.12 InsertChild . . . 79
11.34.3.13 MakeValueSafe . . . 80
11.34.3.14 SetValue . . . 80
11.34.3.15 StripNodes . . . 80
11.35 ogr_style_param Struct Reference . . . 80
11.36 ogr_style_value Struct Reference . . . 81
11.37 OGRAttrIndex Class Reference . . . 81
11.38 OGRCoordinateTransformation Class Reference . . . 81
11.38.1 Detailed Description . . . 81
11.38.2 Member Function Documentation . . . 82
11.38.2.1 DestroyCT . . . 82
11.38.2.2 GetSourceCS . . . 82
11.38.2.3 GetTargetCS . . . 82
11.38.2.4 Transform . . . 82
11.38.2.5 TransformEx . . . 83
11.39 OGRCurve Class Reference . . . 83
11.39.1 Detailed Description . . . 84
11.39.2 Member Function Documentation . . . 84
11.39.2.1 EndPoint . . . 84
11.39.2.2 get_IsClosed . . . 84
11.39.2.3 get_Length . . . 84
11.39.2.4 StartPoint . . . 85
11.39.2.5 Value . . . 85
11.40 OGRDataSource Class Reference . . . 85
11.40.1 Detailed Description . . . 86
11.40.2 Member Function Documentation . . . 87
11.40.2.1 CopyLayer . . . 87
11.40.2.2 CreateLayer . . . 87
11.40.2.3 DeleteLayer . . . 88
11.40.2.4 Dereference . . . 88
11.40.2.5 DestroyDataSource . . . 88

11.40.2.6 ExecuteSQL . . . 89
11.40.2.7 GetDriver . . . 89
11.40.2.8 GetLayer . . . 89
11.40.2.9 GetLayerByName . . . 90
11.40.2.10 GetLayerCount . . . 90
11.40.2.11 GetName . . . 90
11.40.2.12 GetRefCount . . . 90
11.40.2.13 GetStyleTable . . . 91
11.40.2.14 GetSummaryRefCount . . . 91
11.40.2.15 Reference . . . 91
11.40.2.16 Release . . . 91
11.40.2.17 ReleaseResultSet . . . 91
11.40.2.18 SetDriver . . . 92
11.40.2.19 SetStyleTable . . . 92
11.40.2.20 SetStyleTableDirectly . . . 92
11.40.2.21 SyncToDisk . . . 92
11.40.2.22 TestCapability . . . 93
11.41 OGREnvelope Class Reference . . . 93
11.41.1 Detailed Description . . . 93
11.42 OGREnvelope3D Class Reference . . . 94
11.42.1 Detailed Description . . . 94
11.43 OGRFeature Class Reference . . . 94
11.43.1 Detailed Description . . . 96
11.43.2 Constructor & Destructor Documentation . . . 96
11.43.2.1 OGRFeature . . . 96
11.43.3 Member Function Documentation . . . 96
11.43.3.1 Clone . . . 96
11.43.3.2 CreateFeature . . . 97
11.43.3.3 DestroyFeature . . . 97
11.43.3.4 DumpReadable . . . 97
11.43.3.5 Equal . . . 98
11.43.3.6 GetDefnRef . . . 98
11.43.3.7 GetFID . . . 98
11.43.3.8 GetFieldAsBinary . . . 99
11.43.3.9 GetFieldAsDateTime . . . 99
11.43.3.10 GetFieldAsDouble . . . 99
11.43.3.11 GetFieldAsDoubleList . . . 100
11.43.3.12 GetFieldAsInteger . . . 100
11.43.3.13 GetFieldAsIntegerList . . . 100
11.43.3.14 GetFieldAsString . . . 101

11.43.3.15GetFieldAsStringList . . . 101
11.43.3.16GetFieldCount . . . 102
11.43.3.17GetFieldDefnRef . . . 102
11.43.3.18GetFieldIndex . . . 102
11.43.3.19GetGeometryRef . . . 102
11.43.3.20GetRawFieldRef . . . 103
11.43.3.21GetStyleString . . . 103
11.43.3.22IsFieldSet . . . 103
11.43.3.23SetFID . . . 104
11.43.3.24SetField . . . 104
11.43.3.25SetField . . . 104
11.43.3.26SetField . . . 105
11.43.3.27SetField . . . 105
11.43.3.28SetField . . . 105
11.43.3.29SetField . . . 105
11.43.3.30SetField . . . 106
11.43.3.31SetField . . . 106
11.43.3.32SetField . . . 106
11.43.3.33SetFrom . . . 107
11.43.3.34SetFrom . . . 107
11.43.3.35SetGeometry . . . 107
11.43.3.36SetGeometryDirectly . . . 108
11.43.3.37SetStyleString . . . 108
11.43.3.38SetStyleStringDirectly . . . 108
11.43.3.39StealGeometry . . . 109
11.43.3.40UnsetField . . . 109
11.44OGRFeatureDefn Class Reference . . . 109
11.44.1 Detailed Description . . . 110
11.44.2 Constructor & Destructor Documentation . . . 110
11.44.2.1 OGRFeatureDefn . . . 110
11.44.3 Member Function Documentation . . . 111
11.44.3.1 AddFieldDefn . . . 111
11.44.3.2 Clone . . . 111
11.44.3.3 DeleteFieldDefn . . . 111
11.44.3.4 Dereference . . . 112
11.44.3.5 GetFieldCount . . . 112
11.44.3.6 GetFieldDefn . . . 112
11.44.3.7 GetFieldIndex . . . 112
11.44.3.8 GetGeomType . . . 113
11.44.3.9 GetName . . . 113

11.44.3.10 GetReferenceCount . . . 113
11.44.3.11 IsGeometryIgnored . . . 113
11.44.3.12 IsStyleIgnored . . . 114
11.44.3.13 Reference . . . 114
11.44.3.14 ReorderFieldDefns . . . 114
11.44.3.15 SetGeometryIgnored . . . 114
11.44.3.16 SetGeomType . . . 115
11.44.3.17 SetStyleIgnored . . . 115
11.45 OGRFeatureQuery Class Reference . . . 115
11.45.1 Member Function Documentation . . . 115
11.45.1.1 GetUsedFields . . . 115
11.46 OGRField Union Reference . . . 116
11.46.1 Detailed Description . . . 116
11.47 OGRFieldDefn Class Reference . . . 116
11.47.1 Detailed Description . . . 117
11.47.2 Constructor & Destructor Documentation . . . 117
11.47.2.1 OGRFieldDefn . . . 117
11.47.2.2 OGRFieldDefn . . . 117
11.47.3 Member Function Documentation . . . 117
11.47.3.1 GetFieldTypeName . . . 117
11.47.3.2 GetJustify . . . 118
11.47.3.3 GetNameRef . . . 118
11.47.3.4 GetPrecision . . . 118
11.47.3.5 GetType . . . 118
11.47.3.6 GetWidth . . . 119
11.47.3.7 IsIgnored . . . 119
11.47.3.8 Set . . . 119
11.47.3.9 SetDefault . . . 119
11.47.3.10 SetIgnored . . . 120
11.47.3.11 SetJustify . . . 120
11.47.3.12 SetName . . . 120
11.47.3.13 SetPrecision . . . 120
11.47.3.14 SetType . . . 120
11.47.3.15 SetWidth . . . 121
11.48 OGRGenSQLResultsLayer Class Reference . . . 121
11.48.1 Member Function Documentation . . . 122
11.48.1.1 GetExtent . . . 122
11.48.1.2 GetFeature . . . 122
11.48.1.3 GetFeatureCount . . . 123
11.48.1.4 GetLayerDefn . . . 123

11.48.1.5 GetNextFeature . . . 123
11.48.1.6 GetSpatialFilter . . . 124
11.48.1.7 GetSpatialRef . . . 124
11.48.1.8 ResetReading . . . 124
11.48.1.9 SetNextByIndex . . . 124
11.48.1.10 TestCapability . . . 125
11.49 OGRGeometry Class Reference . . . 126
11.49.1 Detailed Description . . . 128
11.49.2 Member Function Documentation . . . 129
11.49.2.1 assignSpatialReference . . . 129
11.49.2.2 Boundary . . . 129
11.49.2.3 Buffer . . . 129
11.49.2.4 Centroid . . . 130
11.49.2.5 clone . . . 130
11.49.2.6 closeRings . . . 130
11.49.2.7 Contains . . . 131
11.49.2.8 ConvexHull . . . 131
11.49.2.9 Crosses . . . 131
11.49.2.10 Difference . . . 131
11.49.2.11 Disjoint . . . 132
11.49.2.12 Distance . . . 132
11.49.2.13 dumpReadable . . . 132
11.49.2.14 empty . . . 133
11.49.2.15 Equals . . . 133
11.49.2.16 exportToGML . . . 133
11.49.2.17 exportToJson . . . 134
11.49.2.18 exportToKML . . . 134
11.49.2.19 exportToWkb . . . 134
11.49.2.20 exportToWkt . . . 135
11.49.2.21 flattenTo2D . . . 135
11.49.2.22 getBoundary . . . 135
11.49.2.23 getCoordinateDimension . . . 135
11.49.2.24 getDimension . . . 136
11.49.2.25 getEnvelope . . . 136
11.49.2.26 getEnvelope . . . 136
11.49.2.27 getGeometryName . . . 137
11.49.2.28 getGeometryType . . . 137
11.49.2.29 getSpatialReference . . . 137
11.49.2.30 importFromWkb . . . 138
11.49.2.31 importFromWkt . . . 138

11.49.2.32Intersection . . . 139
11.49.2.33Intersects . . . 139
11.49.2.34IsEmpty . . . 139
11.49.2.35IsRing . . . 140
11.49.2.36IsSimple . . . 140
11.49.2.37IsValid . . . 140
11.49.2.38Overlaps . . . 140
11.49.2.39Polygonize . . . 141
11.49.2.40segmentize . . . 141
11.49.2.41setCoordinateDimension . . . 141
11.49.2.42Simplify . . . 141
11.49.2.43SimplifyPreserveTopology . . . 142
11.49.2.44swapXY . . . 142
11.49.2.45SymDifference . . . 142
11.49.2.46SymmetricDifference . . . 143
11.49.2.47Touches . . . 143
11.49.2.48transform . . . 143
11.49.2.49transformTo . . . 144
11.49.2.50Union . . . 144
11.49.2.51UnionCascaded . . . 145
11.49.2.52Within . . . 145
11.49.2.53WkbSize . . . 145
11.50OGRGeometryCollection Class Reference . . . 146
11.50.1 Detailed Description . . . 147
11.50.2 Member Function Documentation . . . 147
11.50.2.1 addGeometry . . . 147
11.50.2.2 addGeometryDirectly . . . 148
11.50.2.3 clone . . . 148
11.50.2.4 closeRings . . . 148
11.50.2.5 empty . . . 149
11.50.2.6 Equals . . . 149
11.50.2.7 exportToWkb . . . 149
11.50.2.8 exportToWkt . . . 149
11.50.2.9 flattenTo2D . . . 150
11.50.2.10get_Area . . . 150
11.50.2.11get_Length . . . 150
11.50.2.12getDimension . . . 151
11.50.2.13getEnvelope . . . 151
11.50.2.14getEnvelope . . . 151
11.50.2.15getGeometryName . . . 151

11.50.2.16 getGeometryRef . . . 152
11.50.2.17 getGeometryType . . . 152
11.50.2.18 getNumGeometries . . . 152
11.50.2.19 importFromWkb . . . 153
11.50.2.20 importFromWkt . . . 153
11.50.2.21 IsEmpty . . . 153
11.50.2.22 removeGeometry . . . 154
11.50.2.23 segmentize . . . 154
11.50.2.24 setCoordinateDimension . . . 154
11.50.2.25 swapXY . . . 155
11.50.2.26 transform . . . 155
11.50.2.27 WkbSize . . . 155
11.51 OGRGeometryFactory Class Reference . . . 156
11.51.1 Detailed Description . . . 156
11.51.2 Member Function Documentation . . . 157
11.51.2.1 approximateArcAngles . . . 157
11.51.2.2 createFromFgf . . . 157
11.51.2.3 createFromGML . . . 158
11.51.2.4 createFromWkb . . . 158
11.51.2.5 createFromWkt . . . 159
11.51.2.6 createGeometry . . . 159
11.51.2.7 destroyGeometry . . . 160
11.51.2.8 forceToMultiLineString . . . 160
11.51.2.9 forceToMultiPoint . . . 160
11.51.2.10 forceToMultiPolygon . . . 161
11.51.2.11 forceToPolygon . . . 161
11.51.2.12 haveGEOS . . . 161
11.51.2.13 organizePolygons . . . 162
11.52 OGRLayer Class Reference . . . 162
11.52.1 Detailed Description . . . 164
11.52.2 Member Function Documentation . . . 164
11.52.2.1 AlterFieldDefn . . . 164
11.52.2.2 CreateFeature . . . 165
11.52.2.3 CreateField . . . 165
11.52.2.4 DeleteFeature . . . 166
11.52.2.5 DeleteField . . . 166
11.52.2.6 Dereference . . . 166
11.52.2.7 GetExtent . . . 167
11.52.2.8 GetFeature . . . 167
11.52.2.9 GetFeatureCount . . . 168

11.52.2.10GetFIDColumn . . . 168
11.52.2.11GetGeometryColumn . . . 168
11.52.2.12GetGeomType . . . 168
11.52.2.13GetInfo . . . 169
11.52.2.14GetLayerDefn . . . 169
11.52.2.15GetName . . . 169
11.52.2.16GetNextFeature . . . 170
11.52.2.17GetRefCount . . . 170
11.52.2.18GetSpatialFilter . . . 170
11.52.2.19GetSpatialRef . . . 171
11.52.2.20GetStyleTable . . . 171
11.52.2.21Reference . . . 171
11.52.2.22ReorderField . . . 171
11.52.2.23ReorderFields . . . 172
11.52.2.24ResetReading . . . 172
11.52.2.25SetAttributeFilter . . . 173
11.52.2.26SetFeature . . . 173
11.52.2.27SetIgnoredFields . . . 173
11.52.2.28SetNextByIndex . . . 174
11.52.2.29SetSpatialFilter . . . 174
11.52.2.30SetSpatialFilterRect . . . 175
11.52.2.31SetStyleTable . . . 175
11.52.2.32SetStyleTableDirectly . . . 175
11.52.2.33SyncToDisk . . . 176
11.52.2.34TestCapability . . . 176
11.53OGRLayerAttrIndex Class Reference . . . 177
11.54OGRLinearRing Class Reference . . . 178
11.54.1 Detailed Description . . . 178
11.54.2 Member Function Documentation . . . 179
11.54.2.1 clone . . . 179
11.54.2.2 closeRings . . . 179
11.54.2.3 exportToWkb . . . 179
11.54.2.4 get_Area . . . 179
11.54.2.5 getGeometryName . . . 180
11.54.2.6 importFromWkb . . . 180
11.54.2.7 isClockwise . . . 180
11.54.2.8 WkbSize . . . 180
11.55OGRLineString Class Reference . . . 181
11.55.1 Detailed Description . . . 183
11.55.2 Member Function Documentation . . . 183

11.55.2.1 addPoint . . . 183
11.55.2.2 addPoint . . . 183
11.55.2.3 addSubLineString . . . 183
11.55.2.4 clone . . . 184
11.55.2.5 empty . . . 184
11.55.2.6 EndPoint . . . 184
11.55.2.7 Equals . . . 184
11.55.2.8 exportToWkb . . . 185
11.55.2.9 exportToWkt . . . 185
11.55.2.10 flattenTo2D . . . 185
11.55.2.11 get_Length . . . 186
11.55.2.12 getDimension . . . 186
11.55.2.13 getEnvelope . . . 186
11.55.2.14 getEnvelope . . . 186
11.55.2.15 getGeometryName . . . 187
11.55.2.16 getGeometryType . . . 187
11.55.2.17 getNumPoints . . . 187
11.55.2.18 getPoint . . . 188
11.55.2.19 getPoints . . . 188
11.55.2.20 getPoints . . . 188
11.55.2.21 getX . . . 189
11.55.2.22 getY . . . 189
11.55.2.23 getZ . . . 189
11.55.2.24 importFromWkb . . . 189
11.55.2.25 importFromWkt . . . 190
11.55.2.26 IsEmpty . . . 190
11.55.2.27 segmentize . . . 191
11.55.2.28 setCoordinateDimension . . . 191
11.55.2.29 setNumPoints . . . 191
11.55.2.30 setPoint . . . 191
11.55.2.31 setPoint . . . 192
11.55.2.32 setPoints . . . 192
11.55.2.33 setPoints . . . 192
11.55.2.34 StartPoint . . . 193
11.55.2.35 swapXY . . . 193
11.55.2.36 transform . . . 193
11.55.2.37 Value . . . 193
11.55.2.38 WkbSize . . . 194
11.56 OGRMIAttrIndex Class Reference . . . 194
11.57 OGRMILayerAttrIndex Class Reference . . . 194

11.58 OGRMultiLineString Class Reference . . . 195
11.58.1 Detailed Description . . . 195
11.58.2 Member Function Documentation . . . 195
11.58.2.1 addGeometryDirectly . . . 195
11.58.2.2 clone . . . 196
11.58.2.3 exportToWkt . . . 196
11.58.2.4 getGeometryName . . . 197
11.58.2.5 getGeometryType . . . 197
11.58.2.6 importFromWkt . . . 197
11.59 OGRMultiPoint Class Reference . . . 198
11.59.1 Detailed Description . . . 198
11.59.2 Member Function Documentation . . . 198
11.59.2.1 addGeometryDirectly . . . 198
11.59.2.2 clone . . . 199
11.59.2.3 exportToWkt . . . 199
11.59.2.4 getGeometryName . . . 199
11.59.2.5 getGeometryType . . . 200
11.59.2.6 importFromWkt . . . 200
11.60 OGRMultiPolygon Class Reference . . . 200
11.60.1 Detailed Description . . . 201
11.60.2 Member Function Documentation . . . 201
11.60.2.1 addGeometryDirectly . . . 201
11.60.2.2 clone . . . 201
11.60.2.3 exportToWkt . . . 202
11.60.2.4 get_Area . . . 202
11.60.2.5 getGeometryName . . . 202
11.60.2.6 getGeometryType . . . 203
11.60.2.7 importFromWkt . . . 203
11.61 OGRPoint Class Reference . . . 203
11.61.1 Detailed Description . . . 205
11.61.2 Member Function Documentation . . . 205
11.61.2.1 clone . . . 205
11.61.2.2 empty . . . 205
11.61.2.3 Equals . . . 205
11.61.2.4 exportToWkb . . . 205
11.61.2.5 exportToWkt . . . 206
11.61.2.6 flattenTo2D . . . 206
11.61.2.7 getDimension . . . 206
11.61.2.8 getEnvelope . . . 206
11.61.2.9 getEnvelope . . . 207

11.61.2.10 getGeometryName . . . 207
11.61.2.11 getGeometryType . . . 207
11.61.2.12 getX . . . 208
11.61.2.13 getY . . . 208
11.61.2.14 getZ . . . 208
11.61.2.15 importFromWkb . . . 208
11.61.2.16 importFromWkt . . . 209
11.61.2.17 IsEmpty . . . 209
11.61.2.18 setCoordinateDimension . . . 209
11.61.2.19 setX . . . 210
11.61.2.20 setY . . . 210
11.61.2.21 setZ . . . 210
11.61.2.22 swapXY . . . 210
11.61.2.23 transform . . . 210
11.61.2.24 WkbSize . . . 211
11.62 OGRPolygon Class Reference . . . 211
11.62.1 Detailed Description . . . 213
11.62.2 Member Function Documentation . . . 213
11.62.2.1 addRing . . . 213
11.62.2.2 addRingDirectly . . . 213
11.62.2.3 clone . . . 213
11.62.2.4 closeRings . . . 214
11.62.2.5 empty . . . 214
11.62.2.6 Equals . . . 214
11.62.2.7 exportToWkb . . . 214
11.62.2.8 exportToWkt . . . 215
11.62.2.9 flattenTo2D . . . 215
11.62.2.10 get_Area . . . 215
11.62.2.11 getDimension . . . 215
11.62.2.12 getEnvelope . . . 216
11.62.2.13 getEnvelope . . . 216
11.62.2.14 getExteriorRing . . . 216
11.62.2.15 getGeometryName . . . 216
11.62.2.16 getGeometryType . . . 217
11.62.2.17 getInteriorRing . . . 217
11.62.2.18 getNumInteriorRings . . . 217
11.62.2.19 importFromWkb . . . 218
11.62.2.20 importFromWkt . . . 218
11.62.2.21 IsEmpty . . . 218
11.62.2.22 PointOnSurface . . . 219

11.62.2.23 segmentize . . . 219
11.62.2.24 setCoordinateDimension . . . 219
11.62.2.25 swapXY . . . 219
11.62.2.26 transform . . . 220
11.62.2.27 WkbSize . . . 220
11.63 OGRProj4CT Class Reference . . . 220
11.63.1 Member Function Documentation . . . 221
11.63.1.1 GetSourceCS . . . 221
11.63.1.2 GetTargetCS . . . 221
11.63.1.3 Transform . . . 221
11.63.1.4 TransformEx . . . 221
11.64 OGRProj4Datum Struct Reference . . . 222
11.65 OGRProj4PM Struct Reference . . . 222
11.66 OGRRawPoint Class Reference . . . 222
11.66.1 Detailed Description . . . 222
11.67 OGRSFDriver Class Reference . . . 222
11.67.1 Detailed Description . . . 223
11.67.2 Member Function Documentation . . . 223
11.67.2.1 CopyDataSource . . . 223
11.67.2.2 CreateDataSource . . . 223
11.67.2.3 DeleteDataSource . . . 224
11.67.2.4 GetName . . . 224
11.67.2.5 Open . . . 225
11.67.2.6 TestCapability . . . 225
11.68 OGRSFDriverRegistrar Class Reference . . . 226
11.68.1 Detailed Description . . . 226
11.68.2 Member Function Documentation . . . 226
11.68.2.1 AutoLoadDrivers . . . 226
11.68.2.2 DeregisterDriver . . . 227
11.68.2.3 GetDriver . . . 227
11.68.2.4 GetDriverByName . . . 227
11.68.2.5 GetDriverCount . . . 228
11.68.2.6 GetOpenDS . . . 228
11.68.2.7 GetOpenDSCount . . . 228
11.68.2.8 GetRegistrar . . . 228
11.68.2.9 Open . . . 229
11.68.2.10 RegisterDriver . . . 229
11.69 OGRSpatialReference Class Reference . . . 230
11.69.1 Detailed Description . . . 235
11.69.2 Constructor & Destructor Documentation . . . 235

11.69.2.1 OGRSpatialReference . . . 235
11.69.2.2 ∼OGRSpatialReference . . . 236
11.69.3 Member Function Documentation . . . 236
11.69.3.1 AutoIdentifyEPSG . . . 236
11.69.3.2 Clear . . . 236
11.69.3.3 Clone . . . 236
11.69.3.4 CloneGeogCS . . . 237
11.69.3.5 CopyGeogCSFrom . . . 237
11.69.3.6 Dereference . . . 237
11.69.3.7 DestroySpatialReference . . . 237
11.69.3.8 EPSGTreatsAsLatLong . . . 238
11.69.3.9 exportToERM . . . 238
11.69.3.10 exportToMICoordSys . . . 238
11.69.3.11 exportToPanorama . . . 239
11.69.3.12 exportToPCI . . . 239
11.69.3.13 exportToPrettyWkt . . . 240
11.69.3.14 exportToProj4 . . . 240
11.69.3.15 exportToUSGS . . . 240
11.69.3.16 exportToWkt . . . 241
11.69.3.17 exportToXML . . . 241
11.69.3.18 FindProjParm . . . 242
11.69.3.19 Fixup . . . 242
11.69.3.20 FixupOrdering . . . 242
11.69.3.21 GetAngularUnits . . . 243
11.69.3.22 GetAttrNode . . . 243
11.69.3.23 GetAttrValue . . . 243
11.69.3.24 GetAuthorityCode . . . 244
11.69.3.25 GetAuthorityName . . . 244
11.69.3.26 GetAxis . . . 245
11.69.3.27 GetExtension . . . 245
11.69.3.28 GetInvFlattening . . . 245
11.69.3.29 GetLinearUnits . . . 246
11.69.3.30 GetNormProjParm . . . 246
11.69.3.31 GetPrimeMeridian . . . 246
11.69.3.32 GetProjParm . . . 247
11.69.3.33 GetReferenceCount . . . 247
11.69.3.34 GetSemiMajor . . . 247
11.69.3.35 GetSemiMinor . . . 248
11.69.3.36 GetTargetLinearUnits . . . 248
11.69.3.37 GetTOWGS84 . . . 249

11.69.3.38 GetUTMZone . . . 249
11.69.3.39 ImportFromDict . . . 249
11.69.3.40 ImportFromEPSG . . . 250
11.69.3.41 ImportFromEPSGA . . . 250
11.69.3.42 ImportFromERM . . . 251
11.69.3.43 ImportFromESRI . . . 251
11.69.3.44 ImportFromMICoordSys . . . 252
11.69.3.45 ImportFromOzi . . . 252
11.69.3.46 ImportFromPanorama . . . 252
11.69.3.47 ImportFromPCI . . . 254
11.69.3.48 ImportFromProj4 . . . 254
11.69.3.49 ImportFromUrl . . . 255
11.69.3.50 ImportFromURN . . . 256
11.69.3.51 ImportFromUSGS . . . 256
11.69.3.52 ImportFromWkt . . . 260
11.69.3.53 ImportFromWMSAUTO . . . 260
11.69.3.54 ImportFromXML . . . 260
11.69.3.55 IsCompound . . . 261
11.69.3.56 IsGeocentric . . . 261
11.69.3.57 IsGeographic . . . 261
11.69.3.58 IsLocal . . . 261
11.69.3.59 IsProjected . . . 262
11.69.3.60 IsSame . . . 262
11.69.3.61 IsSameGeogCS . . . 262
11.69.3.62 IsSameVertCS . . . 262
11.69.3.63 IsVertical . . . 263
11.69.3.64 MorphFromESRI . . . 263
11.69.3.65 MorphToESRI . . . 264
11.69.3.66 Reference . . . 264
11.69.3.67 Release . . . 265
11.69.3.68 SetACEA . . . 265
11.69.3.69 SetAE . . . 265
11.69.3.70 SetAngularUnits . . . 265
11.69.3.71 SetAuthority . . . 266
11.69.3.72 SetAxes . . . 266
11.69.3.73 SetBonne . . . 266
11.69.3.74 SetCEA . . . 267
11.69.3.75 SetCompoundCS . . . 267
11.69.3.76 SetCS . . . 267
11.69.3.77 SetEC . . . 267

11.69.3.78 SetEckert . . . 267
11.69.3.79 SetEquirectangular . . . 268
11.69.3.80 SetEquirectangular2 . . . 268
11.69.3.81 SetExtension . . . 268
11.69.3.82 SetFromUserInput . . . 268
11.69.3.83 SetGaussSchreiberTMercator . . . 269
11.69.3.84 SetGeocCS . . . 269
11.69.3.85 SetGeogCS . . . 270
11.69.3.86 SetGEOS . . . 270
11.69.3.87 SetGH . . . 271
11.69.3.88 SetGnomonic . . . 271
11.69.3.89 SetGS . . . 271
11.69.3.90 SetHOM . . . 271
11.69.3.91 SetHOM2PNO . . . 271
11.69.3.92 SetIGH . . . 272
11.69.3.93 SetIWMPolyconic . . . 272
11.69.3.94 SetKrovak . . . 272
11.69.3.95 SetLAEA . . . 272
11.69.3.96 SetLCC . . . 272
11.69.3.97 SetLCC1SP . . . 273
11.69.3.98 SetLCCB . . . 273
11.69.3.99 SetLinearUnits . . . 273
11.69.3.100 SetLinearUnitsAndUpdateParameters . . . 273
11.69.3.101 SetLocalCS . . . 274
11.69.3.102 SetMC . . . 274
11.69.3.103 SetMercator . . . 274
11.69.3.104 SetMollweide . . . 274
11.69.3.105 SetNode . . . 275
11.69.3.106 SetNormProjParm . . . 275
11.69.3.107 SetNZMG . . . 275
11.69.3.108 SetOrthographic . . . 276
11.69.3.109 SetOS . . . 276
11.69.3.110 SetPolyconic . . . 276
11.69.3.111 SetProjCS . . . 276
11.69.3.112 SetProjection . . . 276
11.69.3.113 SetProjParm . . . 277
11.69.3.114 SetPS . . . 277
11.69.3.115 SetRobinson . . . 277
11.69.3.116 SetRoot . . . 277
11.69.3.117 SetSinusoidal . . . 278

11.69.3.118 SetSOC . . . 278
11.69.3.119 SetStatePlane . . . 278
11.69.3.120 SetStereographic . . . 279
11.69.3.121 SetTargetLinearUnits . . . 279
11.69.3.122 SetTM . . . 279
11.69.3.123 SetTMG . . . 279
11.69.3.124 SetTMSO . . . 279
11.69.3.125 SetTMVariant . . . 280
11.69.3.126 SetTOWGS84 . . . 280
11.69.3.127 SetTPED . . . 280
11.69.3.128 SetUTM . . . 280
11.69.3.129 SetVDG . . . 281
11.69.3.130 SetVertCS . . . 281
11.69.3.131 SetWagner . . . 281
11.69.3.132 SetWellKnownGeogCS . . . 282
11.69.3.133 StripCTParms . . . 282
11.69.3.134 StripVertical . . . 282
11.69.3.135 Validate . . . 283
11.70 OGRStyleBrush Class Reference . . . 283
11.70.1 Detailed Description . . . 284
11.71 OGRStyleLabel Class Reference . . . 284
11.71.1 Detailed Description . . . 284
11.72 OGRStyleMgr Class Reference . . . 284
11.72.1 Detailed Description . . . 285
11.72.2 Constructor & Destructor Documentation . . . 285
11.72.2.1 OGRStyleMgr . . . 285
11.72.2.2 ~OGRStyleMgr . . . 285
11.72.3 Member Function Documentation . . . 285
11.72.3.1 AddPart . . . 285
11.72.3.2 AddPart . . . 286
11.72.3.3 AddStyle . . . 286
11.72.3.4 GetPart . . . 286
11.72.3.5 GetPartCount . . . 286
11.72.3.6 GetStyleByName . . . 287
11.72.3.7 GetStyleName . . . 287
11.72.3.8 GetStyleString . . . 287
11.72.3.9 InitFromFeature . . . 287
11.72.3.10 InitStyleString . . . 288
11.72.3.11 SetFeatureStyleString . . . 288
11.73 OGRStylePen Class Reference . . . 288

11.73.1 Detailed Description . . . 289
11.74 OGRStyleSymbol Class Reference . . . 289
11.74.1 Detailed Description . . . 289
11.75 OGRStyleTable Class Reference . . . 289
11.75.1 Detailed Description . . . 290
11.75.2 Member Function Documentation . . . 290
11.75.2.1 AddStyle . . . 290
11.75.2.2 Clone . . . 290
11.75.2.3 Find . . . 290
11.75.2.4 GetStyleName . . . 291
11.75.2.5 IsExist . . . 291
11.75.2.6 LoadStyleTable . . . 291
11.75.2.7 ModifyStyle . . . 291
11.75.2.8 Print . . . 292
11.75.2.9 RemoveStyle . . . 292
11.75.2.10 SaveStyleTable . . . 292
11.76 OGRStyleTool Class Reference . . . 292
11.76.1 Detailed Description . . . 293
11.77 OGRSurface Class Reference . . . 293
11.77.1 Detailed Description . . . 293
11.77.2 Member Function Documentation . . . 293
11.77.2.1 get_Area . . . 293
11.77.2.2 PointOnSurface . . . 294
11.78 OZIDatums Struct Reference . . . 294
11.79 ParseContext Struct Reference . . . 294
11.80 PCIDatums Struct Reference . . . 294
11.81 projUV Struct Reference . . . 294
11.82 SFRegion Class Reference . . . 295
11.83 StackContext Struct Reference . . . 295
11.84 swq_col_def Struct Reference . . . 295
11.85 swq_expr_node Class Reference . . . 295
11.86 swq_field_list Class Reference . . . 295
11.87 swq_join_def Struct Reference . . . 295
11.88 swq_op_registrar Class Reference . . . 295
11.89 swq_operation Class Reference . . . 296
11.90 swq_order_def Struct Reference . . . 296
11.91 swq_parse_context Class Reference . . . 296
11.92 swq_select Class Reference . . . 296
11.93 swq_summary Struct Reference . . . 296
11.94 swq_table_def Struct Reference . . . 296

11.95tm_unz_s Struct Reference . . . 296
11.96tm_zip_s Struct Reference . . . 296
11.97unz_file_info_internal_s Struct Reference . . . 297
11.98unz_file_info_s Struct Reference . . . 297
11.99unz_file_pos_s Struct Reference . . . 297
11.100unz_global_info_s Struct Reference . . . 297
11.101unz_s Struct Reference . . . 297
11.102VSIArchiveContent Struct Reference . . . 297
11.103VSIArchiveEntry Struct Reference . . . 297
11.104VSIArchiveEntryFileOffset Class Reference . . . 298
11.105VSIArchiveFilesystemHandler Class Reference . . . 298
11.106VSIArchiveReader Class Reference . . . 298
11.107VSIBufferedReaderHandle Class Reference . . . 299
11.108VSICacheChunk Class Reference . . . 299
11.109VSICachedFile Class Reference . . . 299
11.110VSICurlFilesystemHandler Class Reference . . . 299
11.111VSICurlHandle Class Reference . . . 300
11.112VSIFileManager Class Reference . . . 300
11.113VSIFilesystemHandler Class Reference . . . 300
11.114VSIGZipFilesystemHandler Class Reference . . . 301
11.115VSIGZipHandle Class Reference . . . 301
11.116VSIGZipWriteHandle Class Reference . . . 301
11.117VSIMemFile Class Reference . . . 302
11.118VSIMemFilesystemHandler Class Reference . . . 302
11.119VSIMemHandle Class Reference . . . 302
11.120VSISparseFileFilesystemHandler Class Reference . . . 302
11.121VSISparseFileHandle Class Reference . . . 303
11.122VSIStdinFilesystemHandler Class Reference . . . 303
11.123VSIStdinHandle Class Reference . . . 303
11.124VSIStdoutFilesystemHandler Class Reference . . . 303
11.125VSIStdoutHandle Class Reference . . . 304
11.126VSIStdoutRedirectFilesystemHandler Class Reference . . . 304
11.127VSIStdoutRedirectHandle Class Reference . . . 304
11.128VSISubFileFilesystemHandler Class Reference . . . 305
11.129VSISubFileHandle Class Reference . . . 305
11.130VSITarEntryFileOffset Class Reference . . . 305
11.131VSITarFilesystemHandler Class Reference . . . 305
11.132VSITarReader Class Reference . . . 306
11.133VSIUnixStdioFilesystemHandler Class Reference . . . 306
11.134VSIUnixStdioHandle Class Reference . . . 306

11.135VSIVirtualHandle Class Reference . . . 307
11.136VSIZipEntryFileOffset Class Reference . . . 307
11.137VSIZipFilesystemHandler Class Reference . . . 308
11.138VSIZipReader Class Reference . . . 308
11.139VSIZipWriteHandle Class Reference . . . 308
11.140WriteFuncStruct Struct Reference . . . 309
11.141yyalloc Union Reference . . . 309
11.142zip_fileinfo Struct Reference . . . 309
11.143zip_internal Struct Reference . . . 309
11.144zlib_filefunc_def_s Struct Reference . . . 309

12 File Documentation **311**

12.1 cpl_conv.h File Reference . . . 311
12.1.1 Detailed Description . . . 312
12.1.2 Function Documentation . . . 312
12.1.2.1 CPLAtof . . . 312
12.1.2.2 CPLAtofDelim . . . 313
12.1.2.3 CPLAtofM . . . 313
12.1.2.4 CPLCalloc . . . 314
12.1.2.5 CPLCheckForFile . . . 314
12.1.2.6 CPLCleanTrailingSlash . . . 315
12.1.2.7 CPLCloseShared . . . 315
12.1.2.8 CPLCorrespondingPaths . . . 315
12.1.2.9 CPLDecToPackedDMS . . . 316
12.1.2.10 CPLDumpSharedList . . . 316
12.1.2.11 CPLExtractRelativePath . . . 316
12.1.2.12 CPLFGets . . . 317
12.1.2.13 CPLFormCIFilename . . . 317
12.1.2.14 CPLFormFilename . . . 317
12.1.2.15 CPLGenerateTempFilename . . . 318
12.1.2.16 CPLGetBasename . . . 318
12.1.2.17 CPLGetConfigOption . . . 319
12.1.2.18 CPLGetCurrentDir . . . 319
12.1.2.19 CPLGetDirname . . . 319
12.1.2.20 CPLGetExecPath . . . 320
12.1.2.21 CPLGetExtension . . . 320
12.1.2.22 CPLGetFilename . . . 320
12.1.2.23 CPLGetPath . . . 321
12.1.2.24 CPLGetSharedList . . . 321
12.1.2.25 CPLGetSymbol . . . 321

12.1.2.26 CPLIsFilenameRelative . . . 322
12.1.2.27 CPLMalloc . . . 322
12.1.2.28 CPLOpenShared . . . 323
12.1.2.29 CPLPackedDMSToDec . . . 323
12.1.2.30 CPLPrintDouble . . . 324
12.1.2.31 CPLPrintInt32 . . . 324
12.1.2.32 CPLPrintPointer . . . 324
12.1.2.33 CPLPrintString . . . 325
12.1.2.34 CPLPrintStringFill . . . 325
12.1.2.35 CPLPrintTime . . . 325
12.1.2.36 CPLPrintUIntBig . . . 326
12.1.2.37 CPLProjectRelativeFilename . . . 326
12.1.2.38 CPLReadLine . . . 327
12.1.2.39 CPLReadLine2L . . . 327
12.1.2.40 CPLReadLineL . . . 327
12.1.2.41 CPLRealloc . . . 328
12.1.2.42 CPLResetExtension . . . 328
12.1.2.43 CPLScanDouble . . . 328
12.1.2.44 CPLScanLong . . . 329
12.1.2.45 CPLScanPointer . . . 329
12.1.2.46 CPLScanString . . . 329
12.1.2.47 CPLScanUIntBig . . . 330
12.1.2.48 CPLScanULong . . . 330
12.1.2.49 CPLSetConfigOption . . . 330
12.1.2.50 CPLSetThreadLocalConfigOption . . . 330
12.1.2.51 CPLStrdup . . . 331
12.1.2.52 CPLStrlwr . . . 331
12.1.2.53 CPLStrtod . . . 331
12.1.2.54 CPLStrtodDelim . . . 332
12.1.2.55 CPLStrtof . . . 332
12.1.2.56 CPLStrtofDelim . . . 332
12.1.2.57 CPLUnlinkTree . . . 333
12.2 cpl_error.h File Reference . . . 333
12.2.1 Detailed Description . . . 333
12.2.2 Function Documentation . . . 334
12.2.2.1 _CPLAssert . . . 334
12.2.2.2 CPLDebug . . . 334
12.2.2.3 CPLEmergencyError . . . 334
12.2.2.4 CPLError . . . 334
12.2.2.5 CPLErrorReset . . . 335

12.2.2.6 CPLGetErrorHandlerUserData . . . 335
12.2.2.7 CPLGetLastErrorMsg . . . 335
12.2.2.8 CPLGetLastErrorNo . . . 335
12.2.2.9 CPLGetLastErrorType . . . 336
12.2.2.10 CPLPopErrorHandler . . . 336
12.2.2.11 CPLPushErrorHandler . . . 336
12.2.2.12 CPLPushErrorHandlerEx . . . 336
12.2.2.13 CPLSetErrorHandler . . . 336
12.2.2.14 CPLSetErrorHandlerEx . . . 337
12.3 cpl_hash_set.h File Reference . . . 337
12.3.1 Detailed Description . . . 338
12.3.2 Function Documentation . . . 338
12.3.2.1 CPLHashSetDestroy . . . 338
12.3.2.2 CPLHashSetEqualPointer . . . 338
12.3.2.3 CPLHashSetEqualStr . . . 338
12.3.2.4 CPLHashSetForeach . . . 339
12.3.2.5 CPLHashSetHashPointer . . . 339
12.3.2.6 CPLHashSetHashStr . . . 339
12.3.2.7 CPLHashSetInsert . . . 339
12.3.2.8 CPLHashSetLookup . . . 340
12.3.2.9 CPLHashSetNew . . . 340
12.3.2.10 CPLHashSetRemove . . . 340
12.3.2.11 CPLHashSetSize . . . 340
12.4 cpl_http.h File Reference . . . 341
12.4.1 Detailed Description . . . 341
12.4.2 Function Documentation . . . 341
12.4.2.1 CPLHTTPDestroyResult . . . 341
12.4.2.2 CPLHTTPEnabled . . . 342
12.4.2.3 CPLHTTPFetch . . . 342
12.4.2.4 CPLHTTPParseMultipartMime . . . 342
12.5 cpl_list.h File Reference . . . 343
12.5.1 Detailed Description . . . 343
12.5.2 Typedef Documentation . . . 343
12.5.2.1 CPLList . . . 343
12.5.3 Function Documentation . . . 344
12.5.3.1 CPLListAppend . . . 344
12.5.3.2 CPLListCount . . . 344
12.5.3.3 CPLListDestroy . . . 344
12.5.3.4 CPLListGet . . . 344
12.5.3.5 CPLListGetData . . . 345

12.5.3.6 CPLListGetLast . . . 345
12.5.3.7 CPLListGetNext . . . 345
12.5.3.8 CPLListInsert . . . 345
12.5.3.9 CPLListRemove . . . 346
12.6 cpl_minixml.h File Reference . . . 346
12.6.1 Detailed Description . . . 347
12.6.2 Typedef Documentation . . . 347
12.6.2.1 CPLXMLNode . . . 347
12.6.3 Enumeration Type Documentation . . . 347
12.6.3.1 CPLXMLNodeType . . . 347
12.6.4 Function Documentation . . . 348
12.6.4.1 CPLAddXMLChild . . . 348
12.6.4.2 CPLAddXMLSibling . . . 348
12.6.4.3 CPLCleanXMLElementName . . . 348
12.6.4.4 CPLCloneXMLTree . . . 348
12.6.4.5 CPLCreateXMLElementAndValue . . . 349
12.6.4.6 CPLCreateXMLNode . . . 349
12.6.4.7 CPLDestroyXMLNode . . . 350
12.6.4.8 CPLGetXMLNode . . . 350
12.6.4.9 CPLGetXMLValue . . . 350
12.6.4.10 CPLParseXMLFile . . . 351
12.6.4.11 CPLParseXMLString . . . 351
12.6.4.12 CPLRemoveXMLChild . . . 351
12.6.4.13 CPLSearchXMLNode . . . 352
12.6.4.14 CPLSerializeXMLTree . . . 352
12.6.4.15 CPLSerializeXMLTreeToFile . . . 352
12.6.4.16 CPLSetXMLValue . . . 353
12.6.4.17 CPLStripXMLNamespace . . . 353
12.7 cpl_odbc.h File Reference . . . 354
12.7.1 Detailed Description . . . 354
12.8 cpl_port.h File Reference . . . 354
12.8.1 Detailed Description . . . 354
12.8.2 Macro Definition Documentation . . . 354
12.8.2.1 CPL_LSBINT16PTR . . . 354
12.8.2.2 CPL_LSBINT32PTR . . . 355
12.9 cpl_quad_tree.h File Reference . . . 355
12.9.1 Detailed Description . . . 355
12.9.2 Function Documentation . . . 355
12.9.2.1 CPLQuadTreeCreate . . . 355
12.9.2.2 CPLQuadTreeDestroy . . . 356

12.9.2.3 CPLQuadTreeForeach . . . 356
12.9.2.4 CPLQuadTreeGetAdvisedMaxDepth . . . 356
12.9.2.5 CPLQuadTreeInsert . . . 356
12.9.2.6 CPLQuadTreeSearch . . . 356
12.9.2.7 CPLQuadTreeSetBucketCapacity . . . 357
12.9.2.8 CPLQuadTreeSetMaxDepth . . . 357
12.10 cpl_string.h File Reference . . . 357
12.10.1 Detailed Description . . . 358
12.10.2 Function Documentation . . . 358
12.10.2.1 CPLBinaryToHex . . . 358
12.10.2.2 CPLEncodingCharSize . . . 359
12.10.2.3 CPLEscapeString . . . 359
12.10.2.4 CPLForceToASCII . . . 360
12.10.2.5 CPLGetValueType . . . 360
12.10.2.6 CPLHexToBinary . . . 360
12.10.2.7 CPLIsUTF8 . . . 361
12.10.2.8 CPLParseNameValue . . . 361
12.10.2.9 CPLRecode . . . 361
12.10.2.10 CPLRecodeFromWChar . . . 362
12.10.2.11 CPLRecodeToWChar . . . 362
12.10.2.12 CPLStrlcat . . . 363
12.10.2.13 CPLStrlcpy . . . 363
12.10.2.14 CPLStrnlen . . . 364
12.10.2.15 CPLUnescapeString . . . 364
12.10.2.16 CPLURLAddKVP . . . 364
12.10.2.17 CPLURLGetValue . . . 365
12.10.2.18 CSLCount . . . 365
12.10.2.19 CSLDestroy . . . 365
12.10.2.20 CSLDuplicate . . . 366
12.10.2.21 CSLFindName . . . 366
12.10.2.22 CSLFindString . . . 366
12.10.2.23 CSLLoad . . . 366
12.10.2.24 CSLLoad2 . . . 367
12.10.2.25 CSLMerge . . . 367
12.10.2.26 CSLPartialFindString . . . 367
12.10.2.27 CSLSetNameValue . . . 368
12.10.2.28 CSLSetNameValueSeparator . . . 368
12.10.2.29 CSLTestBoolean . . . 368
12.10.2.30 CSLTokenizeString2 . . . 369
12.11 cpl_vsi.h File Reference . . . 370

12.11.1 Detailed Description . . . 371
12.11.2 Function Documentation . . . 371
12.11.2.1 VSIFCloseL . . . 371
12.11.2.2 VSIFEofL . . . 372
12.11.2.3 VSIFFlushL . . . 372
12.11.2.4 VSIFileFromMemBuffer . . . 372
12.11.2.5 VSIFOpenL . . . 373
12.11.2.6 VSIFPrintfL . . . 373
12.11.2.7 VSIFReadL . . . 374
12.11.2.8 VSIFReadMultiRangeL . . . 374
12.11.2.9 VSIFSeekL . . . 375
12.11.2.10 VSIFTellL . . . 375
12.11.2.11 VSIFTruncateL . . . 375
12.11.2.12 VSIFWriteL . . . 376
12.11.2.13 VSIGetMemFileBuffer . . . 376
12.11.2.14 VSIInstallCurlFileHandler . . . 377
12.11.2.15 VSIInstallGZipFileHandler . . . 377
12.11.2.16 VSIInstallMemFileHandler . . . 377
12.11.2.17 VSIInstallSparseFileHandler . . . 378
12.11.2.18 VSIInstallStdinHandler . . . 379
12.11.2.19 VSIInstallStdoutHandler . . . 379
12.11.2.20 VSIInstallSubFileHandler . . . 379
12.11.2.21 VSIInstallTarFileHandler . . . 380
12.11.2.22 VSIInstallZipFileHandler . . . 380
12.11.2.23 VSIIsCaseSensitiveFS . . . 381
12.11.2.24 VSIMalloc2 . . . 381
12.11.2.25 VSIMalloc3 . . . 381
12.11.2.26 VSIMkdir . . . 382
12.11.2.27 VSIReadDir . . . 382
12.11.2.28 VSIRename . . . 382
12.11.2.29 VSIRmdir . . . 383
12.11.2.30 VSIStatExL . . . 383
12.11.2.31 VSIStatL . . . 384
12.11.2.32 VSIUnlink . . . 384
12.12 ogr_api.h File Reference . . . 384
12.12.1 Detailed Description . . . 395
12.12.2 Function Documentation . . . 395
12.12.2.1 OGR_Dr_CopyDataSource . . . 395
12.12.2.2 OGR_Dr_CreateDataSource . . . 395
12.12.2.3 OGR_Dr_DeleteDataSource . . . 396

12.12.2.4 OGR_Dr_GetName . . . 396
12.12.2.5 OGR_Dr_Open . . . 396
12.12.2.6 OGR_Dr_TestCapability . . . 397
12.12.2.7 OGR_DS_CopyLayer . . . 397
12.12.2.8 OGR_DS_CreateLayer . . . 398
12.12.2.9 OGR_DS_DeleteLayer . . . 398
12.12.2.10 OGR_DS_Destroy . . . 399
12.12.2.11 OGR_DS_ExecuteSQL . . . 399
12.12.2.12 OGR_DS_GetDriver . . . 400
12.12.2.13 OGR_DS_GetLayer . . . 400
12.12.2.14 OGR_DS_GetLayerByName . . . 400
12.12.2.15 OGR_DS_GetLayerCount . . . 401
12.12.2.16 OGR_DS_GetName . . . 401
12.12.2.17 OGR_DS_ReleaseResultSet . . . 401
12.12.2.18 OGR_DS_SyncToDisk . . . 402
12.12.2.19 OGR_DS_TestCapability . . . 402
12.12.2.20 OGR_F_Clone . . . 403
12.12.2.21 OGR_F_Create . . . 403
12.12.2.22 OGR_F_Destroy . . . 403
12.12.2.23 OGR_F_DumpReadable . . . 404
12.12.2.24 OGR_F_Equal . . . 404
12.12.2.25 OGR_F_GetDefnRef . . . 404
12.12.2.26 OGR_F_GetFID . . . 404
12.12.2.27 OGR_F_GetFieldAsBinary . . . 405
12.12.2.28 OGR_F_GetFieldAsDateTime . . . 405
12.12.2.29 OGR_F_GetFieldAsDouble . . . 406
12.12.2.30 OGR_F_GetFieldAsDoubleList . . . 406
12.12.2.31 OGR_F_GetFieldAsInteger . . . 406
12.12.2.32 OGR_F_GetFieldAsIntegerList . . . 407
12.12.2.33 OGR_F_GetFieldAsString . . . 407
12.12.2.34 OGR_F_GetFieldAsStringList . . . 407
12.12.2.35 OGR_F_GetFieldCount . . . 408
12.12.2.36 OGR_F_GetFieldDefnRef . . . 408
12.12.2.37 OGR_F_GetFieldIndex . . . 409
12.12.2.38 OGR_F_GetGeometryRef . . . 409
12.12.2.39 OGR_F_GetRawFieldRef . . . 409
12.12.2.40 OGR_F_GetStyleString . . . 410
12.12.2.41 OGR_F_IsFieldSet . . . 410
12.12.2.42 OGR_F_SetFID . . . 410
12.12.2.43 OGR_F_SetFieldBinary . . . 411

12.12.2.44OGR_F_SetFieldDateTime . . . 411
12.12.2.45OGR_F_SetFieldDouble . . . 411
12.12.2.46OGR_F_SetFieldDoubleList . . . 412
12.12.2.47OGR_F_SetFieldInteger . . . 412
12.12.2.48OGR_F_SetFieldIntegerList . . . 412
12.12.2.49OGR_F_SetFieldRaw . . . 413
12.12.2.50OGR_F_SetFieldString . . . 413
12.12.2.51OGR_F_SetFieldStringList . . . 413
12.12.2.52OGR_F_SetFrom . . . 413
12.12.2.53OGR_F_SetFromWithMap . . . 414
12.12.2.54OGR_F_SetGeometry . . . 414
12.12.2.55OGR_F_SetGeometryDirectly . . . 415
12.12.2.56OGR_F_SetStyleString . . . 415
12.12.2.57OGR_F_SetStyleStringDirectly . . . 415
12.12.2.58OGR_F_StealGeometry . . . 416
12.12.2.59OGR_F_UnsetField . . . 416
12.12.2.60OGR_FD_AddFieldDefn . . . 416
12.12.2.61OGR_FD_Create . . . 417
12.12.2.62OGR_FD_DeleteFieldDefn . . . 417
12.12.2.63OGR_FD_Dereference . . . 417
12.12.2.64OGR_FD_Destroy . . . 418
12.12.2.65OGR_FD_GetFieldCount . . . 418
12.12.2.66OGR_FD_GetFieldDefn . . . 418
12.12.2.67OGR_FD_GetFieldIndex . . . 419
12.12.2.68OGR_FD_GetGeomType . . . 419
12.12.2.69OGR_FD_GetName . . . 419
12.12.2.70OGR_FD_GetReferenceCount . . . 419
12.12.2.71OGR_FD_IsGeometryIgnored . . . 420
12.12.2.72OGR_FD_IsStyleIgnored . . . 420
12.12.2.73OGR_FD_Reference . . . 420
12.12.2.74OGR_FD_Release . . . 421
12.12.2.75OGR_FD_SetGeometryIgnored . . . 421
12.12.2.76OGR_FD_SetGeomType . . . 421
12.12.2.77OGR_FD_SetStyleIgnored . . . 422
12.12.2.78OGR_Fld_Create . . . 422
12.12.2.79OGR_Fld_Destroy . . . 422
12.12.2.80OGR_Fld_GetJustify . . . 422
12.12.2.81OGR_Fld_GetNameRef . . . 423
12.12.2.82OGR_Fld_GetPrecision . . . 423
12.12.2.83OGR_Fld_GetType . . . 423

12.12.2.84 OGR_Fld_GetWidth . . . 424
12.12.2.85 OGR_Fld_IsIgnored . . . 424
12.12.2.86 OGR_Fld_Set . . . 424
12.12.2.87 OGR_Fld_SetIgnored . . . 424
12.12.2.88 OGR_Fld_SetJustify . . . 425
12.12.2.89 OGR_Fld_SetName . . . 425
12.12.2.90 OGR_Fld_SetPrecision . . . 425
12.12.2.91 OGR_Fld_SetType . . . 426
12.12.2.92 OGR_Fld_SetWidth . . . 426
12.12.2.93 OGR_G_AddGeometry . . . 426
12.12.2.94 OGR_G_AddGeometryDirectly . . . 426
12.12.2.95 OGR_G_AddPoint . . . 427
12.12.2.96 OGR_G_AddPoint_2D . . . 427
12.12.2.97 OGR_G_ApproximateArcAngles . . . 427
12.12.2.98 OGR_G_Area . . . 428
12.12.2.99 OGR_G_AssignSpatialReference . . . 429
12.12.2.100 OGR_G_Boundary . . . 429
12.12.2.101 OGR_G_Buffer . . . 429
12.12.2.102 OGR_G_Centroid . . . 430
12.12.2.103 OGR_G_Clone . . . 430
12.12.2.104 OGR_G_CloseRings . . . 431
12.12.2.105 OGR_G_Contains . . . 431
12.12.2.106 OGR_G_ConvexHull . . . 431
12.12.2.107 OGR_G_CreateFromGML . . . 432
12.12.2.108 OGR_G_CreateFromWkb . . . 432
12.12.2.109 OGR_G_CreateFromWkt . . . 433
12.12.2.110 OGR_G_CreateGeometry . . . 433
12.12.2.111 OGR_G_Crosses . . . 433
12.12.2.112 OGR_G_DestroyGeometry . . . 434
12.12.2.113 OGR_G_Difference . . . 434
12.12.2.114 OGR_G_Disjoint . . . 434
12.12.2.115 OGR_G_Distance . . . 435
12.12.2.116 OGR_G_DumpReadable . . . 435
12.12.2.117 OGR_G_Empty . . . 435
12.12.2.118 OGR_G_Equals . . . 436
12.12.2.119 OGR_G_ExportToGML . . . 436
12.12.2.120 OGR_G_ExportToGMLEx . . . 436
12.12.2.121 OGR_G_ExportToJson . . . 437
12.12.2.122 OGR_G_ExportToJsonEx . . . 437
12.12.2.123 OGR_G_ExportToKML . . . 438

12.12.2.124 OGR_G_ExportToWkb . . . 438
12.12.2.125 OGR_G_ExportToWkt . . . 438
12.12.2.126 OGR_G_FlattenTo2D . . . 439
12.12.2.127 OGR_G_ForceToMultiLineString . . . 439
12.12.2.128 OGR_G_ForceToMultiPoint . . . 439
12.12.2.129 OGR_G_ForceToMultiPolygon . . . 440
12.12.2.130 OGR_G_ForceToPolygon . . . 440
12.12.2.131 OGR_G_GetArea . . . 440
12.12.2.132 OGR_G_GetBoundary . . . 441
12.12.2.133 OGR_G_GetCoordinateDimension . . . 441
12.12.2.134 OGR_G_GetDimension . . . 441
12.12.2.135 OGR_G_GetEnvelope . . . 442
12.12.2.136 OGR_G_GetEnvelope3D . . . 442
12.12.2.137 OGR_G_GetGeometryCount . . . 442
12.12.2.138 OGR_G_GetGeometryName . . . 443
12.12.2.139 OGR_G_GetGeometryRef . . . 443
12.12.2.140 OGR_G_GetGeometryType . . . 443
12.12.2.141 OGR_G_GetPoint . . . 444
12.12.2.142 OGR_G_GetPointCount . . . 444
12.12.2.143 OGR_G_GetPoints . . . 444
12.12.2.144 OGR_G_GetSpatialReference . . . 445
12.12.2.145 OGR_G_GetX . . . 445
12.12.2.146 OGR_G_GetY . . . 445
12.12.2.147 OGR_G_GetZ . . . 446
12.12.2.148 OGR_G_ImportFromWkb . . . 446
12.12.2.149 OGR_G_ImportFromWkt . . . 446
12.12.2.150 OGR_G_Intersection . . . 447
12.12.2.151 OGR_G_Intersects . . . 447
12.12.2.152 OGR_G_IsEmpty . . . 448
12.12.2.153 OGR_G_IsRing . . . 448
12.12.2.154 OGR_G_IsSimple . . . 448
12.12.2.155 OGR_G_IsValid . . . 449
12.12.2.156 OGR_G_Length . . . 449
12.12.2.157 OGR_G_Overlaps . . . 449
12.12.2.158 OGR_G_Polygonize . . . 450
12.12.2.159 OGR_G_RemoveGeometry . . . 450
12.12.2.160 OGR_G_Segmentize . . . 451
12.12.2.161 OGR_G_SetCoordinateDimension . . . 451
12.12.2.162 OGR_G_SetPoint . . . 451
12.12.2.163 OGR_G_SetPoint_2D . . . 451

12.12.2.164 OGR_G_Simplify . . . 452
12.12.2.165 OGR_G_SimplifyPreserveTopology . . . 452
12.12.2.166 OGR_G_SymDifference . . . 453
12.12.2.167 OGR_G_SymmetricDifference . . . 453
12.12.2.168 OGR_G_Touches . . . 453
12.12.2.169 OGR_G_Transform . . . 454
12.12.2.170 OGR_G_TransformTo . . . 454
12.12.2.171 OGR_G_Union . . . 455
12.12.2.172 OGR_G_UnionCascaded . . . 455
12.12.2.173 OGR_G_Within . . . 455
12.12.2.174 OGR_G_WkbSize . . . 456
12.12.2.175 OGR_GetFieldTypeName . . . 456
12.12.2.176 OGR_L_AlterFieldDefn . . . 456
12.12.2.177 OGR_L_CommitTransaction . . . 457
12.12.2.178 OGR_L_CreateFeature . . . 457
12.12.2.179 OGR_L_CreateField . . . 458
12.12.2.180 OGR_L_DeleteFeature . . . 458
12.12.2.181 OGR_L_DeleteField . . . 459
12.12.2.182 OGR_L_GetExtent . . . 459
12.12.2.183 OGR_L_GetFeature . . . 460
12.12.2.184 OGR_L_GetFeatureCount . . . 460
12.12.2.185 OGR_L_GetFIDColumn . . . 461
12.12.2.186 OGR_L_GetGeometryColumn . . . 461
12.12.2.187 OGR_L_GetGeomType . . . 461
12.12.2.188 OGR_L_GetLayerDefn . . . 462
12.12.2.189 OGR_L_GetName . . . 462
12.12.2.190 OGR_L_GetNextFeature . . . 462
12.12.2.191 OGR_L_GetSpatialFilter . . . 463
12.12.2.192 OGR_L_GetSpatialRef . . . 463
12.12.2.193 OGR_L_ReorderField . . . 464
12.12.2.194 OGR_L_ReorderFields . . . 464
12.12.2.195 OGR_L_ResetReading . . . 465
12.12.2.196 OGR_L_RollbackTransaction . . . 465
12.12.2.197 OGR_L_SetAttributeFilter . . . 465
12.12.2.198 OGR_L_SetFeature . . . 466
12.12.2.199 OGR_L_SetIgnoredFields . . . 466
12.12.2.200 OGR_L_SetNextByIndex . . . 467
12.12.2.201 OGR_L_SetSpatialFilter . . . 467
12.12.2.202 OGR_L_SetSpatialFilterRect . . . 468
12.12.2.203 OGR_L_StartTransaction . . . 468

12.12.2.204 OGR_L_SyncToDisk . . . 468
12.12.2.205 OGR_L_TestCapability . . . 469
12.12.2.206 OGR_SM_AddPart . . . 470
12.12.2.207 OGR_SM_AddStyle . . . 470
12.12.2.208 OGR_SM_Create . . . 471
12.12.2.209 OGR_SM_Destroy . . . 471
12.12.2.210 OGR_SM_GetPart . . . 471
12.12.2.211 OGR_SM_GetPartCount . . . 472
12.12.2.212 OGR_SM_InitFromFeature . . . 472
12.12.2.213 OGR_SM_InitStyleString . . . 472
12.12.2.214 OGR_ST_Create . . . 473
12.12.2.215 OGR_ST_Destroy . . . 473
12.12.2.216 OGR_ST_GetParamDbl . . . 473
12.12.2.217 OGR_ST_GetParamNum . . . 473
12.12.2.218 OGR_ST_GetParamStr . . . 474
12.12.2.219 OGR_ST_GetRGBFromString . . . 474
12.12.2.220 OGR_ST_GetStyleString . . . 475
12.12.2.221 OGR_ST_GetType . . . 475
12.12.2.222 OGR_ST_GetUnit . . . 475
12.12.2.223 OGR_ST_SetParamDbl . . . 476
12.12.2.224 OGR_ST_SetParamNum . . . 476
12.12.2.225 OGR_ST_SetParamStr . . . 476
12.12.2.226 OGR_ST_SetUnit . . . 476
12.12.2.227 OGR_STBL_Create . . . 477
12.12.2.228 OGR_STBL_Destroy . . . 477
12.12.2.229 OGR_STBL_Find . . . 477
12.12.2.230 OGR_STBL_GetLastStyleName . . . 477
12.12.2.231 OGR_STBL_GetNextStyle . . . 478
12.12.2.232 OGR_STBL_LoadStyleTable . . . 478
12.12.2.233 OGR_STBL_ResetStyleStringReading . . . 478
12.12.2.234 OGR_STBL_SaveStyleTable . . . 479
12.12.2.235 OGRBuildPolygonFromEdges . . . 479
12.12.2.236 OGRCleanupAll . . . 479
12.12.2.237 OGRDeregisterDriver . . . 480
12.12.2.238 OGRGetDriver . . . 480
12.12.2.239 OGRGetDriverByName . . . 480
12.12.2.240 OGRGetDriverCount . . . 481
12.12.2.241 OGRGetOpenDS . . . 481
12.12.2.242 OGRGetOpenDSCount . . . 481
12.12.2.243 OGROpen . . . 481

12.12.2.244 OGRRegisterDriver . . . 482
12.12.2.245 OGRReleaseDataSource . . . 483
12.12.2.246 OGRSetGenerate_DB2_V72_BYTE_ORDER . . . 483
12.13 ogr_core.h File Reference . . . 483
12.13.1 Detailed Description . . . 484
12.13.2 Macro Definition Documentation . . . 484
12.13.2.1 GDAL_CHECK_VERSION . . . 484
12.13.3 Typedef Documentation . . . 485
12.13.3.1 OGRSTBrushParam . . . 485
12.13.3.2 OGRSTClassId . . . 485
12.13.3.3 OGRSTLabelParam . . . 485
12.13.3.4 OGRSTPenParam . . . 485
12.13.3.5 OGRSTSymbolParam . . . 485
12.13.3.6 OGRSTUnitId . . . 485
12.13.4 Enumeration Type Documentation . . . 485
12.13.4.1 ogr_style_tool_class_id . . . 485
12.13.4.2 ogr_style_tool_param_brush_id . . . 485
12.13.4.3 ogr_style_tool_param_label_id . . . 485
12.13.4.4 ogr_style_tool_param_pen_id . . . 485
12.13.4.5 ogr_style_tool_param_symbol_id . . . 485
12.13.4.6 ogr_style_tool_units_id . . . 486
12.13.4.7 OGRFieldType . . . 486
12.13.4.8 OGRJustification . . . 486
12.13.4.9 OGRwkbGeometryType . . . 486
12.13.5 Function Documentation . . . 487
12.13.5.1 GDALCheckVersion . . . 487
12.13.5.2 OGRGeometryTypeToName . . . 487
12.13.5.3 OGRMergeGeometryTypes . . . 487
12.13.5.4 OGRParseDate . . . 488
12.14 ogr_feature.h File Reference . . . 488
12.14.1 Detailed Description . . . 488
12.15 ogr_featurestyle.h File Reference . . . 489
12.15.1 Detailed Description . . . 489
12.16 ogr_geometry.h File Reference . . . 489
12.16.1 Detailed Description . . . 489
12.17 ogr_spatialref.h File Reference . . . 490
12.17.1 Detailed Description . . . 490
12.17.2 Function Documentation . . . 490
12.17.2.1 OGRCreateCoordinateTransformation . . . 490
12.18 ogr_srs_api.h File Reference . . . 490

12.18.1 Detailed Description . . . 496
12.18.2 Function Documentation . . . 496
12.18.2.1 OCTDestroyCoordinateTransformation . . . 496
12.18.2.2 OCTNewCoordinateTransformation . . . 496
12.18.2.3 OPTGetParameterInfo . . . 497
12.18.2.4 OPTGetParameterList . . . 497
12.18.2.5 OPTGetProjectionMethods . . . 497
12.18.2.6 OSRAutoIdentifyEPSG . . . 497
12.18.2.7 OSRAxisEnumToName . . . 498
12.18.2.8 OSRCleanup . . . 498
12.18.2.9 OSRClone . . . 498
12.18.2.10 OSRCloneGeogCS . . . 498
12.18.2.11 OSRCopyGeogCSFrom . . . 498
12.18.2.12 OSRDereference . . . 498
12.18.2.13 OSRDestroySpatialReference . . . 499
12.18.2.14 OSREPSGTreatsAsLatLong . . . 499
12.18.2.15 OSRExportToERM . . . 499
12.18.2.16 OSRExportToMICoordSys . . . 499
12.18.2.17 OSRExportToPCI . . . 499
12.18.2.18 OSRExportToPrettyWkt . . . 499
12.18.2.19 OSRExportToProj4 . . . 500
12.18.2.20 OSRExportToUSGS . . . 500
12.18.2.21 OSRExportToWkt . . . 500
12.18.2.22 OSRExportToXML . . . 500
12.18.2.23 OSRFixup . . . 500
12.18.2.24 OSRFixupOrdering . . . 500
12.18.2.25 OSRGetAngularUnits . . . 501
12.18.2.26 OSRGetAttrValue . . . 501
12.18.2.27 OSRGetAuthorityCode . . . 501
12.18.2.28 OSRGetAuthorityName . . . 501
12.18.2.29 OSRGetAxis . . . 501
12.18.2.30 OSRGetInvFlattening . . . 501
12.18.2.31 OSRGetLinearUnits . . . 502
12.18.2.32 OSRGetNormProjParm . . . 502
12.18.2.33 OSRGetPrimeMeridian . . . 502
12.18.2.34 OSRGetProjParm . . . 502
12.18.2.35 OSRGetSemiMajor . . . 502
12.18.2.36 OSRGetSemiMinor . . . 502
12.18.2.37 OSRGetTargetLinearUnits . . . 503
12.18.2.38 OSRGetTOWGS84 . . . 503

12.18.2.39OSRGetUTMZone . . . 503
12.18.2.40OSRImportFromEPSG . . . 503
12.18.2.41OSRImportFromEPSGA . . . 503
12.18.2.42OSRImportFromERM . . . 503
12.18.2.43OSRImportFromESRI . . . 503
12.18.2.44OSRImportFromMICoordSys . . . 504
12.18.2.45OSRImportFromPCI . . . 504
12.18.2.46OSRImportFromProj4 . . . 504
12.18.2.47OSRImportFromUrl . . . 504
12.18.2.48OSRImportFromUSGS . . . 504
12.18.2.49OSRImportFromWkt . . . 504
12.18.2.50OSRImportFromXML . . . 505
12.18.2.51OSRIsCompound . . . 505
12.18.2.52OSRIsGeocentric . . . 505
12.18.2.53OSRIsGeographic . . . 505
12.18.2.54OSRIsLocal . . . 505
12.18.2.55OSRIsProjected . . . 505
12.18.2.56OSRIsSame . . . 506
12.18.2.57OSRIsSameGeogCS . . . 506
12.18.2.58OSRIsSameVertCS . . . 506
12.18.2.59OSRIsVertical . . . 506
12.18.2.60OSRMorphFromESRI . . . 506
12.18.2.61OSRMorphToESRI . . . 507
12.18.2.62OSRNewSpatialReference . . . 507
12.18.2.63OSRReference . . . 507
12.18.2.64OSRRelease . . . 507
12.18.2.65OSRSetACEA . . . 507
12.18.2.66OSRSetAE . . . 507
12.18.2.67OSRSetAngularUnits . . . 507
12.18.2.68OSRSetAttrValue . . . 508
12.18.2.69OSRSetAuthority . . . 508
12.18.2.70OSRSetBonne . . . 508
12.18.2.71OSRSetCEA . . . 508
12.18.2.72OSRSetCompoundCS . . . 508
12.18.2.73OSRSetCS . . . 508
12.18.2.74OSRSetEC . . . 509
12.18.2.75OSRSetEckert . . . 509
12.18.2.76OSRSetEckertIV . . . 509
12.18.2.77OSRSetEckertVI . . . 509
12.18.2.78OSRSetEquirectangular . . . 509

12.18.2.79 OSRSetEquirectangular2 . . . 509
12.18.2.80 OSRSetFromUserInput . . . 510
12.18.2.81 OSRSetGaussSchreiberTMercator . . . 510
12.18.2.82 OSRSetGeocCS . . . 510
12.18.2.83 OSRSetGeogCS . . . 510
12.18.2.84 OSRSetGEOS . . . 510
12.18.2.85 OSRSetGH . . . 510
12.18.2.86 OSRSetGnomonic . . . 511
12.18.2.87 OSRSetGS . . . 511
12.18.2.88 OSRSetHOM . . . 511
12.18.2.89 OSRSetHOM2PNO . . . 511
12.18.2.90 OSRSetIGH . . . 511
12.18.2.91 OSRSetIWMPolyconic . . . 511
12.18.2.92 OSRSetKrovak . . . 512
12.18.2.93 OSRSetLAEA . . . 512
12.18.2.94 OSRSetLCC . . . 512
12.18.2.95 OSRSetLCC1SP . . . 512
12.18.2.96 OSRSetLCCB . . . 512
12.18.2.97 OSRSetLinearUnits . . . 512
12.18.2.98 OSRSetLinearUnitsAndUpdateParameters . . . 512
12.18.2.99 OSRSetLocalCS . . . 513
12.18.2.100 OSRSetMC . . . 513
12.18.2.101 OSRSetMercator . . . 513
12.18.2.102 OSRSetMollweide . . . 513
12.18.2.103 OSRSetNormProjParm . . . 513
12.18.2.104 OSRSetNZMG . . . 513
12.18.2.105 OSRSetOrthographic . . . 514
12.18.2.106 OSRSetOS . . . 514
12.18.2.107 OSRSetPolyconic . . . 514
12.18.2.108 OSRSetProjCS . . . 514
12.18.2.109 OSRSetProjection . . . 514
12.18.2.110 OSRSetProjParm . . . 514
12.18.2.111 OSRSetPS . . . 514
12.18.2.112 OSRSetRobinson . . . 515
12.18.2.113 OSRSetSinusoidal . . . 515
12.18.2.114 OSRSetSOC . . . 515
12.18.2.115 OSRSetStatePlane . . . 515
12.18.2.116 OSRSetStatePlaneWithUnits . . . 515
12.18.2.117 OSRSetStereographic . . . 515
12.18.2.118 OSRSetTargetLinearUnits . . . 515

12.18.2.119 OSRSetTM 516
12.18.2.120 OSRSetTMG 516
12.18.2.121 OSRSetTMSO 516
12.18.2.122 OSRSetTMVariant 516
12.18.2.123 OSRSetTOWGS84 516
12.18.2.124 OSRSetUTM 516
12.18.2.125 OSRSetVDG 517
12.18.2.126 OSRSetVertCS 517
12.18.2.127 OSRSetWagner 517
12.18.2.128 OSRSetWellKnownGeogCS 517
12.18.2.129 OSRStripCTParms 517
12.18.2.130 OSRValidate 517
12.19 ogrsf_frmts.h File Reference 518
12.19.1 Detailed Description 518

Chapter 1

OGR Simple Feature Library

The OGR Simple Features Library is a C++ open source library (and commandline tools) providing read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, and Mapinfo mid/mif and TAB formats.

OGR is a part of the GDAL library.

Resources

- OGR Supported Formats : ESRI Shapefile, ESRI ArcSDE, MapInfo (tab and mid/mif), GML, KML, PostGIS, Oracle Spatial, ...
- OGR Utility Programs : ogrinfo, ogr2ogr, ogrtindex
- OGR Class Documentation
- OGR C++ API Read/Write Tutorial
- OGR Driver Implementation Tutorial
- ogr_api.h: OGR C API
- ogr_srs_api.h: OSR C API
- OGR Projections Tutorial
- OGR Architecture
- OGR SQL
- OGR - Feature Style Specification
- Adam's 2.5 D Simple Features Proposal (OGC 99-402r2)
- Adam's SRS WKT Clarification Proposal in html or doc format.

Download

Ready to Use Executables

The best way to get OGR utilities in ready-to-use form is to download the latest FWTools kit for your platform. While large, these include builds of the OGR utilities with lots of optional components built-in. Once downloaded follow the included instructions to setup your path and other environment variables correctly, and then you can use the various OGR utilities from the command line. The kits also include OpenEV, a viewer that will display OGR supported vector files.

Source

The source code for this effort is intended to be available as OpenSource using an X Consortium style license. The OGR library is currently a loosely coupled subcomponent of the GDAL library, so you get all of GDAL for the "price" of OGR. See the GDAL Download and Building pages for details on getting the source and building it.

Bug Reporting

GDAL/OGR bugs can be reported, and can be listed using Trac.

Mailing Lists

A gdal-announce mailing list subscription is a low volume way of keeping track of major developments with the GDAL/OGR project.

The gdal-dev@lists.osgeo.org mailing list can be used for discussion of development and user issues related to OGR and related technologies. Subscriptions can be done, and archives reviewed on the web.

Alternative Bindings for the OGR API

In addition to the C++ API primarily addressed in the online documentation, there is also a slightly less complete C API implemented on top of the C++ API, and access available from Python.

The C API is primarily intended to provide a less fragile API since slight changes in the C++ API (such as const correctness changes) can cause changes in method and class signatures that prevent use of new DLLs with older clients. The C API is also generally easy to call from other languages which allow call out to DLLs functions, such as Visual Basic, or Delphi. The API can be explored in the ogr_api.h include file. The gdal/ogr/ogr_capi_test.c is a small sample program demonstrating use of the C API.

The Python API isn't really well documented at this time, but parallels the C/C++ APIs. The interface classes can be browsed in the pymod/ogr.py (simple features) and pymod/osr.py (coordinate systems) python modules. The pymod/samples/assemblepoly.py sample script is one demonstration of using the python API.

Chapter 2

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.

2.1 Reading From OGR

For purposes of demonstrating reading with OGR, we will constuct 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 **OGRRegisterAll()** (p. 393) which registers all format drivers built into GDAL/OGR.

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
        OGRRegisterAll();
```

In C :

```
#include "ogr_api.h"

int main()

{
        OGRRegisterAll();
```

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 (FALSE) tells the **OGRSFDriverRegistrar::Open()** (p. 229) method that we don't require update access. On failure NULL is returned, and we report an error.

In C++ :

```
OGRDataSource       *poDS;

poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRDataSourceH hDS;

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

An **OGRDataSource** (p. 85) can potentially have many layers associated with it. The number of layers available can be queried with **OGRDataSource::GetLayerCount()** (p. 90) and individual layers fetched by index using **OGRDataSource::GetLayer()** (p. 89). However, we wil just fetch the layer by name.

In C++ :

```
OGRLayer  *poLayer;

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

In C :

```
OGRLayerH hLayer;

hLayer = OGR_DS_GetLayerByName( 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.

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()** (p. 172) to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using **OGRLayer::GetNextFeature()** (p. 170). It will return NULL when we run out of features.

In C++ :

```
OGRFeature *poFeature;

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

In 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** (p. 109). 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.

In C++ :

```
OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
int iField;

for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
{
    OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

    if( poFieldDefn->GetType() == OFTInteger )
        printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
    else if( poFieldDefn->GetType() == OFTReal )
        printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
    else if( poFieldDefn->GetType() == OFTString )
        printf( "%s,", poFeature->GetFieldAsString(iField) );
    else
        printf( "%s,", poFeature->GetFieldAsString(iField) );
}
```

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 );

    if( OGR_Fld_GetType(hFieldDefn) == OFTInteger )
        printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
    else if( OGR_Fld_GetType(hFieldDefn) == OFTReal )
        printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
    else if( OGR_Fld_GetType(hFieldDefn) == OFTString )
        printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
    else
        printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
}
```

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()** (p. 101) method. In fact we could shorten the above by using **OGRFeature::GetFieldAsString()** (p. 101) 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** (p. 126) 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 )
{
    OGRPoint *poPoint = (OGRPoint *) poGeometry;

    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.

Note that **OGRFeature::GetGeometryRef()** (p. 102) returns a pointer to the internal geometry owned by the **OGRFeature** (p. 94). There we don't actually deleted the return geometry. However, the **OGRLayer::GetNextFeature()** (p. 170) 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** (p. 162) returned by **OGRDataSource::GetLayerByName()** (p. 90) is also a reference to an internal layer owned by the **OGRDataSource** (p. 85) 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 issus.

In C++ :

```
    OGRDataSource::DestroyDataSource( poDS );
}
```

In C :

```
    OGR_DS_Destroy( hDS );
}
```

All together our program looks like this.

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
    OGRRegisterAll();

    OGRDataSource       *poDS;

    poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );
    if( poDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    OGRLayer  *poLayer;

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

    OGRFeature *poFeature;

    poLayer->ResetReading();
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
        int iField;

        for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

            if( poFieldDefn->GetType() == OFTInteger )
                printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            else if( poFieldDefn->GetType() == OFTReal )
                printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
            else if( poFieldDefn->GetType() == OFTString )
                printf( "%s,", poFeature->GetFieldAsString(iField) );
            else
                printf( "%s,", poFeature->GetFieldAsString(iField) );
        }

        OGRGeometry *poGeometry;

        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 );
    }

    OGRDataSource::DestroyDataSource( poDS );
}
```

In C :

```
#include "ogr_api.h"

int main()

{
    OGRRegisterAll();

    OGRDataSourceH hDS;
    OGRLayerH hLayer;
    OGRFeatureH hFeature;

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

    hLayer = OGR_DS_GetLayerByName( hDS, "point" );

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

        hFDefn = OGR_L_GetLayerDefn(hLayer);

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

            if( OGR_Fld_GetType(hFieldDefn) == OFTInteger )
                printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
            else if( OGR_Fld_GetType(hFieldDefn) == OFTReal )
                printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
            else if( OGR_Fld_GetType(hFieldDefn) == OFTString )
                printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
            else
                printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
        }

        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 );
    }

    OGR_DS_Destroy( hDS );
}
```

In Python:

```
import sys
import ogr

ds = ogr.Open( "point.shp" )
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:
            print "%d" % feat.GetFieldAsInteger(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
```

2.2 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 seperated 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";
    OGRSFDriver *poDriver;

    OGRRegisterAll();

    poDriver = OGRSFDriverRegistrar::GetRegistrar()->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";
    OGRSFDriverH hDriver;

    OGRRegisterAll();

    hDriver = OGRGetDriverByName( pszDriverName );
    if( hDriver == 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 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 ++ :

```
OGRDataSource *poDS;

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

In C :

```
OGRDataSourceH hDS;

hDS = OGR_Dr_CreateDataSource( hDriver, "point_out.shp", 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 = OGR_DS_CreateLayer( 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** (p. 116) 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** (p. 116) 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** (p. 94), set attributes and attach geometry before trying to write it to the layer. It is imperative that this feature be instantiated from the **OGRFeatureDefn** (p. 109) 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()** (p. 108) differs from **OGRFeature::SetGeometry()** (p. 107) 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()** (p. 165) 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++ :

```
    OGRDataSource::DestroyDataSource( poDS );
}
```

In C :

```
    OGR_DS_Destroy( hDS );
}
```

The same program all in one block looks like this:

In C++ :

```
#include "ogrsf_frmts.h"

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

    OGRRegisterAll();

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

    OGRDataSource *poDS;

    poDS = poDriver->CreateDataSource( "point_out.shp", 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 );
    }

    OGRDataSource::DestroyDataSource( poDS );
}
```

In C :

```
#include "ogr_api.h"

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

    OGRRegisterAll();

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

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

    hLayer = OGR_DS_CreateLayer( 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 );
   }

   OGR_DS_Destroy( hDS );
}
```

In Python :

```
import sys
import ogr
import string

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

ds = drv.CreateDataSource( "point_out.shp" )
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
```

Chapter 3

OGR Architecture

This document is intended to document the OGR classes. The OGR classes are intended to be generic (not specific to OLE DB or COM or Windows) but are used as a foundation for implementing OLE DB Provider support, as well as client side support for SFCOM. It is intended that these same OGR classes could be used by an implementation of SFCORBA for instance or used directly by C++ programs wanting to use an OpenGIS simple features inspired API.

Because OGR is modelled on the OpenGIS simple features data model, it is very helpful to review the SFCOM, or other simple features interface specifications which can be retrieved from the `Open Geospatial Consortium` web site. Data types, and method names are modelled on those from the interface specifications.

3.1 Class Overview

- **Geometry** (`ogr_geometry.h`): The geometry classes (**OGRGeometry** (p. 126), etc) encapsulate the OpenGIS model vector data as well as providing some geometry operations, and translation to/from well known binary and text format. A geometry includes a spatial reference system (projection).
- **Spatial Reference** (`ogr_spatialref.h`): An **OGRSpatialReference** (p. 230) encapsulates the definition of a projection and datum.
- **Feature** (`ogr_feature.h`): The **OGRFeature** (p. 94) encapsulate the definition of a whole feature, that is a geometry and a set of attributes.
- **Feature Class Definition** (`ogr_feature.h`): The **OGRFeatureDefn** (p. 109) class capsures the schema (set of field definitions) for a group of related features (normally a whole layer).
- **Layer** (`ogrsf_frmts.h`): **OGRLayer** (p. 162) is an abstract base class represent a layer of features in an **OGRDataSource** (p. 85).
- **Data Source** (`ogrsf_frmts.h`): An **OGRDataSource** (p. 85) is an abstract base class representing a file or database containing one or more **OGRLayer** (p. 162) objects.
- **Drivers** (`ogrsf_frmts.h`): An **OGRSFDriver** (p. 222) represents a translator for a specific format, opening **OGRDataSource** (p. 85) objects. All available drivers are managed by the **OGRSFDriverRegistrar** (p. 226).

3.2 Geometry

The geometry classes are represent various kinds of vector geometry. All the geometry classes derived from **OGRGeometry** (p. 126) which defines the common services of all geometries. Types of geometry include **OGRPoint** (p. 203), **OGRLineString** (p. 181), **OGRPolygon** (p. 211), **OGRGeometryCollection** (p. 146), **OGRMultiPolygon** (p. 200), **OGRMultiPoint** (p. 198), and **OGRMultiLineString** (p. 195).

Additional intermediate abstract base classes contain functionality that could eventually be implemented by other geometry types. These include **OGRCurve** (p. 83) (base class for **OGRLineString** (p. 181)) and **OGRSurface** (p. 293) (base class for **OGRPolygon** (p. 211)). Some intermediate interfaces modelled in the simple features abstract model and SFCOM are not modelled in OGR at this time. In most cases the methods are aggregated into other classes. This may change.

The **OGRGeometryFactory** (p. 156) is used to convert well known text, and well known binary format data into geometries. These are predefined ascii and binary formats for representing all the types of simple features geometries.

In a manner based on the geometry object in SFCOM, the **OGRGeometry** (p. 126) includes a reference to an **OGRSpatialReference** (p. 230) object, defining the spatial reference system of that geometry. This is normally a reference to a shared spatial reference object with reference counting for each of the **OGRGeometry** (p. 126) objects using it.

Many of the spatial analysis methods (such as computing overlaps and so forth) are not implemented at this time for **OGRGeometry** (p. 126).

While it is theoretically possible to derive other or more specific geometry classes from the existing **OGRGeometry** (p. 126) classes, this isn't as aspect that has been well thought out. In particular, it would be possible to create specialized classes using the **OGRGeometryFactory** (p. 156) without modifying it.

3.3 Spatial Reference

The **OGRSpatialReference** (p. 230) class is intended to store an OpenGIS Spatial Reference System definition. Currently local, geographic and projected coordinate systems are supported. Vertical coordinate systems, geocentric coordinate systems, and compound (horizontal + vertical) coordinate systems are not supported.

The spatial coordinate system data model is inherited from the OpenGIS **Well Known Text** format. A simple form of this is defined in the Simple Features specifications. A more sophisticated form is found in the Coordinate Transformation specification. The **OGRSpatialReference** (p. 230) is built on the features of the Coordinate Transformation specification but is intended to be compatible with the earlier simple features form.

There is also an associated **OGRCoordinateTransformation** (p. 81) class that encapsulates use of PROJ.4 for converting between different coordinate systems. There is a `tutorial` available describing how to use the **OGRSpatialReference** (p. 230) class.

3.4 Feature / Feature Definition

The **OGRGeometry** (p. 126) captures the geometry of a vector feature ... the spatial position/region of a feature. The **OGRFeature** (p. 94) contains this geometry, and adds feature attributes, feature id, and a feature class identify.

The set of attributes, their types, names and so forth is represented via the **OGRFeatureDefn** (p. 109) class. One **OGRFeatureDefn** (p. 109) normally exists for a layer of features. The same definition is shared in a reference counted manner by the feature of that type (or feature class).

The feature id (FID) of a feature is intended to be a unique identifier for the feature within the layer it is a member of. Freestanding features, or features not yet written to a layer may have a null (OGRNullFID) feature id. The feature ids are modelled in OGR as a long integer; however, this is not sufficiently expressive to model the natural feature ids in some formats. For instance, the GML feature id is a string, and the row id in Oracle is larger than 4 bytes.

The feature class also contains an indicator of the types of geometry allowed for that feature class (returned as an OGRwkbGeometryType from **OGRFeatureDefn::GetGeomType()** (p. 113)). If this is wkbUnknown then any type of geometry is allowed. This implies that features in a given layer can potentially be of different geometry types though they will always share a common attribute schema.

The **OGRFeatureDefn** (p. 109) also contains a concept of default spatial reference system for all features of that type and a feature class name (normally used as a layer name).

3.5 Layer

An **OGRLayer** (p. 162) represents a layer of features within a data source. All features in an **OGRLayer** (p. 162) share a common schema and are of the same **OGRFeatureDefn** (p. 109). An **OGRLayer** (p. 162) class also contains methods for reading features from the data source. The **OGRLayer** (p. 162) can be thought of as a gateway for reading and writing features from an underlying data source, normally a file format. In SFCOM and other table based simple features implementation an **OGRLayer** (p. 162) represents a spatial table.

The **OGRLayer** (p. 162) includes methods for sequential and random reading and writing. Read access (via the **OGRLayer::GetNextFeature()** (p. 170) method) normally reads all features, one at a time sequentially; however, it can be limited to return features intersecting a particular geographic region by installing a spatial filter on the **OGRLayer** (p. 162) (via the **OGRLayer::SetSpatialFilter()** (p. 174) method).

One flaw in the current OGR architecture is that the spatial filter is set directly on the **OGRLayer** (p. 162) which is intended to be the only representative of a given layer in a data source. This means it isn't possible to have multiple read operations active at one time with different spatial filters on each. This aspect may be revised in the future to introduct an OGRLayerView class or something similar.

Another question that might arise is why the **OGRLayer** (p. 162) and **OGRFeatureDefn** (p. 109) classes are distinct. An **OGRLayer** (p. 162) always has a one-to-one relationship to an **OGRFeatureDefn** (p. 109), so why not amalgamate the classes. There are two reasons:

1. As defined now **OGRFeature** (p. 94) and **OGRFeatureDefn** (p. 109) don't depend on **OGRLayer** (p. 162), so they can exist independently in memory without regard to a particular layer in a data store.

2. The SF CORBA model does not have a concept of a layer with a single fixed schema the way that the SFCOM and SFSQL models do. The fact that features belong to a feature collection that is potentially not directly related to their current feature grouping may be important to implementing SFCORBA support using OGR.

The **OGRLayer** (p. 162) class is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. OGRLayers are normally owned directly by their **OGRDataSource** (p. 85), and aren't instantiated or destroyed directly.

3.6 Data Source

An **OGRDataSource** (p. 85) represents a set of **OGRLayer** (p. 162) objects. This usually represents a single file, set of files, database or gateway. An **OGRDataSource** (p. 85) has a list of **OGRLayer** (p. 162)'s which it owns but can return references to.

OGRDataSource (p. 85) is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. **OGRDataSource** (p. 85) objects are not normally instantiated directly but rather with the assistance of an **OGRSFDriver** (p. 222). Deleting an **OGRDataSource** (p. 85) closes access to the underlying persistent data source, but does not normally result in deletion of that file.

An **OGRDataSource** (p. 85) has a name (usually a filename) that can be used to reopen the data source with an **OGRSFDriver** (p. 222).

The **OGRDataSource** (p. 85) also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the **OGRDataSource::ExecuteSQL()** (p. 89) method. While some datasources (such as PostGIS and Oracle) pass the SQL through to an underlying database, OGR also includes support for evaluating a subset of the SQL SELECT statement against any datasource.

3.7 Drivers

An **OGRSFDriver** (p. 222) object is instantiated for each file format supported. The **OGRSFDriver** (p. 222) objects are registered with the **OGRSFDriverRegistrar** (p. 226), a singleton class that is normally used to open new data sources.

It is intended that a new **OGRSFDriver** (p. 222) derived class be implemented for each file format to be supported (along with a file format specific **OGRDataSource** (p. 85), and **OGRLayer** (p. 162) classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate **OGRSFDriver** (p. 222) objects, and register them with the **OGRSFDriverRegistrar** (p. 226). When a data source is to be opened, the registrar will normally try each **OGRSFDriver** (p. 222) in turn, until one succeeds, returning an **OGRDataSource** (p. 85) object.

It is not intended that the **OGRSFDriverRegistrar** (p. 226) be derived from.

Chapter 4

OGR Driver Implementation Tutorial

4.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with subclasses of **OGRSFDriver** (p. 222), **OGRDataSource** (p. 85) and **OGRLayer** (p. 162). The **OGRSFDriver** (p. 222) subclass is registered with the **OGRSFDriverRegistrar** (p. 226) at runtime.

Before following this tutorial to implement an OGR driver, please review the `OGR Architecture` document carefully.

The tutorial will be based on implementing a simple ascii point format.

4.2 Contents

1. **Implementing OGRSFDriver** (p. 19)
2. **Basic Read Only Data Source** (p. 21)
3. **Read Only Layer** (p. 22)

4.3 Implementing OGRSFDriver

The format specific driver class is implemented as a subclass of **OGRSFDriver** (p. 222). One instance of the driver will normally be created, and registered with the OGRSFDriverRegistrar(). The instantiation of the driver is normally handled by a global C callable registration function, similar to the following placed in the same file as the driver class.

```
void RegisterOGRSPF()

{
    OGRSFDriverRegistrar::GetRegistrar()->RegisterDriver( new OGRSPFDriver );
}
```

The driver class declaration generally looks something like this for a format with read or read and update access (the Open() method), creation support (the CreateDataSource() method), and the ability to delete a datasource (the DeleteDataSource() method).

```
class OGRSPFDriver : public OGRSFDriver
{
  public:
                ~OGRSPFDriver();
```

```
    const char    *GetName();
    OGRDataSource *Open( const char *, int );
    OGRDataSource *CreateDataSource( const char *, char ** );
    OGRErr         DeleteDataSource( const char *pszName );
    int            TestCapability( const char * );
};
```

The constructor generally does nothing. The **OGRSFDriver::GetName()** (p. 224) method returns a static string with the name of the driver. This name is specified on the commandline when creating datasources so it is generally good to keep it short and without any special characters or spaces.

```
OGRSPFDriver::~OGRSPFDriver()

{
}

const char *OGRSPFDriver::GetName()
{
    return "SPF";
}
```

The Open() method is called by **OGRSFDriverRegistrar::Open()** (p. 229), or from the C API **OGROpen()** (p. 481). The **OGRSFDriver::Open()** (p. 225) method should quietly return NULL if the passed filename is not of the format supported by the driver. If it is the target format, then a new **OGRDataSource** (p. 85) object for the datasource should be returned.

It is common for the Open() method to be delegated to an Open() method on the actual format's **OGRDataSource** (p. 85) class.

```
OGRDataSource *OGRSPFDriver::Open( const char * pszFilename, int bUpdate )
{
    OGRSPFDataSource   *poDS = new OGRSPFDataSource();

    if( !poDS->Open( pszFilename, bUpdate ) )
    {
        delete poDS;
        return NULL;
    }
    else
        return poDS;
}
```

In OGR the capabilities of drivers, datasources and layers are determined by calling TestCapability() on the various objects with names strings representing specific optional capabilities. For the driver the only two capabilities currently tested for are the ability to create datasources and to delete them. In our first pass as a read only SPF driver, these are both disabled. The default return value for unrecognised capabilities should always be FALSE, and the symbolic #defines for capability names (defined in **ogr_core.h** (p. 483)) should be used instead of the literal strings to avoid typos.

```
int OGRSPFDriver::TestCapability( const char * pszCap )

{
    if( EQUAL(pszCap,ODrCCreateDataSource) )
        return FALSE;
    else if( EQUAL(pszCap,ODrCDeleteDataSource) )
        return FALSE;
    else
        return FALSE;
}
```

Examples of the CreateDataSource() and DeleteDataSource() methods are left for the section on creation and update.

4.4 Basic Read Only Data Source

We will start implementing a minimal read-only datasource. No attempt is made to optimize operations, and default implementations of many methods inherited from **OGRDataSource** (p. 85) are used.

The primary responsibility of the datasource is to manage the list of layers. In the case of the SPF format a datasource is a single file representing one layer so there is at most one layer. The "name" of a datasource should generally be the name passed to the Open() method.

The Open() method below is not overriding a base class method, but we have it to implement the open operation delegated by the driver class.

For this simple case we provide a stub TestCapability() that returns FALSE for all extended capabilities. The TestCapability() method is pure virtual, so it does need to be implemented.

```
class OGRSPFDataSource : public OGRDataSource
{
    char                *pszName;

    OGRSPFLayer       **papoLayers;
    int                 nLayers;

  public:
                        OGRSPFDataSource();
                        ~OGRSPFDataSource();

    int                 Open( const char * pszFilename, int bUpdate );

    const char          *GetName() { return pszName; }

    int                 GetLayerCount() { return nLayers; }
    OGRLayer            *GetLayer( int );

    int                 TestCapability( const char * ) { return FALSE; }
};
```

The constructor is a simple initializer to a default state. The Open() will take care of actually attaching it to a file. The destructor is responsible for orderly cleanup of layers.

```
OGRSPFDataSource::OGRSPFDataSource()

{
    papoLayers = NULL;
    nLayers = 0;

    pszName = NULL;
}

OGRSPFDataSource::~OGRSPFDataSource()

{
    for( int i = 0; i < nLayers; i++ )
        delete papoLayers[i];
    CPLFree( papoLayers );

    CPLFree( pszName );
}
```

The Open() method is the most important one on the datasource, though in this particular instance it passes most of it's work off to the OGRSPFLayer constructor if it believes the file is of the desired format.

Note that Open() methods should try and determine that a file isn't of the identified format as efficiently as possible, since many drivers may be invoked with files of the wrong format before the correct driver is reached. In this particular Open() we just test the file extension but this is generally a poor way of identifying a file format. If available, checking "magic header values" or something similar is preferrable.

In the case of the SPF format, update in place is not supported, so we always fail if bUpdate is FALSE.

```
int  OGRSPFDataSource::Open( const char *pszFilename, int bUpdate )

{
// ----------------------------------------------------------------------
//      Does this appear to be an .spf file?
// ----------------------------------------------------------------------
    if( !EQUAL( CPLGetExtension(pszFilename), "spf" ) )
        return FALSE;

    if( bUpdate )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
                  "Update access not supported by the SPF driver." );
        return FALSE;
    }

// ----------------------------------------------------------------------
//      Create a corresponding layer.
// ----------------------------------------------------------------------
    nLayers = 1;
    papoLayers = (OGRSPFLayer **) CPLMalloc(sizeof(void*));

    papoLayers[0] = new OGRSPFLayer( pszFilename );

    pszName = CPLStrdup( pszFilename );

    return TRUE;
}
```

A GetLayer() method also needs to be implemented. Since the layer list is created in the Open() this is just a lookup with some safety testing.

```
OGRLayer *OGRSPFDataSource::GetLayer( int iLayer )

{
    if( iLayer < 0 || iLayer >= nLayers )
        return NULL;
    else
        return papoLayers[iLayer];
}
```

4.5 Read Only Layer

The OGRSPFLayer is implements layer semantics for an .spf file. It provides access to a set of feature objects in a constistent coordinate system with a particular set of attribute columns. Our class definition looks like this:

```
class OGRSPFLayer : public OGRLayer
{
    OGRFeatureDefn     *poFeatureDefn;

    FILE               *fp;

    int                 nNextFID;

  public:
    OGRSPFLayer( const char *pszFilename );
   ~OGRSPFLayer();

    void                ResetReading();
    OGRFeature *        GetNextFeature();

    OGRFeatureDefn *    GetLayerDefn() { return poFeatureDefn; }

    int                 TestCapability( const char * ) { return FALSE; }
};
```

The layer constructor is responsible for initialization. The most important initialization is setting up the **OGRFeatureDefn** (p. 109) for the layer. This defines the list of fields and their types, the geometry type and the coordinate system

for the layer. In the SPF format the set of fields is fixed - a single string field and we have no coordinate system info to set.

Pay particular attention to the reference counting of the **OGRFeatureDefn** (p. 109). As **OGRFeature** (p. 94)'s for this layer will also take a reference to this definition it is important that we also establish a reference on behalf of the layer itself.

```
OGRSPFLayer::OGRSPFLayer( const char *pszFilename )

{
    nNextFID = 0;

    poFeatureDefn = new OGRFeatureDefn( CPLGetBasename( pszFilename ) );
    poFeatureDefn->Reference();
    poFeatureDefn->SetGeomType( wkbPoint );

    OGRFieldDefn oFieldTemplate( "Name", OFTString );

    poFeatureDefn->AddFieldDefn( &oFieldTemplate );

    fp = VSIFOpenL( pszFilename, "r" );
    if( fp == NULL )
        return;
}
```

Note that the destructor uses Release() on the **OGRFeatureDefn** (p. 109). This will destroy the feature definition if the reference count drops to zero, but if the application is still holding onto a feature from this layer, then that feature will hold a reference to the feature definition and it will not be destroyed here (which is good!).

```
OGRSPFLayer::~OGRSPFLayer()

{
    poFeatureDefn->Release();
    if( fp != NULL )
        VSIFCloseL( fp );
}
```

The GetNextFeature() method is usually the work horse of **OGRLayer** (p. 162) implementations. It is responsible for reading the next feature according to the current spatial and attribute filters installed.

The while() loop is present to loop until we find a satisfactory feature. The first section of code is for parsing a single line of the SPF text file and establishing the x, y and name for the line.

```
OGRFeature *OGRSPFLayer::GetNextFeature()

{
    // ----------------------------------------------------------------
    //  Loop till we find a feature matching our requirements.
    // ----------------------------------------------------------------
    while( TRUE )
    {
        const char *pszLine;
        const char *pszName;

        pszLine = CPLReadLineL( fp );

        // Are we at end of file (out of features)?
        if( pszLine == NULL )
            return NULL;

        double dfX;
        double dfY;

        dfX = atof(pszLine);

        pszLine = strstr(pszLine,"|");
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
```

```
            pszLine++;

        dfY = atof(pszLine);

        pszLine = strstr(pszLine,"|");
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
            pszName = pszLine+1;
```

The next section turns the x, y and name into a feature. Also note that we assign a linearly incremented feature id. In our case we started at zero for the first feature, though some drivers start at 1.

```
        OGRFeature *poFeature = new OGRFeature( poFeatureDefn );

        poFeature->SetGeometryDirectly( new OGRPoint( dfX, dfY ) );
        poFeature->SetField( 0, pszName );
        poFeature->SetFID( nNextFID++ );
```

Next we check if the feature matches our current attribute or spatial filter if we have them. Methods on the **OGRLayer** (p. 162) base class support maintain filters in the **OGRLayer** (p. 162) member fields m_poFilterGeom (spatial filter) and m_poAttrQuery (attribute filter) so we can just use these values here if they are non-NULL. The following test is essentially "stock" and done the same in all formats. Some formats also do some spatial filtering ahead of time using a spatial index.

If the feature meets our criteria we return it. Otherwise we destroy it, and return to the top of the loop to fetch another to try.

```
        if( (m_poFilterGeom == NULL
             || FilterGeometry( poFeature->GetGeometryRef() ) )
            && (m_poAttrQuery == NULL
                || m_poAttrQuery->Evaluate( poFeature )) )
            return poFeature;

        delete poFeature;
    }
}
```

While in the middle of reading a feature set from a layer, or at any other time the application can call ResetReading() which is intended to restart reading at the beginning of the feature set. We implement this by seeking back to the beginning of the file, and resetting our feature id counter.

```
void OGRSPFLayer::ResetReading()

{
    VSIFSeekL( fp, 0, SEEK_SET );
    nNextFID = 0;
}
```

In this implementation we do not provide a custom implementation for the GetFeature() method. This means an attempt to read a particular feature by it's feature id will result in many calls to GetNextFeature() till the desired feature is found. However, in a sequential text format like spf there is little else we could do anyways.

There! We have completed a simple read-only feature file format driver.

Chapter 5

OGR SQL

The **OGRDataSource** (p. 85) supports executing commands against a datasource via the **OGRDataSource::-ExecuteSQL()** (p. 89) method. While in theory any sort of command could be handled this way, in practice the mechanism is used to provide a subset of SQL SELECT capability to applications. This page discusses the generic SQL implementation implemented within OGR, and issue with driver specific SQL support.

The **OGRLayer** (p. 162) class also supports applying an attribute query filter to features returned using the **OGR-Layer::SetAttributeFilter()** (p. 173) method. The syntax for the attribute filter is the same as the WHERE clause in the OGR SQL SELECT statement. So everything here with regard to the WHERE clause applies in the context of the SetAttributeFilter() method.

NOTE: OGR SQL has been reimplemented for GDAL/OGR 1.8.0. Many features discussed below, notably arithmetic expressions, and expressions in the field list, were not support in GDAL/OGR 1.7.x and earlier. See RFC 28 for details of the new features in GDAL/OGR 1.8.0.

5.1 SELECT

The SELECT statement is used to fetch layer features (analygous to table rows in an RDBMS) with the result of the query represented as a temporary layer of features. The layers of the datasource are analygous to tables in an RDBMS and feature attributes are analygous to column values. The simpliest form of OGR SQL SELECT statement looks like this:

```
SELECT * FROM polylayer
```

In this case all features are fetched from the layer named "polylayer", and all attributes of those features are returned. This is essentially equivelent to accessing the layer directly. In this example the "$*$" is the list of fields to fetch from the layer, with "$*$" meaning that all fields should be fetched.

This slightly more sophisticated form still pulls all features from the layer but the schema will only contain the EAS_ID and PROP_VALUE attributes. Any other attributes would be discarded.

```
SELECT eas_id, prop_value FROM polylayer
```

A much more ambitious SELECT, restricting the features fetched with a WHERE clause, and sorting the results might look like:

```
SELECT * from polylayer WHERE prop_value > 220000.0 ORDER BY prop_value DESC
```

This select statement will produce a table with just one feature, with one attribute (named something like "count_-eas_id") containing the number of distinct values of the eas_id attribute.

```
SELECT COUNT(DISTINCT eas_id) FROM polylayer
```

5.1.1 Field List Operators

The field list is a comma separate list of the fields to be carried into the output features from the source layer. They will appear on output features in the order they appear on in the field list, so the field list may be used to re-order the fields.

A special form of the field list uses the DISTINCT keyword. This returns a list of all the distinct values of the named attribute. When the DISTINCT keyword is used, only one attribute may appear in the field list. The DISTINCT keyword may be used against any type of field. Currently the distinctness test against a string value is case insensitive in OGR SQL. The result of a SELECT with a DISTINCT keyword is a layer with one column (named the same as the field operated on), and one feature per distinct value. Geometries are discarded. The distinct values are assembled in memory, so alot of memory may be used for datasets with a large number of distinct values.

```
SELECT DISTINCT areacode FROM polylayer
```

There are also several summarization operators that may be applied to columns. When a summarization operator is applied to any field, then all fields must have summarization operators applied. The summarization operators are COUNT (a count of instances), AVG (numerical average), SUM (numericla sum), MIN (lexical or numerical minimum), and MAX (lexical or numerical maximum). This example produces a variety of sumarization information on parcel property values:

```
SELECT MIN(prop_value), MAX(prop_value), AVG(prop_value), SUM(prop_value),
       COUNT(prop_value) FROM polylayer WHERE prov_name = "Ontario"
```

It is also possible to apply the COUNT() operator to a DISTINCT SELECT to get a count of distinct values, for instance:

```
SELECT COUNT(DISTINCT areacode) FROM polylayer
```

Note: prior to OGR 1.9.0, null values were counted in COUNT(column_name) or COUNT(DISTINCT column_name), which was not conformant with the SQL standard. Since OGR 1.9.0, only non-null values are counted.

As a special case, the COUNT() operator can be given a "$*$" argument instead of a field name which is a short form for count all the records.

```
SELECT COUNT(*) FROM polylayer
```

Field names can also be prefixed by a table name though this is only really meaningful when performing joins. It is further demonstrated in the JOIN section.

Field definitions can also be complex expressions using arithmetic, and functional operators. However, the DISTINCT keyword, and summarization operators like MIN, MAX, AVG and SUM may not be applied to expression fields.

```
SELECT cost+tax from invoice
```

or

```
SELECT CONCAT(owner_first_name,' ',owner_last_name) from properties
```

5.1.1.1 Functions

Starting with OGR 1.8.2, the SUBSTR function can be used to extract a substring from a string. Its syntax is the following one : SUBSTR(string_expr, start_offset [, length]). It extracts a substring of string_expr, starting at offset start_offset (1 being the first character of string_expr, 2 the second one, etc...). If start_offset is a negative value, the substring is extracted from the end of the string (-1 is the last character of the string, -2 the character before the last character, ...). If length is specified, up to length characters are extracted from the string. Otherwise the remainder of the string is extracted.

Note: for the time being, the character as considered to be equivalent to bytes, which may not be appropriate for multi-byte encodings like UTF-8.

```
SELECT SUBSTR('abcdef',1,2) FROM xxx   --> 'ab'
SELECT SUBSTR('abcdef',4)   FROM xxx   --> 'def'
SELECT SUBSTR('abcdef',-2)  FROM xxx   --> 'ef'
```

5.1.1.2 Using the field name alias

OGR SQL supports renaming the fields following the SQL92 specification by using the AS keyword according to the following example:

```
SELECT *, OGR_STYLE AS STYLE FROM polylayer
```

The field name alias can be used as the last operation in the column specification. Therefore we cannot rename the fields inside an operator, but we can rename whole column expression, like these two:

```
SELECT COUNT(areacode) AS 'count' FROM polylayer
SELECT dollars/100.0 AS cents FROM polylayer
```

5.1.1.3 Changing the type of the fields

Starting with GDAL 1.6.0, OGR SQL supports changing the type of the columns by using the SQL92 compliant CAST operator according to the following example:

```
SELECT *, CAST(OGR_STYLE AS character(255)) FROM rivers
```

Currently casting to the following target types are supported:

1. character(field_length). By default, field_length=1.
2. float(field_length)
3. numeric(field_length, field_precision)
4. integer(field_length)
5. date(field_length)
6. time(field_length)
7. timestamp(field_length)

Specifying the field_length and/or the field_precision is optional. An explicit value of zero can be used as the width for character() to indicate variable width. Conversion to the 'integer list', 'double list' and 'string list' OGR data types are not supported, which doesn't conform to the SQL92 specification.

While the CAST operator can be applied anywhere in an expression, including in a WHERE clause, the detailed control of output field format is only supported if the CAST operator is the "outer most" operators on a field in the field definition list. In other contexts it is still useful to convert between numeric, string and date data types.

5.1.2 WHERE

The argument to the WHERE clause is a logical expression used select records from the source layer. In addition to its use within the WHERE statement, the WHERE clause handling is also used for OGR attribute queries on regular layers via **OGRLayer::SetAttributeFilter()** (p. 173).

In addition to the arithmetic and other functional operators available in expressions in the field selection clause of the SELECT statement, in the WHERE context logical operators are also available and the evaluated value of the expression should be logical (true or false).

The available logical operators are **=**, **!=**, <>, <, >, <**=**, >**=**, **LIKE** and **ILIKE**, **BETWEEN** and **IN**. Most of the operators are self explanitory, but is is worth nothing that **!=** is the same as <>, the string equality is case insensitive, but the <, >, <**=** and >**=** operators *are* case sensitive. Both the LIKE and ILIKE operators are case insensitive.

The value argument to the **LIKE** operator is a pattern against which the value string is matched. In this pattern percent (%) matches any number of characters, and underscore (_) matches any one character. An optional ESCAPE escape_char clause can be added so that the percent or underscore characters can be searched as regular characters, by being preceded with the escape_char.

```
String             Pattern       Matches?
------             -------       --------
Alberta            ALB%          Yes
Alberta            _lberta       Yes
St. Alberta        _lberta       No
St. Alberta        %lberta       Yes
Robarts St.        %Robarts%     Yes
12345              123%45        Yes
123.45             12?45         No
N0N 1P0            %N0N%         Yes
L4C 5E2            %N0N%         No
```

The **IN** takes a list of values as it's argument and tests the attribute value for membership in the provided set.

```
Value              Value Set            Matches?
------             -------              --------
321                IN (456,123)         No
"Ontario"          IN ("Ontario","BC")  Yes
"Ont"              IN ("Ontario","BC")  No
1                  IN (0,2,4,6)         No
```

The syntax of the **BETWEEN** operator is "field_name BETWEEN value1 AND value2" and it is equivalent to "field_name >= value1 AND field_name <= value2".

In addition to the above binary operators, there are additional operators for testing if a field is null or not. These are the **IS NULL** and **IS NOT NULL** operators.

Basic field tests can be combined in more complicated predicates using logical operators include **AND**, **OR**, and the unary logical **NOT**. Subexpressions should be bracketed to make precidence clear. Some more complicated predicates are:

```
SELECT * FROM poly WHERE (prop_value >= 100000) AND (prop_value < 200000)
SELECT * FROM poly WHERE NOT (area_code LIKE "N0N%")
SELECT * FROM poly WHERE (prop_value IS NOT NULL) AND (prop_value < 100000)
```

5.1.3 WHERE Limitations

1. Fields must all come from the primary table (the one listed in the FROM clause).
2. All string comparisons are case insensitive except for <, >, <**=** and >**=**.

5.1.4 ORDER BY

The **ORDER BY** clause is used force the returned features to be reordered into sorted order (ascending or descending) on one of the field values. Ascending (increasing) order is the default if neither the ASC or DESC keyword is provided. For example:

```
SELECT * FROM property WHERE class_code = 7 ORDER BY prop_value DESC
SELECT * FROM property ORDER BY prop_value
SELECT * FROM property ORDER BY prop_value ASC
SELECT DISTINCT zip_code FROM property ORDER BY zip_code
```

Note that ORDER BY clauses cause two passes through the feature set. One to build an in-memory table of field values corresponded with feature ids, and a second pass to fetch the features by feature id in the sorted order. For formats which cannot efficiently randomly read features by feature id this can be a very expensive operation.

Sorting of string field values is case sensitive, not case insensitive like in most other parts of OGR SQL.

5.1.5 JOINs

OGR SQL supports a limited form of one to one JOIN. This allows records from a secondary table to be looked up based on a shared key between it and the primary table being queried. For instance, a table of city locations might include a *nation_id* column that can be used as a reference into a secondary *nation* table to fetch a nation name. A joined query might look like:

```
SELECT city.*, nation.name FROM city
     LEFT JOIN nation ON city.nation_id = nation.id
```

This query would result in a table with all the fields from the city table, and an additional "nation.name" field with the nation name pulled from the nation table by looking for the record in the nation table that has the "id" field with the same value as the city.nation_id field.

Joins introduce a number of additional issues. One is the concept of table qualifiers on field names. For instance, referring to city.nation_id instead of just nation_id to indicate the nation_id field from the city layer. The table name qualifiers may only be used in the field list, and within the **ON** clause of the join.

Wildcards are also somewhat more involved. All fields from the primary table (*city* in this case) and the secondary table (*nation* in this case) may be selected using the usual $*$ wildcard. But the fields of just one of the primary or secondary table may be selected by prefixing the asterix with the table name.

The field names in the resulting query layer will be qualified by the table name, if the table name is given as a qualifier in the field list. In addition field names will be qualified with a table name if they would conflict with earlier fields. For instance, the following select would result might result in a results set with a *name, nation_id, nation.nation_id* and *nation.name* field if the city and nation tables both have the *nation_id* and *name* fieldnames.

```
SELECT * FROM city LEFT JOIN nation ON city.nation_id = nation.nation_id
```

On the other hand if the nation table had a *continent_id* field, but the city table did not, then that field would not need to be qualified in the result set. However, if the selected instead looked like the following statement, all result fields would be qualified by the table name.

```
SELECT city.*, nation.* FROM city
    LEFT JOIN nation ON city.nation_id = nation.nation_id
```

In the above examples, the *nation* table was found in the same datasource as the *city* table. However, the OGR join support includes the ability to join against a table in a different data source, potentially of a different format. This is indicated by qualifying the secondary table name with a datasource name. In this case the secondary datasource is opened using normal OGR semantics and utilized to access the secondary table untill the query result is no longer needed.

```
SELECT * FROM city
  LEFT JOIN '/usr2/data/nation.dbf'.nation ON city.nation_id = nation.nation_id
```

While not necessarily very useful, it is also possible to introduce table aliases to simplify some SELECT statements. This can also be useful to disambiguate situations where ables of the same name are being used from different data sources. For instance, if the actual tables names were messy we might want to do something like:

```
SELECT c.name, n.name FROM project_615_city c
  LEFT JOIN '/usr2/data/project_615_nation.dbf'.project_615_nation n
            ON c.nation_id = n.nation_id
```

It is possible to do multiple joins in a single query.

```
SELECT city.name, prov.name, nation.name FROM city
  LEFT JOIN province ON city.prov_id = province.id
  LEFT JOIN nation ON city.nation_id = nation.id
```

5.1.6 JOIN Limitations

1. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.
2. Joined fields may not be used in WHERE clauses, or ORDER BY clauses at this time. The join is essentially evaluated after all primary table subsetting is complete, and after the ORDER BY pass.
3. Joined fields may not be used as keys in later joins. So you could not use the province id in a city to lookup the province record, and then use a nation id from the province id to lookup the nation record. This is a sensible thing to want and could be implemented, but is not currently supported.
4. Datasource names for joined tables are evaluated relative to the current processes working directory, not the path to the primary datasource.
5. These are not true LEFT or RIGHT joins in the RDBMS sense. Whether or not a secondary record exists for the join key or not, one and only one copy of the primary record is returned in the result set. If a secondary record cannot be found, the secondary derived fields will be NULL. If more than one matching secondary field is found only the first will be used.

5.2 SPECIAL FIELDS

The OGR SQL query processor treats some of the attributes of the features as built-in special fields can be used in the SQL statements likewise the other fields. These fields can be placed in the select list, the WHERE clause and the ORDER BY clause respectively. The special field will not be included in the result by default but it may be explicitly included by adding it to the select list. When accessing the field values the special fields will take pecedence over the other fields with the same names in the data source.

5.2.1 FID

Normally the feature id is a special property of a feature and not treated as an attribute of the feature. In some cases it is convenient to be able to utilize the feature id in queries and result sets as a regular field. To do so use the name **FID**. The field wildcard expansions will not include the feature id, but it may be explicitly included using a syntax like:

```
SELECT FID, * FROM nation
```

5.2.2 OGR_GEOMETRY

Some of the data sources (like MapInfo tab) can handle geometries of different types within the same layer. The **OGR_GEOMETRY** special field represents the geometry type returned by **OGRGeometry::getGeometryName()** (p. 137) and can be used to distinguish the various types. By using this field one can select particular types of the geometries like:

```
SELECT * FROM nation WHERE OGR_GEOMETRY='POINT' OR OGR_GEOMETRY='POLYGON'
```

5.2.3 OGR_GEOM_WKT

The Well Known Text representation of the geometry can also be used as a special field. To select the WKT of the geometry **OGR_GEOM_WKT** might be included in the select list, like:

```
SELECT OGR_GEOM_WKT, * FROM nation
```

Using the **OGR_GEOM_WKT** and the **LIKE** operator in the WHERE clause we can get similar effect as using OGR_GEOMETRY:

```
SELECT OGR_GEOM_WKT, * FROM nation WHERE OGR_GEOM_WKT
   LIKE 'POINT%' OR OGR_GEOM_WKT LIKE 'POLYGON%'
```

5.2.4 OGR_GEOM_AREA

(Since GDAL 1.7.0)

The **OGR_GEOM_AREA** special field returns the area of the feature's geometry computed by the **OGRSurface::get_Area()** (p. 293) method. For **OGRGeometryCollection** (p. 146) and **OGRMultiPolygon** (p. 200) the value is the sum of the areas of its members. For non-surface geometries the returned area is 0.0.

For example, to select only polygon features larger than a given area:

```
SELECT * FROM nation WHERE OGR_GEOM_AREA > 10000000'
```

5.2.5 OGR_STYLE

The **OGR_STYLE** special field represents the style string of the feature returned by **OGRFeature::GetStyleString()** (p. 103). By using this field and the **LIKE** operator the result of the query can be filtered by the style. For example we can select the annotation features as:

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

5.3 CREATE INDEX

Some OGR SQL drivers support creating of attribute indexes. Currently this includes the Shapefile driver. An index accelerates very simple attribute queries of the form *fieldname = value*, which is what is used by the **JOIN** capability. To create an attribute index on the nation_id field of the nation table a command like this would be used:

```
CREATE INDEX ON nation USING nation_id
```

5.3.1 Index Limitations

1. Indexes are not maintained dynamically when new features are added to or removed from a layer.
2. Very long strings (longer than 256 characters?) cannot currently be indexed.
3. To recreate an index it is necessary to drop all indexes on a layer and then recreate all the indexes.
4. Indexes are not used in any complex queries. Currently the only query the will accelerate is a simple "field = value" query.

5.4 DROP INDEX

The OGR SQL DROP INDEX command can be used to drop all indexes on a particular table, or just the index for a particular column.

```
DROP INDEX ON nation USING nation_id
DROP INDEX ON nation
```

5.5 ALTER TABLE

(OGR >= 1.9.0)

The following OGR SQL ALTER TABLE commands can be used.

1. "ALTER TABLE tablename ADD [COLUMN] columnname columntype" to add a new field. Supported if the layer declares the OLCCreateField capability.

2. "ALTER TABLE tablename RENAME [COLUMN] oldcolumnname TO newcolumnname" to rename an existing field. Supported if the layer declares the OLCAlterFieldDefn capability.
3. "ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype" to change the type of an existing field. Supported if the layer declares the OLCAlterFieldDefn capability.
4. "ALTER TABLE tablename DROP [COLUMN] columnname" to delete an existing field. Supported if the layer declares the OLCDeleteField capability.

The columntype value follows the syntax of the types supported by the CAST operator descrived above.

```
ALTER TABLE nation ADD COLUMN myfield integer
ALTER TABLE nation RENAME COLUMN myfield TO myfield2
ALTER TABLE nation ALTER COLUMN myfield2 TYPE character(15)
ALTER TABLE nation DROP COLUMN myfield2
```

5.6 DROP TABLE

(OGR >= 1.9.0)

The OGR SQL DROP TABLE command can be used to delete a table. This is only supported on datasources that declare the ODsCDeleteLayer capability.

```
DROP TABLE nation
```

5.7 ExecuteSQL()

SQL is executed against an **OGRDataSource** (p. 85), not against a specific layer. The call looks like this:

```
OGRLayer * OGRDataSource::ExecuteSQL( const char *pszSQLCommand,
                                      OGRGeometry *poSpatialFilter,
                                      const char *pszDialect );
```

The pszDialect argument is in theory intended to allow for support of different command languages against a provider, but for now applications should always pass an empty (not NULL) string to get the default dialect.

The poSpatialFilter argument is a geometry used to select a bounding rectangle for features to be returned in a manner similar to the **OGRLayer::SetSpatialFilter()** (p. 174) method. It may be NULL for no special spatial restriction.

The result of an ExecuteSQL() call is usually a temporary **OGRLayer** (p. 162) representing the results set from the statement. This is the case for a SELECT statement for instance. The returned temporary layer should be released with OGRDataSource::ReleaseResultsSet() method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

5.8 Non-OGR SQL

All OGR drivers for database systems: `MySQL`, PostgreSQL and PostGIS (`PG`), Oracle (`OCI`), `SQLite`, `ODBC`, ESRI Personal Geodatabase (`PGeo`) and MS SQL Spatial (`MSSQLSpatial`), override the **OGRDataSource::ExecuteSQL()** (p. 89) function with dedicated implementation and, by default, pass the SQL statements directly to the underlying RDBMS. In these cases the SQL syntax varies in some particulars from OGR SQL. Also, anything possible in SQL can then be accomplished for these particular databases. Only the result of SQL WHERE statements will be returned as layers.

Chapter 6

OGR Projections Tutorial

6.1 Introduction

The **OGRSpatialReference** (p. 230), and **OGRCoordinateTransformation** (p. 81) classes provide services to represent coordinate systems (projections and datums) and to transform between them. These services are loosely modelled on the OpenGIS Coordinate Transformations specification, and use the same Well Known Text format for describing coordinate systems.

Some background on OpenGIS coordinate systems and services can be found in the Simple Features for COM, and Spatial Reference Systems Abstract Model documents available from the `Open Geospatial Consortium`. The `GeoTIFF Projections Transform List` may also be of assistance in understanding formulations of projections in WKT. The `EPSG` Geodesy web page is also a useful resource.

6.2 Defining a Geographic Coordinate System

Coordinate systems are encapsulated in the **OGRSpatialReference** (p. 230) class. There are a number of ways of initializing an **OGRSpatialReference** (p. 230) object to a valid coordinate system. There are two primary kinds of coordinate systems. The first is geographic (positions are measured in long/lat) and the second is projected (such as UTM - positions are measured in meters or feet).

A Geographic coordinate system contains information on the datum (which implies an spheroid described by a semi-major axis, and inverse flattening), prime meridian(normally Greenwich), and an angular units type which is normally degrees. The following code initializes a geographic coordinate system on supplying all this information along with a user visible name for the geographic coordinate system.

```
OGRSpatialReference oSRS;

oSRS.SetGeogCS( "My geographic coordinate system",
                "WGS_1984",
                "My WGS84 Spheroid",
                SRS_WGS84_SEMIMAJOR, SRS_WGS84_INVFLATTENING,
                "Greenwich", 0.0,
                "degree", SRS_UA_DEGREE_CONV );
```

Of these values, the names "My geographic coordinate system", "My WGS84 Spheroid", "Greenwich" and "degree" are not keys, but are used for display to the user. However, the datum name "WGS_1984" is used as a key to identify the datum, and there are rules on what values can be used. NOTE: Prepare writeup somewhere on valid datums!

The **OGRSpatialReference** (p. 230) has built in support for a few well known coordinate systems, which include "NAD27", "NAD83", "WGS72" and "WGS84" which can be defined in a single call to SetWellKnownGeogCS().

```
oSRS.SetWellKnownGeogCS( "WGS84" );
```

Furthermore, any geographic coordinate system in the EPSG database can be set by it's GCS code number if the EPSG database is available.

```
oSRS.SetWellKnownGeogCS( "EPSG:4326" );
```

For serializization, and transmission of projection definitions to other packages, the OpenGIS Well Known Text format for coordinate systems is used. An **OGRSpatialReference** (p. 230) can be initialized from well known text, or converted back into well known text.

```
char    *pszWKT = NULL;

oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.exportToWkt( &pszWKT );
printf( "%s\n", pszWKT );
```

gives something like:

```
GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
AUTHORITY["EPSG",7030]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY["EPSG",6326]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],UNIT["DMSH",0.0174532925199433,
AUTHORITY["EPSG",9108]],AXIS["Lat",NORTH],AXIS["Long",EAST],AUTHORITY["EPSG",
4326]]
```

or in more readable form:

```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG",7030]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG",6326]],
    PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
    UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
    AXIS["Lat",NORTH],
    AXIS["Long",EAST],
    AUTHORITY["EPSG",4326]]
```

The **OGRSpatialReference::importFromWkt()** (p. 260) method can be used to set an **OGRSpatialReference** (p. 230) from a WKT coordinate system definition.

6.3 Defining a Projected Coordinate System

A projected coordinate system (such as UTM, Lambert Conformal Conic, etc) requires and underlying geographic coordinate system as well as a definition for the projection transform used to translate between linear positions (in meters or feet) and angular long/lat positions. The following code defines a UTM zone 17 projected coordinate system with and underlying geographic coordinate system (datum) of WGS84.

```
OGRSpatialReference     oSRS;

oSRS.SetProjCS( "UTM 17 (WGS84) in northern hemisphere." );
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.SetUTM( 17, TRUE );
```

Calling SetProjCS() sets a user name for the projected coordinate system and establishes that the system is projected. The SetWellKnownGeogCS() associates a geographic coordinate system, and the SetUTM() call sets detailed projection transformation parameters. At this time the above order is important in order to create a valid definition, but in the future the object will automatically reorder the internal representation as needed to remain valid. For now **be careful of the order of steps defining an OGRSpatialReference!**

The above definition would give a WKT version that looks something like the following. Note that the UTM 17 was expanded into the details transverse mercator definition of the UTM zone.

```
PROJCS["UTM 17 (WGS84) in northern hemisphere.",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG",7030]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG",6326]],
        PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
        UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
        AXIS["Lat",NORTH],
        AXIS["Long",EAST],
        AUTHORITY["EPSG",4326]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-81],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0]]
```

There are methods for many projection methods including SetTM() (Transverse Mercator), SetLCC() (Lambert Conformal Conic), and SetMercator().

6.4 Querying Coordinate System

Once an **OGRSpatialReference** (p. 230) has been established, various information about it can be queried. It can be established if it is a projected or geographic coordinate system using the **OGRSpatialReference::IsProjected()** (p. 262) and **OGRSpatialReference::IsGeographic()** (p. 261) methods. The **OGRSpatialReference::GetSemiMajor()** (p. 247), **OGRSpatialReference::GetSemiMinor()** (p. 248) and **OGRSpatialReference::GetInvFlattening()** (p. 245) methods can be used to get information about the spheroid. The **OGRSpatialReference::GetAttrValue()** (p. 243) method can be used to get the PROJCS, GEOGCS, DATUM, SPHEROID, and PROJECTION names strings. The **OGRSpatialReference::GetProjParm()** (p. 247) method can be used to get the projection parameters. The **OGRSpatialReference::GetLinearUnits()** (p. 246) method can be used to fetch the linear units type, and translation to meters.

The following code (from ogr_srs_proj4.cpp) demonstrates use of GetAttrValue() to get the projection, and GetProjParm() to get projection parameters. The GetAttrValue() method searches for the first "value" node associated with the named entry in the WKT text representation. The #define'ed constants for projection parameters (such as SRS_PP_CENTRAL_MERIDIAN) should be used when fetching projection parameter with GetProjParm(). The code for the Set methods of the various projections in ogrspatialreference.cpp can be consulted to find which parameters apply to which projections.

```
const char *pszProjection = poSRS->GetAttrValue("PROJECTION");

if( pszProjection == NULL )
{
    if( poSRS->IsGeographic() )
        sprintf( szProj4+strlen(szProj4), "+proj=longlat " );
    else
        sprintf( szProj4+strlen(szProj4), "unknown " );
}
else if( EQUAL(pszProjection,SRS_PT_CYLINDRICAL_EQUAL_AREA) )
{
    sprintf( szProj4+strlen(szProj4),
       "+proj=cea +lon_0=%.9f +lat_ts=%.9f +x_0=%.3f +y_0=%.3f ",
            poSRS->GetProjParm(SRS_PP_CENTRAL_MERIDIAN,0.0),
            poSRS->GetProjParm(SRS_PP_STANDARD_PARALLEL_1,0.0),
            poSRS->GetProjParm(SRS_PP_FALSE_EASTING,0.0),
            poSRS->GetProjParm(SRS_PP_FALSE_NORTHING,0.0) );
}
...
```

6.5 Coordinate Transformation

The **OGRCoordinateTransformation** (p. 81) class is used for translating positions between different coordinate systems. New transformation objects are created using **OGRCreateCoordinateTransformation()** (p. 490), and then the **OGRCoordinateTransformation::Transform()** (p. 82) method can be used to convert points between coordinate systems.

```
OGRSpatialReference oSourceSRS, oTargetSRS;
OGRCoordinateTransformation *poCT;
double                  x, y;

oSourceSRS.importFromEPSG( atoi(papszArgv[i+1]) );
oTargetSRS.importFromEPSG( atoi(papszArgv[i+2]) );

poCT = OGRCreateCoordinateTransformation( &oSourceSRS,
                                          &oTargetSRS );
x = atof( papszArgv[i+3] );
y = atof( papszArgv[i+4] );

if( poCT == NULL || !poCT->Transform( 1, &x, &y ) )
    printf( "Transformation failed.\n" );
else
    printf( "(%f,%f) -> (%f,%f)\n",
            atof( papszArgv[i+3] ),
            atof( papszArgv[i+4] ),
            x, y );
```

There are a couple of points at which transformations can fail. First, **OGRCreateCoordinateTransformation()** (p. 490) may fail, generally because the internals recognise that no transformation between the indicated systems can be established. This might be due to use of a projection not supported by the internal PROJ.4 library, differing datums for which no relationship is known, or one of the coordinate systems being inadequately defined. If **OGRCreateCoordinateTransformation()** (p. 490) fails it will return a NULL.

The **OGRCoordinateTransformation::Transform()** (p. 82) method itself can also fail. This may be as a delayed result of one of the above problems, or as a result of an operation being numerically undefined for one or more of the passed in points. The Transform() function will return TRUE on success, or FALSE if any of the points fail to transform. The point array is left in an indeterminate state on error.

Though not shown above, the coordinate transformation service can take 3D points, and will adjust elevations for elevation differents in spheroids, and datums. At some point in the future shifts between different vertical datums may also be applied. If no Z is passed, it is assume that the point is on the geoide.

The following example shows how to conveniently create a lat/long coordinate system using the same geographic coordinate system as a projected coordinate system, and using that to transform between projected coordinates and lat/long.

```
OGRSpatialReference    oUTM, *poLatLong;
OGRCoordinateTransformation *poTransform;

oUTM.SetProjCS("UTM 17 / WGS84");
oUTM.SetWellKnownGeogCS( "WGS84" );
oUTM.SetUTM( 17 );

poLatLong = oUTM.CloneGeogCS();

poTransform = OGRCreateCoordinateTransformation( &oUTM, poLatLong );
if( poTransform == NULL )
{
    ...
}

...

if( !poTransform->Transform( nPoints, x, y, z ) )
...
```

6.6 Alternate Interfaces

A C interface to the coordinate system services is defined in **ogr_srs_api.h** (p. 490), and Python bindings are available via the osr.py module. Methods are close analogs of the C++ methods but C and Python bindings are missing for some C++ methods.

C Bindings

```
typedef void *OGRSpatialReferenceH;
typedef void *OGRCoordinateTransformationH;

OGRSpatialReferenceH OSRNewSpatialReference( const char * );
void    OSRDestroySpatialReference( OGRSpatialReferenceH );

int     OSRReference( OGRSpatialReferenceH );
int     OSRDereference( OGRSpatialReferenceH );

OGRErr  OSRImportFromEPSG( OGRSpatialReferenceH, int );
OGRErr  OSRImportFromWkt( OGRSpatialReferenceH, char ** );
OGRErr  OSRExportToWkt( OGRSpatialReferenceH, char ** );

OGRErr  OSRSetAttrValue( OGRSpatialReferenceH hSRS, const char * pszNodePath,
                         const char * pszNewNodeValue );
const char *OSRGetAttrValue( OGRSpatialReferenceH hSRS,
                             const char * pszName, int iChild);

OGRErr  OSRSetLinearUnits( OGRSpatialReferenceH, const char *, double );
double  OSRGetLinearUnits( OGRSpatialReferenceH, char ** );

int     OSRIsGeographic( OGRSpatialReferenceH );
int     OSRIsProjected( OGRSpatialReferenceH );
int     OSRIsSameGeogCS( OGRSpatialReferenceH, OGRSpatialReferenceH );
int     OSRIsSame( OGRSpatialReferenceH, OGRSpatialReferenceH );

OGRErr  OSRSetProjCS( OGRSpatialReferenceH hSRS, const char * pszName );
OGRErr  OSRSetWellKnownGeogCS( OGRSpatialReferenceH hSRS,
                               const char * pszName );

OGRErr  OSRSetGeogCS( OGRSpatialReferenceH hSRS,
                      const char * pszGeogName,
                      const char * pszDatumName,
                      const char * pszEllipsoidName,
                      double dfSemiMajor, double dfInvFlattening,
                      const char * pszPMName ,
                      double dfPMOffset ,
                      const char * pszUnits,
                      double dfConvertToRadians );

double  OSRGetSemiMajor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetSemiMinor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetInvFlattening( OGRSpatialReferenceH, OGRErr * );

OGRErr  OSRSetAuthority( OGRSpatialReferenceH hSRS,
                         const char * pszTargetKey,
                         const char * pszAuthority,
                         int nCode );
OGRErr  OSRSetProjParm( OGRSpatialReferenceH, const char *, double );
double  OSRGetProjParm( OGRSpatialReferenceH hSRS,
                        const char * pszParmName,
                        double dfDefault,
                        OGRErr * );

OGRErr  OSRSetUTM( OGRSpatialReferenceH hSRS, int nZone, int bNorth );
int     OSRGetUTMZone( OGRSpatialReferenceH hSRS, int *pbNorth );

OGRCoordinateTransformationH
OCTNewCoordinateTransformation( OGRSpatialReferenceH hSourceSRS,
                                OGRSpatialReferenceH hTargetSRS );
void OCTDestroyCoordinateTransformation( OGRCoordinateTransformationH );

int OCTTransform( OGRCoordinateTransformationH hCT,
                  int nCount, double *x, double *y, double *z );
```

Python Bindings

```
class osr.SpatialReference
    def __init__(self,obj=None):
    def ImportFromWkt( self, wkt ):
    def ExportToWkt(self):
    def ImportFromEPSG(self,code):
    def IsGeographic(self):
    def IsProjected(self):
    def GetAttrValue(self, name, child = 0):
    def SetAttrValue(self, name, value):
    def SetWellKnownGeogCS(self, name):
    def SetProjCS(self, name = "unnamed" ):
    def IsSameGeogCS(self, other):
    def IsSame(self, other):
    def SetLinearUnits(self, units_name, to_meters ):
    def SetUTM(self, zone, is_north = 1):
```

```
class CoordinateTransformation:
    def __init__(self,source,target):
    def TransformPoint(self, x, y, z = 0):
    def TransformPoints(self, points):
```

6.7 Internal Implementation

The **OGRCoordinateTransformation** (p. 81) service is implemented on top of the PROJ.4 library originally written by Gerald Evenden of the USGS.

Chapter 7

Deprecated List

Member OGR_G_GetArea (p. **440**) **(OGRGeometryH)**

Member OGR_G_GetBoundary (p. **441**) **(OGRGeometryH)**

Member OGR_G_SymmetricDifference (p. **453**) **(OGRGeometryH, OGRGeometryH)**

Member OGRGeometry::getBoundary (p. **135**) **() const**

Member OGRGeometry::SymmetricDifference (p. **143**) **(const OGRGeometry** (p. **126**) $*$**) const**

Member OGRLayer::GetInfo (p. **169**) **(const char** $*$**)**

Member OGRSpatialReference::$\sim$**OGRSpatialReference** (p. **236**) **()**

Chapter 8

Class Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_CPLHashSet . . . 51
_CPLList . . . 51
_CPLQuadTree . . . 52
_QuadTreeNode . . . 52
_sPolyExtended . . . 52
CachedConnection . . . 52
CachedDirList . . . 52
CachedFileProp . . . 52
CachedRegion . . . 52
CPLErrorContext . . . 53
CPLHTTPResult . . . 53
CPLKeywordParser . . . 54
CPLLocaleC . . . 54
CPLMimePart . . . 54
CPLMutexHolder . . . 55
CPLODBCDriverInstaller . . . 55
CPLODBCSession . . . 56
CPLODBCStatement . . . 57
CPLRectObj . . . 65
CPLSharedFileInfo . . . 65
CPLStdCallThreadInfo . . . 65
CPLString . . . 65
CPLStringList . . . 67
CPLXMLNode . . . 72
ctb . . . 73
curfile_info . . . 73
DefaultCSVFileNameTLS . . . 74
errHandler . . . 74
file_in_zip_read_info_s . . . 74
FindFileTLS . . . 74
GZipSnapshot . . . 74
linkedlist_data_s . . . 74
linkedlist_datablock_internal_s . . . 74
OGR_SRSNode . . . 75
ogr_style_param . . . 80
ogr_style_value . . . 81
OGRAttrIndex . . . 81
    OGRMIAttrIndex . . . 194

OGRCoordinateTransformation . . . 81
OGRProj4CT . . . 220
OGRDataSource . . . 85
OGREnvelope . . . 93
OGREnvelope3D . . . 94
OGRFeature . . . 94
OGRFeatureDefn . . . 109
OGRFeatureQuery . . . 115
OGRField . . . 116
OGRFieldDefn . . . 116
OGRGeometry . . . 126
OGRCurve . . . 83
OGRLineString . . . 181
OGRLinearRing . . . 178
OGRGeometryCollection . . . 146
OGRMultiLineString . . . 195
OGRMultiPoint . . . 198
OGRMultiPolygon . . . 200
OGRPoint . . . 203
OGRSurface . . . 293
OGRPolygon . . . 211
OGRGeometryFactory . . . 156
OGRLayer . . . 162
OGRGenSQLResultsLayer . . . 121
OGRLayerAttrIndex . . . 177
OGRMILayerAttrIndex . . . 194
OGRProj4Datum . . . 222
OGRProj4PM . . . 222
OGRRawPoint . . . 222
OGRSFDriver . . . 222
OGRSFDriverRegistrar . . . 226
OGRSpatialReference . . . 230
OGRStyleMgr . . . 284
OGRStyleTable . . . 289
OGRStyleTool . . . 292
OGRStyleBrush . . . 283
OGRStyleLabel . . . 284
OGRStylePen . . . 288
OGRStyleSymbol . . . 289
OZIDatums . . . 294
ParseContext . . . 294
PCIDatums . . . 294
projUV . . . 294
SFRegion . . . 295
StackContext . . . 295
swq_col_def . . . 295
swq_expr_node . . . 295
swq_field_list . . . 295
swq_join_def . . . 295
swq_op_registrar . . . 295
swq_operation . . . 296
swq_order_def . . . 296
swq_parse_context . . . 296
swq_select . . . 296
swq_summary . . . 296
swq_table_def . . . 296

tm_unz_s . . . 296
tm_zip_s . . . 296
unz_file_info_internal_s . . . 297
unz_file_info_s . . . 297
unz_file_pos_s . . . 297
unz_global_info_s . . . 297
unz_s . . . 297
VSIArchiveContent . . . 297
VSIArchiveEntry . . . 297
VSIArchiveEntryFileOffset . . . 298
VSITarEntryFileOffset . . . 305
VSIZipEntryFileOffset . . . 307
VSIArchiveReader . . . 298
VSITarReader . . . 306
VSIZipReader . . . 308
VSICacheChunk . . . 299
VSIFileManager . . . 300
VSIFilesystemHandler . . . 300
VSIArchiveFilesystemHandler . . . 298
VSITarFilesystemHandler . . . 305
VSIZipFilesystemHandler . . . 308
VSICurlFilesystemHandler . . . 299
VSIGZipFilesystemHandler . . . 301
VSIMemFilesystemHandler . . . 302
VSISparseFileFilesystemHandler . . . 302
VSIStdinFilesystemHandler . . . 303
VSIStdoutFilesystemHandler . . . 303
VSIStdoutRedirectFilesystemHandler . . . 304
VSISubFileFilesystemHandler . . . 305
VSIUnixStdioFilesystemHandler . . . 306
VSIMemFile . . . 302
VSIVirtualHandle . . . 307
VSIBufferedReaderHandle . . . 299
VSICachedFile . . . 299
VSICurlHandle . . . 300
VSIGZipHandle . . . 301
VSIGZipWriteHandle . . . 301
VSIMemHandle . . . 302
VSISparseFileHandle . . . 303
VSIStdinHandle . . . 303
VSIStdoutHandle . . . 304
VSIStdoutRedirectHandle . . . 304
VSISubFileHandle . . . 305
VSIUnixStdioHandle . . . 306
VSIZipWriteHandle . . . 308
WriteFuncStruct . . . 309
yyalloc . . . 309
zip_fileinfo . . . 309
zip_internal . . . 309
zlib_filefunc_def_s . . . 309

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_CPLHashSet . . . 51
_CPLList . . . 51
_CPLQuadTree . . . 52
_QuadTreeNode . . . 52
_sPolyExtended . . . 52
CachedConnection . . . 52
CachedDirList . . . 52
CachedFileProp . . . 52
CachedRegion . . . 52
CPLErrorContext . . . 53
CPLHTTPResult . . . 53
CPLKeywordParser . . . 54
CPLLocaleC . . . 54
CPLMimePart . . . 54
CPLMutexHolder . . . 55
CPLODBCDriverInstaller . . . 55
CPLODBCSession . . . 56
CPLODBCStatement . . . 57
CPLRectObj . . . 65
CPLSharedFileInfo . . . 65
CPLStdCallThreadInfo . . . 65
CPLString
Convenient string class based on std::string . . . 65
CPLStringList
String list class designed around our use of C "char∗∗" string lists . . . 67
CPLXMLNode . . . 72
ctb . . . 73
curfile_info . . . 73
DefaultCSVFileNameTLS . . . 74
errHandler . . . 74
file_in_zip_read_info_s . . . 74
FindFileTLS . . . 74
GZipSnapshot . . . 74
linkedlist_data_s . . . 74
linkedlist_datablock_internal_s . . . 74
OGR_SRSNode . . . 75
ogr_style_param . . . 80
ogr_style_value . . . 81

OGRAttrIndex . . . 81
OGRCoordinateTransformation . . . 81
OGRCurve . . . 83
OGRDataSource . . . 85
OGREnvelope . . . 93
OGREnvelope3D . . . 94
OGRFeature . . . 94
OGRFeatureDefn . . . 109
OGRFeatureQuery . . . 115
OGRField . . . 116
OGRFieldDefn . . . 116
OGRGenSQLResultsLayer . . . 121
OGRGeometry . . . 126
OGRGeometryCollection . . . 146
OGRGeometryFactory . . . 156
OGRLayer . . . 162
OGRLayerAttrIndex . . . 177
OGRLinearRing . . . 178
OGRLineString . . . 181
OGRMIAttrIndex . . . 194
OGRMILayerAttrIndex . . . 194
OGRMultiLineString . . . 195
OGRMultiPoint . . . 198
OGRMultiPolygon . . . 200
OGRPoint . . . 203
OGRPolygon . . . 211
OGRProj4CT . . . 220
OGRProj4Datum . . . 222
OGRProj4PM . . . 222
OGRRawPoint . . . 222
OGRSFDriver . . . 222
OGRSFDriverRegistrar . . . 226
OGRSpatialReference . . . 230
OGRStyleBrush . . . 283
OGRStyleLabel . . . 284
OGRStyleMgr . . . 284
OGRStylePen . . . 288
OGRStyleSymbol . . . 289
OGRStyleTable . . . 289
OGRStyleTool . . . 292
OGRSurface . . . 293
OZIDatums . . . 294
ParseContext . . . 294
PCIDatums . . . 294
projUV . . . 294
SFRegion . . . 295
StackContext . . . 295
swq_col_def . . . 295
swq_expr_node . . . 295
swq_field_list . . . 295
swq_join_def . . . 295
swq_op_registrar . . . 295
swq_operation . . . 296
swq_order_def . . . 296
swq_parse_context . . . 296
swq_select . . . 296
swq_summary . . . 296
swq_table_def . . . 296

tm_unz_s . . . 296
tm_zip_s . . . 296
unz_file_info_internal_s . . . 297
unz_file_info_s . . . 297
unz_file_pos_s . . . 297
unz_global_info_s . . . 297
unz_s . . . 297
VSIArchiveContent . . . 297
VSIArchiveEntry . . . 297
VSIArchiveEntryFileOffset . . . 298
VSIArchiveFilesystemHandler . . . 298
VSIArchiveReader . . . 298
VSIBufferedReaderHandle . . . 299
VSICacheChunk . . . 299
VSICachedFile . . . 299
VSICurlFilesystemHandler . . . 299
VSICurlHandle . . . 300
VSIFileManager . . . 300
VSIFilesystemHandler . . . 300
VSIGZipFilesystemHandler . . . 301
VSIGZipHandle . . . 301
VSIGZipWriteHandle . . . 301
VSIMemFile . . . 302
VSIMemFilesystemHandler . . . 302
VSIMemHandle . . . 302
VSISparseFileFilesystemHandler . . . 302
VSISparseFileHandle . . . 303
VSIStdinFilesystemHandler . . . 303
VSIStdinHandle . . . 303
VSIStdoutFilesystemHandler . . . 303
VSIStdoutHandle . . . 304
VSIStdoutRedirectFilesystemHandler . . . 304
VSIStdoutRedirectHandle . . . 304
VSISubFileFilesystemHandler . . . 305
VSISubFileHandle . . . 305
VSITarEntryFileOffset . . . 305
VSITarFilesystemHandler . . . 305
VSITarReader . . . 306
VSIUnixStdioFilesystemHandler . . . 306
VSIUnixStdioHandle . . . 306
VSIVirtualHandle . . . 307
VSIZipEntryFileOffset . . . 307
VSIZipFilesystemHandler . . . 308
VSIZipReader . . . 308
VSIZipWriteHandle . . . 308
WriteFuncStruct . . . 309
yyalloc . . . 309
zip_fileinfo . . . 309
zip_internal . . . 309
zlib_filefunc_def_s . . . 309

Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

cpl_atomic_ops.h . . . . . **??**
cpl_config.h . . . . . **??**
cpl_config_extras.h . . . . . **??**
cpl_conv.h 311
cpl_csv.h . . . . . **??**
cpl_error.h 333
cpl_hash_set.h 337
cpl_http.h 341
cpl_list.h 343
cpl_minixml.h 346
cpl_minizip_ioapi.h . . . . . **??**
cpl_minizip_unzip.h . . . . . **??**
cpl_minizip_zip.h . . . . . **??**
cpl_multiproc.h . . . . . **??**
cpl_odbc.h 354
cpl_port.h 354
cpl_quad_tree.h 355
cpl_string.h 357
cpl_time.h . . . . . **??**
cpl_vsi.h 370
cpl_vsi_virtual.h . . . . . **??**
cpl_win32ce_api.h . . . . . **??**
cpl_wince.h . . . . . **??**
cplkeywordparser.h . . . . . **??**
gdal_csv.h . . . . . **??**
ogr_api.h 384
ogr_attrind.h . . . . . **??**
ogr_core.h 483
ogr_expat.h . . . . . **??**
ogr_feature.h 488
ogr_featurestyle.h 489
ogr_gensql.h . . . . . **??**
ogr_geometry.h 489
ogr_geos.h . . . . . **??**
ogr_p.h . . . . . **??**
ogr_spatialref.h 490
ogr_srs_api.h 490
ogr_srs_esri_names.h . . . . . **??**

ogrgeomediageometry.h . . . . . **??**
ogrpgeogeometry.h . . . . . **??**
ogrsf_frmts.h 518
swq.h . . . . . **??**

Chapter 11

Class Documentation

11.1 _CPLHashSet Struct Reference

The documentation for this struct was generated from the following file:

- cpl_hash_set.cpp

11.2 _CPLList Struct Reference

`#include <cpl_list.h>`

Public Attributes

- void * **pData**
- struct **_CPLList** * **psNext**

11.2.1 Detailed Description

List element structure.

11.2.2 Member Data Documentation

11.2.2.1 void* _CPLList::pData

Pointer to the data object. Should be allocated and frred by the caller.

Referenced by CPLHashSetDestroy(), CPLHashSetForeach(), CPLHashSetRemove(), CPLListAppend(), CPLListGetData(), and CPLListInsert().

11.2.2.2 struct _CPLList* _CPLList::psNext

Pointer to the next element in list. NULL, if current element is the last one

Referenced by CPLHashSetDestroy(), CPLHashSetForeach(), CPLHashSetRemove(), CPLListAppend(), CPLListCount(), CPLListDestroy(), CPLListGet(), CPLListGetLast(), CPLListGetNext(), CPLListInsert(), and CPLListRemove().

The documentation for this struct was generated from the following file:

- **cpl_list.h**

11.3 _CPLQuadTree Struct Reference

The documentation for this struct was generated from the following file:

- cpl_quad_tree.cpp

11.4 _QuadTreeNode Struct Reference

The documentation for this struct was generated from the following file:

- cpl_quad_tree.cpp

11.5 _sPolyExtended Struct Reference

The documentation for this struct was generated from the following file:

- ogrgeometryfactory.cpp

11.6 CachedConnection Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_curl.cpp

11.7 CachedDirList Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_curl.cpp

11.8 CachedFileProp Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_curl.cpp

11.9 CachedRegion Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_curl.cpp

11.10 CPLErrorContext Struct Reference

The documentation for this struct was generated from the following file:

- cpl_error.cpp

11.11 CPLHTTPResult Struct Reference

`#include <cpl_http.h>`

Public Attributes

- int **nStatus**
- char * **pszContentType**
- char * **pszErrBuf**
- int **nDataLen**
- GByte * **pabyData**
- char ** **papszHeaders**
- int **nMimePartCount**
- **CPLMimePart** * **pasMimePart**

11.11.1 Detailed Description

Describe the result of a **CPLHTTPFetch()** (p. 342) call

11.11.2 Member Data Documentation

11.11.2.1 int CPLHTTPResult::nDataLen

Length of the pabyData buffer

Referenced by CPLHTTPFetch(), CPLHTTPParseMultipartMime(), and OGRSpatialReference::importFromUrl().

11.11.2.2 int CPLHTTPResult::nMimePartCount

Number of parts in a multipart message

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

11.11.2.3 int CPLHTTPResult::nStatus

cURL error code : 0=success, non-zero if request failed

Referenced by CPLHTTPFetch(), and OGRSpatialReference::importFromUrl().

11.11.2.4 GByte* CPLHTTPResult::pabyData

Buffer with downloaded data

Referenced by CPLHTTPDestroyResult(), CPLHTTPParseMultipartMime(), and OGRSpatialReference::importFromUrl().

11.11.2.5 char** CPLHTTPResult::papszHeaders

Headers returned

Referenced by CPLHTTPDestroyResult(), and CPLHTTPFetch().

11.11.2.6 CPLMimePart* CPLHTTPResult::pasMimePart

Array of parts (resolved by **CPLHTTPParseMultipartMime()** (p. 342))

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

11.11.2.7 char* CPLHTTPResult::pszContentType

Content-Type of the response

Referenced by CPLHTTPDestroyResult(), CPLHTTPFetch(), and CPLHTTPParseMultipartMime().

11.11.2.8 char* CPLHTTPResult::pszErrBuf

Error message from curl, or NULL

Referenced by CPLHTTPDestroyResult(), CPLHTTPFetch(), and OGRSpatialReference::importFromUrl().

The documentation for this struct was generated from the following file:

- **cpl_http.h**

11.12 CPLKeywordParser Class Reference

The documentation for this class was generated from the following files:

- cplkeywordparser.h
- cplkeywordparser.cpp

11.13 CPLLocaleC Class Reference

The documentation for this class was generated from the following files:

- **cpl_conv.h**
- cpl_conv.cpp

11.14 CPLMimePart Struct Reference

`#include <cpl_http.h>`

Public Attributes

- char ** **papszHeaders**
- GByte * **pabyData**
- int **nDataLen**

11.14.1 Detailed Description

Describe a part of a multipart message

11.14.2 Member Data Documentation

11.14.2.1 int CPLMimePart::nDataLen

Buffer length

Referenced by CPLHTTPParseMultipartMime().

11.14.2.2 GByte* CPLMimePart::pabyData

Buffer with data of the part

Referenced by CPLHTTPParseMultipartMime().

11.14.2.3 char** CPLMimePart::papszHeaders

NULL terminated array of headers

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

The documentation for this struct was generated from the following file:

- **cpl_http.h**

11.15 CPLMutexHolder Class Reference

The documentation for this class was generated from the following files:

- cpl_multiproc.h
- cpl_multiproc.cpp

11.16 CPLODBCDriverInstaller Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- int **InstallDriver** (const char *pszDriver, const char *pszPathIn, WORD fRequest=ODBC_INSTALL_COMPLETE)
- int **RemoveDriver** (const char *pszDriverName, int fRemoveDSN=0)

11.16.1 Detailed Description

A class providing functions to install or remove ODBC driver.

11.16.2 Member Function Documentation

11.16.2.1 int CPLODBCDriverInstaller::InstallDriver (const char * *pszDriver,* const char * *pszPathIn,* WORD *fRequest =* ODBC_INSTALL_COMPLETE)

Installs ODBC driver or updates definition of already installed driver. Interanally, it calls ODBC's SQLInstallDriverEx function.

Parameters

pszDriver	- The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).
pszPathIn	- Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).
fRequest	- The fRequest argument must contain one of the following values: ODBC_INSTALL_COMPLETE - (default) complete the installation request ODBC_INSTALL_INQUIRY - inquire about where a driver can be installed

Returns

TRUE indicates success, FALSE if it fails.

11.16.2.2 int CPLODBCDriverInstaller::RemoveDriver (const char * *pszDriverName,* int *fRemoveDSN =* 0)

Removes or changes information about the driver from the Odbcinst.ini entry in the system information.

Parameters

pszDriverName	- The name of the driver as registered in the Odbcinst.ini key of the system information.
fRemoveDSN	- TRUE: Remove DSNs associated with the driver specified in lpszDriver. FALSE: Do not remove DSNs associated with the driver specified in lpszDriver.

Returns

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE. In order to obtain usage count value, call GetUsageCount().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**
- cpl_odbc.cpp

11.17 CPLODBCSession Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- int **EstablishSession** (const char *pszDSN, const char *pszUserid, const char *pszPassword)
- const char * **GetLastError** ()

11.17.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

11.17.2 Member Function Documentation

11.17.2.1 int CPLODBCSession::EstablishSession (const char * *pszDSN,* const char * *pszUserid,* const char * *pszPassword*)

Connect to database and logon.

Parameters

pszDSN	The name of the DSN being used to connect. This is not optional.
pszUserid	the userid to logon as, may be NULL if not not required, or provided by the DSN.
pszPassword	the password to logon with. May be NULL if not required or provided by the DSN.

Returns

TRUE on success or FALSE on failure. Call **GetLastError()** (p. 57) to get details on failure.

References GetLastError().

11.17.2.2 const char * CPLODBCSession::GetLastError ( )

Returns the last ODBC error message.

Returns

pointer to an internal buffer with the error message in it. Do not free or alter. Will be an empty (but not NULL) string if there is no pending error info.

Referenced by EstablishSession(), and CPLODBCStatement::Fetch().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**
- cpl_odbc.cpp

11.18 CPLODBCStatement Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- void **Clear** ()
- void **AppendEscaped** (const char *)
- void **Append** (const char *)
- void **Append** (int)
- void **Append** (double)
- int **Appendf** (const char *,...)
- int **ExecuteSQL** (const char *=0)
- int **Fetch** (int nOrientation=SQL_FETCH_NEXT, int nOffset=0)
- int **GetColCount** ()
- const char * **GetColName** (int)
- short **GetColType** (int)
- const char * **GetColTypeName** (int)

- short **GetColSize** (int)
- short **GetColPrecision** (int)
- short **GetColNullable** (int)
- int **GetColId** (const char *)
- const char * **GetColData** (int, const char *=0)
- const char * **GetColData** (const char *, const char *=0)
- int **GetColumns** (const char *pszTable, const char *pszCatalog=0, const char *pszSchema=0)
- int **GetPrimaryKeys** (const char *pszTable, const char *pszCatalog=0, const char *pszSchema=0)
- int **GetTables** (const char *pszCatalog=0, const char *pszSchema=0)
- void **DumpResult** (FILE *fp, int bShowSchema=0)

Static Public Member Functions

- static **CPLString GetTypeName** (int)
- static SQLSMALLINT **GetTypeMapping** (SQLSMALLINT)

11.18.1 Detailed Description

Abstraction for statement, and resultset.

Includes methods for executing an SQL statement, and for accessing the resultset from that statement. Also provides for executing other ODBC requests that produce results sets such as SQLColumns() and SQLTables() requests.

11.18.2 Member Function Documentation

11.18.2.1 void CPLODBCStatement::Append (const char * *pszText*)

Append text to internal command.

The passed text is appended to the internal SQL command text.

Parameters

pszText	text to append.

Referenced by Append(), AppendEscaped(), Appendf(), and ExecuteSQL().

11.18.2.2 void CPLODBCStatement::Append (int *nValue*)

Append to internal command.

The passed value is formatted and appended to the internal SQL command text.

Parameters

nValue	value to append to the command.

References Append().

11.18.2.3 void CPLODBCStatement::Append (double *dfValue*)

Append to internal command.

The passed value is formatted and appended to the internal SQL command text.

Parameters

dfValue	value to append to the command.

References Append().

11.18.2.4 void CPLODBCStatement::AppendEscaped (const char * *pszText*)

Append text to internal command.

The passed text is appended to the internal SQL command text after escaping any special characters so it can be used as a character string in an SQL statement.

Parameters

pszText	text to append.

References Append().

11.18.2.5 int CPLODBCStatement::Appendf (const char * *pszFormat, ...*)

Append to internal command.

The passed format is used to format other arguments and the result is appended to the internal command text. Long results may not be formatted properly, and should be appended with the direct **Append()** (p. 58) methods.

Parameters

pszFormat	printf() style format string.

Returns

FALSE if formatting fails dueto result being too large.

References Append().

11.18.2.6 void CPLODBCStatement::Clear ( )

Clear internal command text and result set definitions.

Referenced by ExecuteSQL().

11.18.2.7 void CPLODBCStatement::DumpResult (FILE * *fp,* int *bShowSchema* = 0)

Dump resultset to file.

The contents of the current resultset are dumped in a simply formatted form to the provided file. If requested, the schema definition will be written first.

Parameters

fp	the file to write to. stdout or stderr are acceptable.
bShowSchema	TRUE to force writing schema information for the rowset before the rowset data itself. Default is FALSE.

References Fetch(), GetColCount(), GetColData(), GetColName(), GetColNullable(), GetColPrecision(), GetColSize(), GetColType(), and GetTypeName().

11.18.2.8 int CPLODBCStatement::ExecuteSQL (const char * *pszStatement* = 0)

Execute an SQL statement.

This method will execute the passed (or stored) SQL statement, and initialize information about the resultset if there is one. If a NULL statement is passed, the internal stored statement that has been previously set via **Append()** (p. 58) or **Appendf()** (p. 59) calls will be used.

Parameters

pszStatement	the SQL statement to execute, or NULL if the internally saved one should be used.

Returns

TRUE on success or FALSE if there is an error. Error details can be fetched with OGRODBCSession::GetLastError().

References Append(), and Clear().

11.18.2.9 int CPLODBCStatement::Fetch (int *nOrientation* = SQL_FETCH_NEXT, int *nOffset* = 0)

Fetch a new record.

Requests the next row in the current resultset using the SQLFetchScroll() call. Note that many ODBC drivers only support the default forward fetching one record at a time. Only SQL_FETCH_NEXT (the default) should be considered reliable on all drivers.

Currently it isn't clear how to determine whether an error or a normal out of data condition has occured if **Fetch()** (p. 60) fails.

Parameters

nOrientation	One of SQL_FETCH_NEXT, SQL_FETCH_LAST, SQL_FETCH_PRIOR, SQL_FETCH_ABSOLUTE, or SQL_FETCH_RELATIVE (default is SQL_FETCH_NEXT).
nOffset	the offset (number of records), ignored for some orientations.

Returns

TRUE if a new row is successfully fetched, or FALSE if not.

References CPLODBCSession::GetLastError(), and GetTypeMapping().

Referenced by DumpResult().

11.18.2.10 int CPLODBCStatement::GetColCount ( )

Fetch the resultset column count.

Returns

the column count, or zero if there is no resultset.

Referenced by DumpResult().

11.18.2.11 const char * CPLODBCStatement::GetColData (int *iCol*, const char * *pszDefault* = 0)

Fetch column data.

Fetches the data contents of the requested column for the currently loaded row. The result is returned as a string regardless of the column type. NULL is returned if an illegal column is given, or if the actual column is "NULL".

Parameters

iCol	the zero based column to fetch.
pszDefault	the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns

pointer to internal column data or NULL on failure.

Referenced by DumpResult(), and GetColData().

11.18.2.12 const char * CPLODBCStatement::GetColData (const char * *pszColName*, const char * *pszDefault* = 0)

Fetch column data.

Fetches the data contents of the requested column for the currently loaded row. The result is returned as a string regardless of the column type. NULL is returned if an illegal column is given, or if the actual column is "NULL".

Parameters

pszColName	the name of the column requested.
pszDefault	the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns

pointer to internal column data or NULL on failure.

References GetColData(), and GetColId().

11.18.2.13 int CPLODBCStatement::GetColId (const char * *pszColName*)

Fetch column index.

Gets the column index corresponding with the passed name. The name comparisons are case insensitive.

Parameters

pszColName	the name to search for.

Returns

the column index, or -1 if not found.

Referenced by GetColData().

11.18.2.14 const char * CPLODBCStatement::GetColName (int *iCol*)

Fetch a column name.

Parameters

iCol	the zero based column index.

Returns

NULL on failure (out of bounds column), or a pointer to an internal copy of the column name.

Referenced by DumpResult().

11.18.2.15 short CPLODBCStatement::GetColNullable (int *iCol*)

Fetch the column nullability.

Parameters

iCol	the zero based column index.

Returns

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

11.18.2.16 short CPLODBCStatement::GetColPrecision (int *iCol*)

Fetch the column precision.

Parameters

iCol	the zero based column index.

Returns

column precision, may be zero or the same as column size for columns to which it does not apply.

Referenced by DumpResult().

11.18.2.17 short CPLODBCStatement::GetColSize (int *iCol*)

Fetch the column width.

Parameters

iCol	the zero based column index.

Returns

column width, zero for unknown width columns.

Referenced by DumpResult().

11.18.2.18 short CPLODBCStatement::GetColType (int *iCol*)

Fetch a column data type.

The return type code is a an ODBC SQL_ code, one of SQL_UNKNOWN_TYPE, SQL_CHAR, SQL_NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE, SQL_DATETIME, SQL_VARCHAR, SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMPT.

Parameters

iCol	the zero based column index.

Returns

type code or -1 if the column is illegal.

Referenced by DumpResult().

11.18.2.19 const char * CPLODBCStatement::GetColTypeName (int *iCol*)

Fetch a column data type name.

Returns data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINAR", or "CHAR ( ) FOR BIT DATA".

Parameters

iCol	the zero based column index.

Returns

NULL on failure (out of bounds column), or a pointer to an internal copy of the column dat type name.

11.18.2.20 int CPLODBCStatement::GetColumns (const char * *pszTable*, const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch column definitions for a table.

The SQLColumn() method is used to fetch the definitions for the columns of a table (or other queriable object such as a view). The column definitions are digested and used to populate the **CPLODBCStatement** (p. 57) column definitions essentially as if a "SELECT * FROM tablename" had been done; however, no resultset will be available.

Parameters

pszTable	the name of the table to query information on. This should not be empty.
pszCatalog	the catalog to find the table in, use NULL (the default) if no catalog is available.
pszSchema	the schema to find the table in, use NULL (the default) if no schema is available.

Returns

TRUE on success or FALSE on failure.

11.18.2.21 int CPLODBCStatement::GetPrimaryKeys (const char * *pszTable*, const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch primary keys for a table.

The SQLPrimaryKeys() function is used to fetch a list of fields forming the primary key. The result is returned as a result set matching the SQLPrimaryKeys() function result set. The 4th column in the result set is the column name of the key, and if the result set contains only one record then that single field will be the complete primary key.

Parameters

pszTable	the name of the table to query information on. This should not be empty.
pszCatalog	the catalog to find the table in, use NULL (the default) if no catalog is available.
pszSchema	the schema to find the table in, use NULL (the default) if no schema is available.

Returns

TRUE on success or FALSE on failure.

11.18.2.22 int CPLODBCStatement::GetTables (const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch tables in database.

The SQLTables() function is used to fetch a list tables in the database. The result is returned as a result set matching the SQLTables() function result set. The 3rd column in the result set is the table name. Only tables of type "TABLE" are returned.

Parameters

pszCatalog	the catalog to find the table in, use NULL (the default) if no catalog is available.
pszSchema	the schema to find the table in, use NULL (the default) if no schema is available.

Returns

TRUE on success or FALSE on failure.

11.18.2.23 SQLSMALLINT CPLODBCStatement::GetTypeMapping (SQLSMALLINT *nTypeCode*) [static]

Get appropriate C data type for SQL column type.

Returns a C data type code, corresponding to the indicated SQL data type code (as returned from **CPLODBCStatement::GetColType()** (p. 62)).

Parameters

nTypeCode	the SQL_ code, such as SQL_CHAR.

Returns

data type code. The valid code is always returned. If SQL code is not recognised, SQL_C_BINARY will be returned.

Referenced by Fetch().

11.18.2.24 CPLString CPLODBCStatement::GetTypeName (int *nTypeCode*) [static]

Get name for SQL column type.

Returns a string name for the indicated type code (as returned from **CPLODBCStatement::GetColType()** (p. 62)).

Parameters

nTypeCode	the SQL_ code, such as SQL_CHAR.

Returns

internal string, "UNKNOWN" if code not recognised.

Referenced by DumpResult().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**

- cpl_odbc.cpp

11.19 CPLRectObj Struct Reference

The documentation for this struct was generated from the following file:

- **cpl_quad_tree.h**

11.20 CPLSharedFileInfo Struct Reference

The documentation for this struct was generated from the following file:

- **cpl_conv.h**

11.21 CPLStdCallThreadInfo Struct Reference

The documentation for this struct was generated from the following file:

- cpl_multiproc.cpp

11.22 CPLString Class Reference

Convenient string class based on std::string.

`#include <cpl_string.h>`

Public Member Functions

- **CPLString** & **FormatC** (double dfValue, const char *pszFormat=0)
- **CPLString** & **Trim** ()
- size_t **ifind** (const std::string &str, size_t pos=0) const
- size_t **ifind** (const char *s, size_t pos=0) const
- **CPLString** & **toupper** (void)
- **CPLString** & **tolower** (void)

11.22.1 Detailed Description

Convenient string class based on std::string.

11.22.2 Member Function Documentation

11.22.2.1 CPLString & CPLString::FormatC (double *dfValue,* const char * *pszFormat* = `0`)

Format double in C locale.

The passed value is formatted using the C locale (period as decimal seperator) and appended to the target **CPLString** (p. 65).

Parameters

dfValue	the value to format.
pszFormat	the sprintf() style format to use or omit for default. Note that this format string should only include one substitution argument and it must be for a double (f or g).

Returns

a reference to the **CPLString** (p. 65).

11.22.2.2 size_t CPLString::ifind (const std::string & *str*, size_t *pos* = 0) const

Case insensitive find() alternative.

Parameters

str	substring to find.
pos	offset in the string at which the search starts.

Returns

the position of substring in the string or std::string::npos if not found.

Since

GDAL 1.9.0

Referenced by CPLURLAddKVP(), and CPLURLGetValue().

11.22.2.3 size_t CPLString::ifind (const char * *s*, size_t *nPos* = 0) const

Case insensitive find() alternative.

Parameters

s	substring to find.
nPos	offset in the string at which the search starts.

Returns

the position of the substring in the string or std::string::npos if not found.

Since

GDAL 1.9.0

References tolower().

11.22.2.4 CPLString & CPLString::tolower (void)

Convert to lower case in place.

Referenced by ifind().

11.22.2.5 CPLString & CPLString::toupper (void)

Convert to upper case in place.

11.22.2.6 CPLString & CPLString::Trim ( )

Trim white space.

Trims white space off the let and right of the string. White space is any of a space, a tab, a newline (’

’) or a carriage control (”).

Returns

a reference to the **CPLString** (p. 65).

The documentation for this class was generated from the following files:

- **cpl_string.h**
- cplstring.cpp

11.23 CPLStringList Class Reference

String list class designed around our use of C "char∗∗" string lists.

`#include <cpl_string.h>`

Public Member Functions

- **CPLStringList** (char ∗∗papszList, int bTakeOwnership=1)
- **CPLStringList** (const **CPLStringList** &oOther)

  Copy constructor.
- **CPLStringList** & **Clear** ()
- int **Count** () const
- **CPLStringList** & **AddString** (const char ∗pszNewString)
- **CPLStringList** & **AddStringDirectly** (char ∗pszNewString)
- **CPLStringList** & **InsertString** (int nInsertAtLineNo, const char ∗pszNewLine)

  Insert into the list at identified location.
- **CPLStringList** & **InsertStringDirectly** (int nInsertAtLineNo, char ∗pszNewLine)
- int **FindName** (const char ∗pszName) const
- int **FetchBoolean** (const char ∗pszKey, int bDefault) const
- const char ∗ **FetchNameValue** (const char ∗pszKey) const
- const char ∗ **FetchNameValueDef** (const char ∗pszKey, const char ∗pszDefault) const
- **CPLStringList** & **AddNameValue** (const char ∗pszKey, const char ∗pszValue)
- **CPLStringList** & **SetNameValue** (const char ∗pszKey, const char ∗pszValue)
- **CPLStringList** & **Assign** (char ∗∗papszList, int bTakeOwnership=1)
- char ∗ **operator[]** (int i)
- char ∗∗ **StealList** ()
- **CPLStringList** & **Sort** ()

11.23.1 Detailed Description

String list class designed around our use of C "char∗∗" string lists.

11.23.2 Constructor & Destructor Documentation

11.23.2.1 CPLStringList::CPLStringList (char ** papszListIn, int bTakeOwnership = 1)

CPLStringList (p. 67) constructor.

Parameters

papszListIn	the NULL terminated list of strings to consume.
bTakeOwnership	TRUE if the **CPLStringList** (p. 67) should take ownership of the list of strings which implies responsibility to free them.

References Assign().

11.23.3 Member Function Documentation

11.23.3.1 CPLStringList & CPLStringList::AddNameValue (const char * pszKey, const char * pszValue)

A a name=value entry to the list.

A key=value string is prepared and appended to the list. There is no check for other values for the same key in the list.

Parameters

pszKey	the key name to add.
pszValue	the key value to add.

References AddStringDirectly(), and InsertStringDirectly().

Referenced by SetNameValue().

11.23.3.2 CPLStringList & CPLStringList::AddString (const char * pszNewString)

Add a string to the list.

A copy of the passed in string is made and inserted in the list.

Parameters

pszNewString	the string to add to the list.

References AddStringDirectly().

Referenced by CSLTokenizeString2().

11.23.3.3 CPLStringList & CPLStringList::AddStringDirectly (char * pszNewString)

Add a string to the list.

This method is similar to **AddString()** (p. 68), but ownership of the pszNewString is transferred to the **CPLStringList** (p. 67) class.

Parameters

pszNewString	the string to add to the list.

References Count().

Referenced by AddNameValue(), and AddString().

11.23.3.4 CPLStringList & CPLStringList::Assign (char ** papszListIn, int bTakeOwnership = 1)

Assign a list of strings.

Parameters

papszListIn	the NULL terminated list of strings to consume.
bTakeOwnership	TRUE if the **CPLStringList** (p. 67) should take ownership of the list of strings which implies responsibility to free them.

Returns

a reference to the **CPLStringList** (p. 67) on which it was invoked.

References Clear().

Referenced by CPLStringList(), and CSLTokenizeString2().

11.23.3.5 CPLStringList & CPLStringList::Clear ( )

Clear the string list.

Referenced by Assign().

11.23.3.6 int CPLStringList::Count ( ) const

Returns

count of strings in the list, zero if empty.

Referenced by AddStringDirectly(), CSLTokenizeString2(), InsertStringDirectly(), operator[](), SetNameValue(), and Sort().

11.23.3.7 int CPLStringList::FetchBoolean (const char * pszKey, int bDefault) const

Check for boolean key value.

In a **CPLStringList** (p. 67) of "Name=Value" pairs, look to see if there is a key with the given name, and if it can be interpreted as being TRUE. If the key appears without any "=Value" portion it will be considered true. If the value is NO, FALSE or 0 it will be considered FALSE otherwise if the key appears in the list it will be considered TRUE. If the key doesn't appear at all, the indicated default value will be returned.

Parameters

pszKey	the key value to look for (case insensitive).
bDefault	the value to return if the key isn't found at all.

Returns

TRUE or FALSE

References FetchNameValue().

11.23.3.8 const char * CPLStringList::FetchNameValue (const char * pszName) const

Fetch value associated with this key name.

If this list sorted, a fast binary search is done, otherwise a linear scan is done. Name lookup is case insensitive.

Parameters

pszName	the key name to search for.

Returns

the corresponding value or NULL if not found. The returned string should not be modified and points into internal object state that may change on future calls.

References FindName().

Referenced by FetchBoolean(), and FetchNameValueDef().

11.23.3.9 const char * CPLStringList::FetchNameValueDef (const char * *pszName,* const char * *pszDefault*) const

Fetch value associated with this key name.

If this list sorted, a fast binary search is done, otherwise a linear scan is done. Name lookup is case insensitive.

Parameters

pszName	the key name to search for.
pszDefault	the default value returned if the named entry isn't found.

Returns

the corresponding value or the passed default if not found.

References FetchNameValue().

11.23.3.10 int CPLStringList::FindName (const char * *pszKey*) const

Get index of given name/value keyword.

Note that this search is for a line in the form name=value or name:value. Use FindString() or PartialFindString() for searches not based on name=value pairs.

Parameters

pszKey	the name to search for.

Returns

the string list index of this name, or -1 on failure.

Referenced by FetchNameValue(), and SetNameValue().

11.23.3.11 CPLStringList * CPLStringList::InsertString (int *nInsertAtLineNo,* const char * *pszNewLine*) [inline]

Insert into the list at identified location.

This method will insert a string into the list at the identified location. The insertion point must be within or at the end of the list. The following entries are pushed down to make space.

Parameters

nInsertAtLineNo	the line to insert at, zero to insert at front.
pszNewLine	to the line to insert. This string will be copied.

11.23.3.12 CPLStringList & CPLStringList::InsertStringDirectly (int *nInsertAtLineNo,* char * *pszNewLine*)

Insert into the list at identified location.

This method will insert a string into the list at the identified location. The insertion point must be within or at the end of the list. The following entries are pushed down to make space.

Parameters

nInsertAtLineNo	the line to insert at, zero to insert at front.
pszNewLine	to the line to insert, the ownership of this string will be taken over the by the object. It must have been allocated on the heap.

References Count().

Referenced by AddNameValue().

11.23.3.13 char * CPLStringList::operator[] (int *i*)

Fetch entry "i".

Fetches the requested item in the list. Note that the returned string remains owned by the **CPLStringList** (p. 67). If "i" is out of range NULL is returned.

Parameters

i	the index of the list item to return.

Returns

selected entry in the list.

References Count().

11.23.3.14 CPLStringList & CPLStringList::SetNameValue (const char * *pszKey,* const char * *pszValue*)

Set name=value entry in the list.

Similar to **AddNameValue()** (p. 68), except if there is already a value for the key in the list it is replaced instead of adding a new entry to the list. If pszValue is NULL any existing key entry is removed.

Parameters

pszKey	the key name to add.
pszValue	the key value to add.

References AddNameValue(), Count(), and FindName().

11.23.3.15 CPLStringList & CPLStringList::Sort ( )

Sort the entries in the list and mark list sorted.

Note that once put into "sorted" mode, the **CPLStringList** (p. 67) will attempt to keep things in sorted order through calls to **AddString()** (p. 68), **AddStringDirectly()** (p. 68), **AddNameValue()** (p. 68), **SetNameValue()** (p. 71). Complete list assignments (via **Assign()** (p. 69) and operator= will clear the sorting state. When in sorted order **FindName()** (p. 70), **FetchNameValue()** (p. 69) and **FetchNameValueDef()** (p. 70) will do a binary search to find the key, substantially improve lookup performance in large lists.

References Count().

11.23.3.16 char ** CPLStringList::StealList ( )

Seize ownership of underlying string array.

This method is simmilar to List(), except that the returned list is now owned by the caller and the **CPLStringList** (p. 67) is emptied.

Returns

the C style string list.

Referenced by CSLTokenizeString2().

The documentation for this class was generated from the following files:

- **cpl_string.h**
- cplstringlist.cpp

11.24 CPLXMLNode Struct Reference

`#include <cpl_minixml.h>`

Public Attributes

- **CPLXMLNodeType eType**

  Node type.
- char * **pszValue**

  Node value.
- struct **CPLXMLNode** * **psNext**

  Next sibling.
- struct **CPLXMLNode** * **psChild**

  Child node.

11.24.1 Detailed Description

Document node structure.

This C structure is used to hold a single text fragment representing a component of the document when parsed. It should be allocated with the appropriate CPL function, and freed with **CPLDestroyXMLNode()** (p. 350). The structure contents should not normally be altered by application code, but may be freely examined by application code.

Using the psChild and psNext pointers, a heirarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. 72) structures.

11.24.2 Member Data Documentation

11.24.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

One of CXT_Element, CXT_Text, CXT_Attribute, CXT_Comment, or CXT_Literal.

Referenced by CPLAddXMLChild(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLSearchXMLNode(), CPLSetXMLValue(), and CPLStripXMLNamespace().

11.24.2.2 struct CPLXMLNode* CPLXMLNode::psChild

Child node.

Pointer to first child node, if any. Only CXT_Element and CXT_Attribute nodes should have children. For CXT_-Attribute it should be a single CXT_Text value node, while CXT_Element can have any kind of child. The full list of children for a node are identified by walking the psNext's starting with the psChild node.

Referenced by CPLAddXMLChild(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPL-GetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), CPLSetXMLValue(), and C-PLStripXMLNamespace().

11.24.2.3 struct CPLXMLNode* CPLXMLNode::psNext

Next sibling.

Pointer to next sibling, that is the next node appearing after this one that has the same parent as this node. NULL if this node is the last child of the parent element.

Referenced by CPLAddXMLChild(), CPLAddXMLSibling(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPL-DestroyXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), C-PLSerializeXMLTree(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::importFromX-ML().

11.24.2.4 char* CPLXMLNode::pszValue

Node value.

For CXT_Element this is the name of the element, without the angle brackets. Note there is a single CXT_Element even when the document contains a start and end element tag. The node represents the pair. All text or other elements between the start and end tag will appear as children nodes of this CXT_Element node.

For CXT_Attribute the pszValue is the attribute name. The value of the attribute will be a CXT_Text child.

For CXT_Text this is the text itself (value of an attribute, or a text fragment between an element start and end tags.

For CXT_Literal it is all the literal text. Currently this is just used for !DOCTYPE lines, and the value would be the entire line.

For CXT_Comment the value is all the literal text within the comment, but not including the comment start/end indicators ("<–" and "–>").

Referenced by CPLCloneXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLGetXMLNode(), CPL-GetXMLValue(), CPLParseXMLString(), CPLSearchXMLNode(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::importFromXML().

The documentation for this struct was generated from the following file:

- **cpl_minixml.h**

11.25 ctb Struct Reference

The documentation for this struct was generated from the following file:

- cpl_csv.cpp

11.26 curfile_info Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.cpp

11.27 DefaultCSVFileNameTLS Struct Reference

The documentation for this struct was generated from the following file:

- cpl_csv.cpp

11.28 errHandler Struct Reference

The documentation for this struct was generated from the following file:

- cpl_error.cpp

11.29 file_in_zip_read_info_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.cpp

11.30 FindFileTLS Struct Reference

The documentation for this struct was generated from the following file:

- cpl_findfile.cpp

11.31 GZipSnapshot Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_gzip.cpp

11.32 linkedlist_data_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.cpp

11.33 linkedlist_datablock_internal_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.cpp

11.34 OGR_SRSNode Class Reference

`#include <ogr_spatialref.h>`

Public Member Functions

- **OGR_SRSNode** (const char *=NULL)
- int **GetChildCount** () const
- **OGR_SRSNode** * **GetChild** (int)
- **OGR_SRSNode** * **GetNode** (const char *)
- void **InsertChild** (**OGR_SRSNode** *, int)
- void **AddChild** (**OGR_SRSNode** *)
- int **FindChild** (const char *) const
- void **DestroyChild** (int)
- void **StripNodes** (const char *)
- const char * **GetValue** () const
- void **SetValue** (const char *)
- void **MakeValueSafe** ()
- **OGR_SRSNode** * **Clone** () const
- OGRErr **importFromWkt** (char **)
- OGRErr **exportToWkt** (char **) const
- OGRErr **applyRemapper** (const char *pszNode, char **papszSrcValues, char **papszDstValues, int nStepSize=1, int bChildOfHit=FALSE)

11.34.1 Detailed Description

Objects of this class are used to represent value nodes in the parsed representation of the WKT SRS format. For instance UNIT["METER",1] would be rendered into three OGR_SRSNodes. The root node would have a value of UNIT, and two children, the first with a value of METER, and the second with a value of 1.

Normally application code just interacts with the **OGRSpatialReference** (p. 230) object, which uses the **OGR_SRSNode** (p. 75) to implement it's data structure; however, this class is user accessable for detailed access to components of an SRS definition.

11.34.2 Constructor & Destructor Documentation

11.34.2.1 OGR_SRSNode::OGR_SRSNode (const char * *pszValueIn* = NULL)

Constructor.

Parameters

pszValueIn	this optional parameter can be used to initialize the value of the node upon creation. If omitted the node will be created with a value of "". Newly created OGR_SRSNodes have no children.

Referenced by Clone(), and importFromWkt().

11.34.3 Member Function Documentation

11.34.3.1 void OGR_SRSNode::AddChild (OGR_SRSNode * *poNew*)

Add passed node as a child of target node.

Note that ownership of the passed node is assumed by the node on which the method is invoked ... use the **Clone()** (p. 76) method if the original is to be preserved. New children are always added at the end of the list.

Parameters

poNew	the node to add as a child.

References InsertChild().

Referenced by Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), importFromWkt(), OGRSpatialReference::importFromWkt(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetCompoundCS(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and OGRSpatialReference::SetVertCS().

11.34.3.2 OGRErr OGR_SRSNode::applyRemapper (const char * *pszNode*, char ** *papszSrcValues*, char ** *papszDstValues*, int *nStepSize* = 1, int *bChildOfHit* = FALSE)

Remap node values matching list.

Remap the value of this node or any of it's children if it matches one of the values in the source list to the corresponding value from the destination list. If the pszNode value is set, only do so if the parent node matches that value. Even if a replacement occurs, searching continues.

Parameters

pszNode	Restrict remapping to children of this type of node (eg. "PROJECTION")
papszSrcValues	a NULL terminated array of source string. If the node value matches one of these (case insensitive) then replacement occurs.
papszDstValues	an array of destination strings. On a match, the one corresponding to a source value will be used to replace a node.
nStepSize	increment when stepping through source and destination arrays, allowing source and destination arrays to be one interleaved array for instances. Defaults to 1.
bChildOfHit	Only TRUE if we the current node is the child of a match, and so needs to be set. Application code would normally pass FALSE for this argument.

Returns

returns OGRERR_NONE unless something bad happens. There is no indication returned about whether any replacement occured.

References applyRemapper(), GetChild(), GetChildCount(), and SetValue().

Referenced by applyRemapper(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::morphToESRI().

11.34.3.3 OGR_SRSNode * OGR_SRSNode::Clone ( ) const

Make a duplicate of this node, and it's children.

Returns

a new node tree, which becomes the responsiblity of the caller.

References AddChild(), and OGR_SRSNode().

Referenced by OGRSpatialReference::Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetCompoundCS(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), and OGRSpatialReference::StripVertical().

11.34.3.4 void OGR_SRSNode::DestroyChild (int *iChild*)

Remove a child node, and it's subtree.

Note that removing a child node will result in children after it being renumbered down one.

Parameters

iChild	the index of the child.

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

11.34.3.5 OGRErr OGR_SRSNode::exportToWkt (char ** *ppszResult*) const

Convert this tree of nodes into WKT format.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

Parameters

ppszResult	the resulting string is returned in this pointer.

Returns

currently OGRERR_NONE is always returned, but the future it is possible error conditions will develop.

References exportToWkt().

Referenced by exportToWkt(), and OGRSpatialReference::exportToWkt().

11.34.3.6 int OGR_SRSNode::FindChild (const char * *pszValue*) const

Find the index of the child matching the given string.

Note that the node value must match pszValue with the exception of case. The comparison is case insensitive.

Parameters

pszValue	the node value being searched for.

Returns

the child index, or -1 on failure.

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

11.34.3.7 OGR_SRSNode * OGR_SRSNode::GetChild (int *iChild*)

Fetch requested child.

Parameters

iChild	the index of the child to fetch, from 0 to **GetChildCount()** (p. 78) - 1.

Returns

a pointer to the child **OGR_SRSNode** (p. 75), or NULL if there is no such child.

Referenced by applyRemapper(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), StripNodes(), and OGRSpatialReference::StripVertical().

11.34.3.8 int OGR_SRSNode::GetChildCount ( ) const [inline]

Get number of children nodes.

Returns

0 for leaf nodes, or the number of children nodes.

Referenced by applyRemapper(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

11.34.3.9 OGR_SRSNode * OGR_SRSNode::GetNode (const char * *pszName*)

Find named node in tree.

This method does a pre-order traversal of the node tree searching for a node with this exact value (case insensitive), and returns it. Leaf nodes are not considered, under the assumption that they are just attribute value nodes.

If a node appears more than once in the tree (such as UNIT for instance), the first encountered will be returned. Use **GetNode()** (p. 78) on a subtree to be more specific.

Parameters

pszName	the name of the node to search for.

Returns

a pointer to the node found, or NULL if none.

References GetNode().

Referenced by OGRSpatialReference::exportToProj4(), OGRSpatialReference::GetAttrNode(), GetNode(), OGRSpatialReference::importFromProj4(), and OGRSpatialReference::SetGeocCS().

11.34.3.10 const char * OGR_SRSNode::GetValue ( ) const [inline]

Fetch value string for this node.

Returns

A non-NULL string is always returned. The returned pointer is to the internal value of this node, and should not be modified, or freed.

Referenced by OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::IsCompound(), OGRSpatialReference::IsGeocentric(), OGRSpatialReference::IsGeographic(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatialReference::IsVertical(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetVertCS(), and OGRSpatialReference::StripCTParms().

11.34.3.11 OGRErr OGR_SRSNode::importFromWkt (char ** *ppszInput*)

Import from WKT string.

This method will wipe the existing children and value of this node, and reassign them based on the contents of the passed WKT string. Only as much of the input string as needed to construct this node, and it's children is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

Parameters

ppszInput	Pointer to pointer to input. The pointer is updated to point to remaining unused input text.

Returns

OGRERR_NONE if import succeeds, or OGRERR_CORRUPT_DATA if it fails for any reason.

References AddChild(), importFromWkt(), OGR_SRSNode(), and SetValue().

Referenced by importFromWkt(), and OGRSpatialReference::importFromWkt().

11.34.3.12 void OGR_SRSNode::InsertChild (OGR_SRSNode * *poNew,* int *iChild*)

Insert the passed node as a child of target node, at the indicated position.

Note that ownership of the passed node is assumed by the node on which the method is invoked ... use the **Clone()** (p. 76) method if the original is to be preserved. All existing children at location iChild and beyond are push down one space to make space for the new child.

Parameters

poNew	the node to add as a child.
iChild	position to insert, use 0 to insert at the beginning.

Referenced by AddChild(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), and OGRSpatialReference::SetTOWGS84().

11.34.3.13 void OGR_SRSNode::MakeValueSafe ( )

Massage value string, stripping special characters so it will be a database safe string.

The operation is also applies to all subnodes of the current node.

References GetChild(), GetChildCount(), and MakeValueSafe().

Referenced by MakeValueSafe().

11.34.3.14 void OGR_SRSNode::SetValue (const char * *pszNewValue*)

Set the node value.

Parameters

pszNewValue	the new value to assign to this node. The passed string is duplicated and remains the responsibility of the caller.

Referenced by applyRemapper(), importFromWkt(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), and OGRSpatialReference::SetTargetLinearUnits().

11.34.3.15 void OGR_SRSNode::StripNodes (const char * *pszName*)

Strip child nodes matching name.

Removes any decendent nodes of this node that match the given name. Of course children of removed nodes are also discarded.

Parameters

pszName	the name for nodes that should be removed.

References DestroyChild(), FindChild(), GetChild(), GetChildCount(), and StripNodes().

Referenced by OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::importFromEPSG(), OGRSpatialReference::StripCTParms(), and StripNodes().

The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogr_srsnode.cpp

11.35 ogr_style_param Struct Reference

The documentation for this struct was generated from the following file:

- **ogr_featurestyle.h**

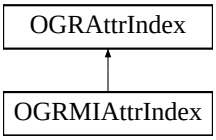
11.36 ogr_style_value Struct Reference

The documentation for this struct was generated from the following file:

- **ogr_featurestyle.h**

11.37 OGRAttrIndex Class Reference

Inheritance diagram for OGRAttrIndex:



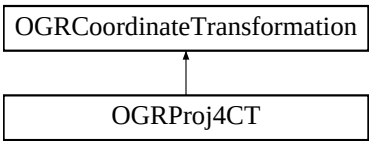
The documentation for this class was generated from the following files:

- ogr_attrind.h
- ogr_attrind.cpp

11.38 OGRCoordinateTransformation Class Reference

`#include <ogr_spatialref.h>`

Inheritance diagram for OGRCoordinateTransformation:



Public Member Functions

- virtual **OGRSpatialReference** ∗ **GetSourceCS** ()=0
- virtual **OGRSpatialReference** ∗ **GetTargetCS** ()=0
- virtual int **Transform** (int nCount, double ∗x, double ∗y, double ∗z=NULL)=0
- virtual int **TransformEx** (int nCount, double ∗x, double ∗y, double ∗z=NULL, int ∗pabSuccess=NULL)=0

Static Public Member Functions

- static void **DestroyCT** (**OGRCoordinateTransformation** ∗poCT)

  OGRCoordinateTransformation *(*p. *81) destructor.*

11.38.1 Detailed Description

Interface for transforming between coordinate systems.

Currently, the only implementation within OGR is **OGRProj4CT** (p. 220), which requires the PROJ.4 library to be available at run-time.

Also, see **OGRCreateCoordinateTransformation()** (p. 490) for creating transformations.

11.38.2 Member Function Documentation

11.38.2.1 void OGRCoordinateTransformation::DestroyCT (OGRCoordinateTransformation * *poCT*) [static]

OGRCoordinateTransformation (p. 81) destructor.

This function is the same as OGRCoordinateTransformation::~OGRCoordinateTransformation() and **OCTDestroyCoordinateTransformation()** (p. 496)

This static method will destroy a **OGRCoordinateTransformation** (p. 81). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

Parameters

poCT	the object to delete

Since

GDAL 1.7.0

11.38.2.2 virtual OGRSpatialReference* OGRCoordinateTransformation::GetSourceCS ( ) [pure virtual]

Fetch internal source coordinate system.

Implemented in **OGRProj4CT** (p. 221).

11.38.2.3 virtual OGRSpatialReference* OGRCoordinateTransformation::GetTargetCS ( ) [pure virtual]

Fetch internal target coordinate system.

Implemented in **OGRProj4CT** (p. 221).

Referenced by OGRPoint::transform(), OGRLineString::transform(), OGRPolygon::transform(), and OGRGeometryCollection::transform().

11.38.2.4 virtual int OGRCoordinateTransformation::Transform (int *nCount*, double * *x*, double * *y*, double * *z* = NULL) [pure virtual]

Transform points from source to destination space.

This method is the same as the C function OCTTransform().

The method **TransformEx()** (p. 83) allows extended success information to be captured indicating which points failed to transform.

Parameters

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.

Returns

TRUE on success, or FALSE if some or all points fail to transform.

Implemented in **OGRProj4CT** (p. 221).

Referenced by OGRPoint::transform().

11.38.2.5 virtual int OGRCoordinateTransformation::TransformEx (int *nCount*, double * *x*, double * *y*, double * *z* = NULL, int * *pabSuccess* = NULL) [pure virtual]

Transform points from source to destination space.

This method is the same as the C function OCTTransformEx().

Parameters

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.
pabSuccess	array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

Returns

TRUE if some or all points transform successfully, or FALSE if if none transform.

Implemented in **OGRProj4CT** (p. 221).

Referenced by OGRLineString::transform().

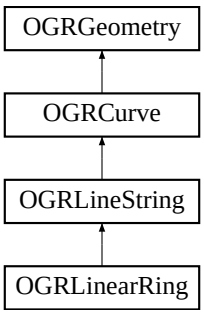
The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogrct.cpp

11.39 OGRCurve Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRCurve:



Public Member Functions

- virtual double **get_Length** () const =0

  Returns the length of the curve.

- virtual void **StartPoint** (**OGRPoint** *) const =0

  Return the curve start point.
- virtual void **EndPoint** (**OGRPoint** *) const =0

  Return the curve end point.
- virtual int **get_IsClosed** () const

  Return TRUE if curve is closed.
- virtual void **Value** (double, **OGRPoint** *) const =0

  Fetch point at given distance along curve.

11.39.1 Detailed Description

Abstract curve base class.

11.39.2 Member Function Documentation

11.39.2.1 void OGRCurve::EndPoint (OGRPoint * *poPoint*) const [pure virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get_EndPoint() method.

Parameters

poPoint	the point to be assigned the end location.

Implemented in **OGRLineString** (p. 184).

Referenced by get_IsClosed().

11.39.2.2 int OGRCurve::get_IsClosed ( ) const [virtual]

Return TRUE if curve is closed.

Tests if a curve is closed. A curve is closed if its start point is equal to its end point.

This method relates to the SFCOM ICurve::get_IsClosed() method.

Returns

TRUE if closed, else FALSE.

References EndPoint(), OGRPoint::getX(), OGRPoint::getY(), and StartPoint().

11.39.2.3 double OGRCurve::get_Length ( ) const [pure virtual]

Returns the length of the curve.

This method relates to the SFCOM ICurve::get_Length() method.

Returns

the length of the curve, zero if the curve hasn't been initialized.

Implemented in **OGRLineString** (p. 186).

11.39.2.4 void OGRCurve::StartPoint (OGRPoint * *poPoint*) const [pure virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get_StartPoint() method.

Parameters

poPoint	the point to be assigned the start location.

Implemented in **OGRLineString** (p. 193).

Referenced by get_IsClosed().

11.39.2.5 void OGRCurve::Value (double *dfDistance,* OGRPoint * *poPoint*) const [pure virtual]

Fetch point at given distance along curve.

This method relates to the SF COM ICurve::get_Value() method.

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. 84) for this curve.
poPoint	the point to be assigned the curve position.

Implemented in **OGRLineString** (p. 193).

The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrcurve.cpp

11.40 OGRDataSource Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * **GetName** ()=0

  Returns the name of the data source.
- virtual int **GetLayerCount** ()=0

  Get the number of layers in this data source.
- virtual **OGRLayer** * **GetLayer** (int)=0

  Fetch a layer by index.
- virtual **OGRLayer** * **GetLayerByName** (const char *)

  Fetch a layer by name.
- virtual OGRErr **DeleteLayer** (int)

  Delete the indicated layer from the datasource.
- virtual int **TestCapability** (const char *)=0

  Test if capability is available.
- virtual **OGRLayer** * **CreateLayer** (const char *pszName, **OGRSpatialReference** *poSpatialRef=NULL, **OGRwkbGeometryType** eGType=**wkbUnknown**, char **papszOptions=NULL)

  This method attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.

- virtual **OGRLayer** * **CopyLayer** (**OGRLayer** *poSrcLayer, const char *pszNewName, char **papsz-Options=NULL)

  Duplicate an existing layer.
- virtual **OGRStyleTable** * **GetStyleTable** ()

  Returns data source style table.
- virtual void **SetStyleTableDirectly** (**OGRStyleTable** *poStyleTable)

  Set data source style table.
- virtual void **SetStyleTable** (**OGRStyleTable** *poStyleTable)

  Set data source style table.
- virtual **OGRLayer** * **ExecuteSQL** (const char *pszStatement, **OGRGeometry** *poSpatialFilter, const char *pszDialect)

  Execute an SQL statement against the data store.
- virtual void **ReleaseResultSet** (**OGRLayer** *poResultsSet)

  *Release results of **ExecuteSQL()** (p. 89).*
- virtual OGRErr **SyncToDisk** ()

  Flush pending changes to disk.
- int **Reference** ()

  Increment datasource reference count.
- int **Dereference** ()

  Decrement datasource reference count.
- int **GetRefCount** () const

  Fetch reference count.
- int **GetSummaryRefCount** () const

  Fetch reference count of datasource and all owned layers.
- OGRErr **Release** ()

  Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.
- **OGRSFDriver** * **GetDriver** () const

  Returns the driver that the dataset was opened with.
- void **SetDriver** (**OGRSFDriver** *poDriver)

  Sets the driver that the dataset was created or opened with.

Static Public Member Functions

- static void **DestroyDataSource** (**OGRDataSource** *)

  Closes opened datasource and releases allocated resources.

Friends

- class **OGRSFDriverRegistrar**

11.40.1 Detailed Description

This class represents a data source. A data source potentially consists of many layers (**OGRLayer** (p. 162)). A data source normally consists of one, or a related set of files, though the name doesn't have to be a real item in the file system.

When an **OGRDataSource** (p. 85) is destroyed, all it's associated OGRLayers objects are also destroyed.

11.40.2 Member Function Documentation

11.40.2.1 OGRLayer * OGRDataSource::CopyLayer (OGRLayer * *poSrcLayer,* const char * *pszNewName,* char ** *papszOptions* = NULL) [virtual]

Duplicate an existing layer.

This method creates a new layer, duplicate the field definitions of the source layer and then duplicate each features of the source layer. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation. The source layer may come from another dataset.

This method is the same as the C function **OGR_DS_CopyLayer()** (p. 397).

Parameters

poSrcLayer	source layer.
pszNewName	the name of the layer to create.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

an handle to the layer, or NULL if an error occurs.

References OGRLayer::CreateFeature(), OGRFeature::CreateFeature(), OGRLayer::CreateField(), CreateLayer(), OGRFeature::DestroyFeature(), OGRFeature::GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetGeomType(), OGRLayer::GetLayerDefn(), OGRFeatureDefn::GetName(), OGRLayer::GetNextFeature(), OGRLayer::GetSpatialRef(), OGRLayer::ResetReading(), OGRFeature::SetFID(), OGRFeature::SetFrom(), OGRLayer::TestCapability(), and TestCapability().

Referenced by OGRSFDriver::CopyDataSource().

11.40.2.2 OGRLayer * OGRDataSource::CreateLayer (const char * *pszName,* OGRSpatialReference * *poSpatialRef* = NULL, OGRwkbGeometryType *eGType* = wkbUnknown, char ** *papszOptions* = NULL) [virtual]

This method attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

Parameters

pszName	the name for the new layer. This should ideally not match any existing layer on the datasource.
poSpatialRef	the coordinate system to use for the new layer, or NULL if no coordinate system is available.
eGType	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

NULL is returned on failure, or a new **OGRLayer** (p. 162) handle on success.

Example:

```
#include "ogrsf_frmts.h"
#include "cpl_string.h"

...

        OGRLayer *poLayer;
```

```
char     *papszOptions;

if( !poDS->TestCapability( ODsCCreateLayer ) )
{
    ...
}

papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
poLayer = poDS->CreateLayer( "NewLayer", NULL, wkbUnknown,
                             papszOptions );
CSLDestroy( papszOptions );

if( poLayer == NULL )
{
    ...
}
```

Referenced by CopyLayer().

11.40.2.3 OGRErr OGRDataSource::DeleteLayer (int *iLayer*) [virtual]

Delete the indicated layer from the datasource.

If this method is supported the ODsCDeleteLayer capability will test TRUE on the **OGRDataSource** (p. 85).

This method is the same as the C function **OGR_DS_DeleteLayer()** (p. 398).

Parameters

iLayer	the index of the layer to delete.

Returns

OGRERR_NONE on success, or OGRERR_UNSUPPORTED_OPERATION if deleting layers is not supported for this datasource.

11.40.2.4 int OGRDataSource::Dereference ( )

Decrement datasource reference count.

This method is the same as the C function OGR_DS_Dereference().

Returns

the reference count after decrementing.

Referenced by ExecuteSQL().

11.40.2.5 void OGRDataSource::DestroyDataSource (OGRDataSource * *poDS*) [static]

Closes opened datasource and releases allocated resources.

This static method will close and destroy a datasource. It is equivelent to calling delete on the object, but it ensures that the deallocation is properly executed within the GDAL libraries heap on platforms where this can matter (win32).

This method is the same as the C function **OGR_DS_Destroy()** (p. 399).

Parameters

poDS	pointer to allocated datasource object.

11.40.2.6 OGRLayer * OGRDataSource::ExecuteSQL (const char * *pszStatement*, OGRGeometry * *poSpatialFilter*, const char * *pszDialect*) [virtual]

Execute an SQL statement against the data store.

The result of an SQL query is either NULL for statements that are in error, or that have no results set, or an **OGRLayer** (p. 162) pointer representing a results set from the query. Note that this **OGRLayer** (p. 162) is in addition to the layers in the data store and must be destroyed with **OGRDataSource::ReleaseResultSet()** (p. 91) before the data source is closed (destroyed).

This method is the same as the C function **OGR_DS_ExecuteSQL()** (p. 399).

For more information on the SQL dialect supported internally by OGR review the OGR SQL document. Some drivers (ie. Oracle and PostGIS) pass the SQL directly through to the underlying RDBMS.

Parameters

pszStatement	the SQL statement to execute.
poSpatialFilter	geometry which represents a spatial filter. Can be NULL.
pszDialect	allows control of the statement dialect. If set to NULL, the OGR SQL engine will be used, except for RDBMS drivers that will use their dedicated SQL engine, unless OGRSQL is explicitely passed as the dialect.

Returns

an **OGRLayer** (p. 162) containing the results of the query. Deallocate with **ReleaseResultSet()** (p. 91).

References Dereference(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetLayerByName(), OGRLayer::GetLayerDefn(), OGRSFDriver::GetName(), OGRFieldDefn::GetNameRef(), OGRSFDriverRegistrar::GetRegistrar(), OGRFieldDefn::GetType(), OFTInteger, OFTReal, and OFTString.

11.40.2.7 OGRSFDriver * OGRDataSource::GetDriver ( ) const

Returns the driver that the dataset was opened with.

This method is the same as the C function **OGR_DS_GetDriver()** (p. 400).

Returns

NULL if driver info is not available, or pointer to a driver owned by the OGRSFDriverManager.

Referenced by OGRSFDriver::CopyDataSource(), OGR_Dr_CopyDataSource(), OGR_Dr_CreateDataSource(), OGR_Dr_Open(), and OGRSFDriverRegistrar::Open().

11.40.2.8 OGRLayer * OGRDataSource::GetLayer (int *iLayer*) [pure virtual]

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. 85) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayer()** (p. 400).

Parameters

iLayer	a layer number between 0 and **GetLayerCount()** (p. 90)-1.

Returns

the layer, or NULL if iLayer is out of range or an error occurs.

Referenced by OGRSFDriver::CopyDataSource(), GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

11.40.2.9 OGRLayer * OGRDataSource::GetLayerByName (const char * *pszLayerName*) [virtual]

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. 85) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayerByName()** (p. 400).

Parameters

pszLayerName	the layer name of the layer to fetch.

Returns

the layer, or NULL if Layer is not found or an error occurs.

References GetLayer(), GetLayerCount(), and OGRLayer::GetName().

Referenced by ExecuteSQL().

11.40.2.10 int OGRDataSource::GetLayerCount ( ) [pure virtual]

Get the number of layers in this data source.

This method is the same as the C function **OGR_DS_GetLayerCount()** (p. 401).

Returns

layer count.

Referenced by OGRSFDriver::CopyDataSource(), GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

11.40.2.11 const char * OGRDataSource::GetName ( ) [pure virtual]

Returns the name of the data source.

This string should be sufficient to open the data source if passed to the same **OGRSFDriver** (p. 222) that this data source was opened with, but it need not be exactly the same string that was used to open the data source. Normally this is a filename.

This method is the same as the C function **OGR_DS_GetName()** (p. 401).

Returns

pointer to an internal name string which should not be modified or freed by the caller.

11.40.2.12 int OGRDataSource::GetRefCount ( ) const

Fetch reference count.

This method is the same as the C function OGR_DS_GetRefCount().

Returns

the current reference count for the datasource object itself.

11.40.2.13 OGRStyleTable * OGRDataSource::GetStyleTable ( ) [virtual]

Returns data source style table.

This method is the same as the C function OGR_DS_GetStyleTable().

Returns

pointer to a style table which should not be modified or freed by the caller.

11.40.2.14 int OGRDataSource::GetSummaryRefCount ( ) const

Fetch reference count of datasource and all owned layers.

This method is the same as the C function OGR_DS_GetSummaryRefCount().

Returns

the current summary reference count for the datasource and its layers.

References GetLayer(), GetLayerCount(), and OGRLayer::GetRefCount().

11.40.2.15 int OGRDataSource::Reference ( )

Increment datasource reference count.

This method is the same as the C function OGR_DS_Reference().

Returns

the reference count after incrementing.

Referenced by OGRSFDriverRegistrar::Open().

11.40.2.16 OGRErr OGRDataSource::Release ( )

Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.

Internally this actually calls the OGRSFDriverRegistrar::ReleaseDataSource() method. This method is essentially a convenient alias.

This method is the same as the C function **OGRReleaseDataSource()** (p. 483).

Returns

OGRERR_NONE on success or an error code.

References OGRSFDriverRegistrar::GetRegistrar().

11.40.2.17 void OGRDataSource::ReleaseResultSet (OGRLayer * *poResultsSet*) [virtual]

Release results of **ExecuteSQL()** (p. 89).

This method should only be used to deallocate OGRLayers resulting from an **ExecuteSQL()** (p. 89) call on the same **OGRDataSource** (p. 85). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 85) may cause errors.

This method is the same as the C function OGR_L_ReleaseResultSet().

Parameters

poResultsSet	the result of a previous **ExecuteSQL()** (p. 89) call.

11.40.2.18 void OGRDataSource::SetDriver (OGRSFDriver * *poDriver*)

Sets the driver that the dataset was created or opened with.

Note

This method is not exposed as the OGR C API function.

Parameters

poDriver	pointer to driver instance associated with the data source.

Referenced by OGRSFDriver::CopyDataSource(), OGR_Dr_CopyDataSource(), OGR_Dr_CreateDataSource(), and OGR_Dr_Open().

11.40.2.19 void OGRDataSource::SetStyleTable (OGRStyleTable * *poStyleTable*) [virtual]

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTableDirectly()** (p. 92) except that it does not assume ownership of the passed table.

This method is the same as the C function OGR_DS_SetStyleTable().

Parameters

poStyleTable	pointer to style table to set

References OGRStyleTable::Clone().

11.40.2.20 void OGRDataSource::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [virtual]

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTable()** (p. 92) except that it assumes ownership of the passed table.

This method is the same as the C function OGR_DS_SetStyleTableDirectly().

Parameters

poStyleTable	pointer to style table to set

11.40.2.21 OGRErr OGRDataSource::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

This call is intended to force the datasource to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some data sources do not implement this method, and will still return OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

The default implementation of this method just calls the **SyncToDisk()** (p. 92) method on each of the layers. Conceptionally, calling **SyncToDisk()** (p. 92) on a datasource should include any work that might be accomplished by

calling **SyncToDisk()** (p. 92) on layers in that data source.

In any event, you should always close any opened datasource with **OGRDataSource::DestroyDataSource()** (p. 88) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR_DS_SyncToDisk()** (p. 402).

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

References GetLayer(), GetLayerCount(), and OGRLayer::SyncToDisk().

11.40.2.22 int OGRDataSource::TestCapability (const char * *pszCapability*) [pure virtual]

Test if capability is available.

One of the following data source capability names can be passed into this method, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODsCCreateLayer**: True if this datasource can create new layers.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid mispelling.

This method is the same as the C function **OGR_DS_TestCapability()** (p. 402).

Parameters

pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

Referenced by CopyLayer().

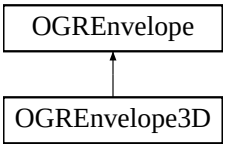
The documentation for this class was generated from the following files:

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrdatasource.cpp

11.41 OGREnvelope Class Reference

`#include <ogr_core.h>`

Inheritance diagram for OGREnvelope:



11.41.1 Detailed Description

Simple container for a bounding region.

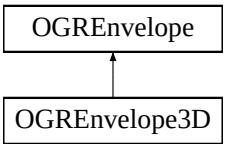
The documentation for this class was generated from the following file:

- **ogr_core.h**

11.42 OGREnvelope3D Class Reference

`#include <ogr_core.h>`

Inheritance diagram for OGREnvelope3D:



11.42.1 Detailed Description

Simple container for a bounding region in 3D.

The documentation for this class was generated from the following file:

- **ogr_core.h**

11.43 OGRFeature Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFeature** (**OGRFeatureDefn** ∗)

  Constructor.
- **OGRFeatureDefn** ∗ **GetDefnRef** ()

  Fetch feature definition.
- OGRErr **SetGeometryDirectly** (**OGRGeometry** ∗)

  Set feature geometry.
- OGRErr **SetGeometry** (**OGRGeometry** ∗)

  Set feature geometry.
- **OGRGeometry** ∗ **GetGeometryRef** ()

  Fetch pointer to feature geometry.
- **OGRGeometry** ∗ **StealGeometry** ()

  Take away ownership of geometry.
- **OGRFeature** ∗ **Clone** ()

  Duplicate feature.
- virtual OGRBoolean **Equal** (**OGRFeature** ∗poFeature)

  Test if two features are the same.
- int **GetFieldCount** ()

  *Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. 109).*
- **OGRFieldDefn** ∗ **GetFieldDefnRef** (int iField)

Fetch definition for this field.

- int **GetFieldIndex** (const char *pszName)

  Fetch the field index given field name.

- int **IsFieldSet** (int iField) const

  Test if a field has ever been assigned a value or not.

- void **UnsetField** (int iField)

  Clear a field, marking it as unset.

- **OGRField** * **GetRawFieldRef** (int i)

  Fetch a pointer to the internal field value given the index.

- int **GetFieldAsInteger** (int i)

  Fetch field value as integer.

- double **GetFieldAsDouble** (int i)

  Fetch field value as a double.

- const char * **GetFieldAsString** (int i)

  Fetch field value as a string.

- const int * **GetFieldAsIntegerList** (int i, int *pnCount)

  Fetch field value as a list of integers.

- const double * **GetFieldAsDoubleList** (int i, int *pnCount)

  Fetch field value as a list of doubles.

- char ** **GetFieldAsStringList** (int i) const

  Fetch field value as a list of strings.

- GByte * **GetFieldAsBinary** (int i, int *pnCount)

  Fetch field value as binary data.

- int **GetFieldAsDateTime** (int i, int *pnYear, int *pnMonth, int *pnDay, int *pnHour, int *pnMinute, int *pnSecond, int *pnTZFlag)

  Fetch field value as date and time.

- void **SetField** (int i, int nValue)

  Set field to integer value.

- void **SetField** (int i, double dfValue)

  Set field to double value.

- void **SetField** (int i, const char *pszValue)

  Set field to string value.

- void **SetField** (int i, int nCount, int *panValues)

  Set field to list of integers value.

- void **SetField** (int i, int nCount, double *padfValues)

  Set field to list of doubles value.

- void **SetField** (int i, char **papszValues)

  Set field to list of strings value.

- void **SetField** (int i, **OGRField** *puValue)

  Set field.

- void **SetField** (int i, int nCount, GByte *pabyBinary)

  Set field to binary data.

- void **SetField** (int i, int nYear, int nMonth, int nDay, int nHour=0, int nMinute=0, int nSecond=0, int nTZFlag=0)

  Set field to date.

- long **GetFID** ()

  Get feature identifier.

- virtual OGRErr **SetFID** (long nFID)

  Set the feature identifier.

- void **DumpReadable** (FILE *, char **papszOptions=NULL)

  Dump this feature in a human readable form.

- OGRErr **SetFrom** (**OGRFeature** ∗, int=TRUE)

  Set one feature from another.
- OGRErr **SetFrom** (**OGRFeature** ∗, int ∗, int=TRUE)

  Set one feature from another.
- virtual const char ∗ **GetStyleString** ()

  Fetch style string for this feature.
- virtual void **SetStyleString** (const char ∗)

  *Set feature style string. This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. 108) except that it does not assume ownership of the passed string, but instead makes a copy of it.*
- virtual void **SetStyleStringDirectly** (char ∗)

  *Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. 108) except that it assumes ownership of the passed string.*

Static Public Member Functions

- static **OGRFeature** ∗ **CreateFeature** (**OGRFeatureDefn** ∗)

  Feature factory.
- static void **DestroyFeature** (**OGRFeature** ∗)

  Destroy feature.

11.43.1 Detailed Description

A simple feature, including geometry and attributes.

11.43.2 Constructor & Destructor Documentation

11.43.2.1 OGRFeature::OGRFeature (OGRFeatureDefn ∗ *poDefnIn*)

Constructor.

Note that the **OGRFeature** (p. 94) will increment the reference count of it's defining **OGRFeatureDefn** (p. 109). Destruction of the **OGRFeatureDefn** (p. 109) before destruction of all OGRFeatures that depend on it is likely to result in a crash.

This method is the same as the C function **OGR_F_Create()** (p. 403).

Parameters

poDefnIn	feature class (layer) definition to which the feature will adhere.

References OGRFeatureDefn::GetFieldCount(), and OGRFeatureDefn::Reference().

Referenced by Clone(), and CreateFeature().

11.43.3 Member Function Documentation

11.43.3.1 OGRFeature ∗ OGRFeature::Clone ( )

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. 109).

This method is the same as the C function **OGR_F_Clone()** (p. 403).

Returns

new feature, exactly matching this feature.

References GetFID(), OGRFeatureDefn::GetFieldCount(), GetStyleString(), OGRFeature(), SetFID(), SetField(), SetGeometry(), and SetStyleString().

Referenced by OGRGenSQLResultsLayer::GetFeature().

11.43.3.2 OGRFeature * OGRFeature::CreateFeature (OGRFeatureDefn * *poDefn*) [static]

Feature factory.

This is essentially a feature factory, useful for applications creating features but wanting to ensure they are created out of the OGR/GDAL heap.

This method is the same as the C function **OGR_F_Create()** (p. 403).

Parameters

poDefn	Feature definition defining schema.

Returns

new feature object with null fields and no geometry. May be deleted with delete.

References OGRFeature().

Referenced by OGRDataSource::CopyLayer().

11.43.3.3 void OGRFeature::DestroyFeature (OGRFeature * *poFeature*) [static]

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This method is the same as the C function **OGR_F_Destroy()** (p. 403).

Parameters

poFeature	the feature to delete.

Referenced by OGRDataSource::CopyLayer().

11.43.3.4 void OGRFeature::DumpReadable (FILE * *fpOut,* char ** *papszOptions* = NULL)

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

A few options can be defined to change the default dump :

- DISPLAY_FIELDS=NO : to hide the dump of the attributes
- DISPLAY_STYLE=NO : to hide the dump of the style string
- DISPLAY_GEOMETRY=NO : to hide the dump of the geometry
- DISPLAY_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR_F_DumpReadable()** (p. 404).

Parameters

fpOut	the stream to write to, such as stdout. If NULL stdout will be used.
papszOptions	NULL terminated list of options (may be NULL)

References OGRGeometry::dumpReadable(), GetFID(), GetFieldAsString(), GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetFieldTypeName(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), GetStyleString(), OGRFieldDefn::GetType(), and IsFieldSet().

11.43.3.5 OGRBoolean OGRFeature::Equal (OGRFeature * *poFeature*) [virtual]

Test if two features are the same.

Two features are considered equal if the share them (pointer equality) same **OGRFeatureDefn** (p. 109), have the same field values, and the same geometry (as tested by OGRGeometry::Equal()) as well as the same feature id.

This method is the same as the C function **OGR_F_Equal()** (p. 404).

Parameters

poFeature	the other feature to test this one against.

Returns

TRUE if they are equal, otherwise FALSE.

References OGRGeometry::Equals(), GetDefnRef(), GetFID(), GetFieldAsBinary(), GetFieldAsDateTime(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeometryRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

11.43.3.6 OGRFeatureDefn * OGRFeature::GetDefnRef ( ) [inline]

Fetch feature definition.

This method is the same as the C function **OGR_F_GetDefnRef()** (p. 404).

Returns

a reference to the feature definition object.

Referenced by Equal().

11.43.3.7 long OGRFeature::GetFID ( ) [inline]

Get feature identifier.

This method is the same as the C function **OGR_F_GetFID()** (p. 404).

Returns

feature id or OGRNullFID if none has been assigned.

Referenced by Clone(), OGRDataSource::CopyLayer(), DumpReadable(), Equal(), OGRLayer::GetFeature(), GetFieldAsDouble(), GetFieldAsInteger(), and GetFieldAsString().

11.43.3.8 GByte * OGRFeature::GetFieldAsBinary (int *iField,* int * *pnBytes*)

Fetch field value as binary data.

Currently this method only works for OFTBinary fields.

This method is the same as the C function **OGR_F_GetFieldAsBinary()** (p. 405).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
pnBytes	location to put the number of bytes returned.

Returns

the field value. This data is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTBinary.

Referenced by Equal().

11.43.3.9 int OGRFeature::GetFieldAsDateTime (int *iField,* int * *pnYear,* int * *pnMonth,* int * *pnDay,* int * *pnHour,* int * *pnMinute,* int * *pnSecond,* int * *pnTZFlag*)

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR_F_GetFieldAsDateTime()** (p. 405).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
pnYear	(including century)
pnMonth	(1-12)
pnDay	(1-31)
pnHour	(0-23)
pnMinute	(0-59)
pnSecond	(0-59)
pnTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns

TRUE on success or FALSE on failure.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, and OFTTime.

Referenced by Equal().

11.43.3.10 double OGRFeature::GetFieldAsDouble (int *iField*)

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsDouble()** (p. 406).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the field value.

References GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTReal, and OFTString.

Referenced by Equal(), and SetFrom().

11.43.3.11 const double * OGRFeature::GetFieldAsDoubleList (int *iField,* int * *pnCount*)

Fetch field value as a list of doubles.

Currently this method only works for OFTRealList fields.

This method is the same as the C function **OGR_F_GetFieldAsDoubleList()** (p. 406).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
pnCount	an integer to put the list count (number of doubles) into.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTRealList.

Referenced by Equal(), and SetFrom().

11.43.3.12 int OGRFeature::GetFieldAsInteger (int *iField*)

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsInteger()** (p. 406).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the field value.

References GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTReal, and OFTString.

Referenced by Equal(), and SetFrom().

11.43.3.13 const int * OGRFeature::GetFieldAsIntegerList (int *iField,* int * *pnCount*)

Fetch field value as a list of integers.

Currently this method only works for OFTIntegerList fields.

This method is the same as the C function **OGR_F_GetFieldAsIntegerList()** (p. 407).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
pnCount	an integer to put the list count (number of integers) into.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTIntegerList.

Referenced by Equal(), and SetFrom().

11.43.3.14 const char * OGRFeature::GetFieldAsString (int *iField*)

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using sprintf(), but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsString()** (p. 407).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRGeometry::exportToWkt(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRGeometry::getGeometryName(), OGRFieldDefn::GetPrecision(), GetStyleString(), OGRFieldDefn::GetType(), OGRFieldDefn::GetWidth(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by DumpReadable(), Equal(), GetStyleString(), and SetFrom().

11.43.3.15 char ** OGRFeature::GetFieldAsStringList (int *iField*) const

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. 365).

This method is the same as the C function **OGR_F_GetFieldAsStringList()** (p. 407).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTStringList.

Referenced by Equal().

11.43.3.16 int OGRFeature::GetFieldCount ( ) [inline]

Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. 109).

This method is the same as the C function **OGR_F_GetFieldCount()** (p. 408).

Returns

count of fields.

Referenced by DumpReadable(), OGR_F_IsFieldSet(), and SetFrom().

11.43.3.17 OGRFieldDefn * OGRFeature::GetFieldDefnRef (int *iField*) [inline]

Fetch definition for this field.

This method is the same as the C function **OGR_F_GetFieldDefnRef()** (p. 408).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the field definition (from the **OGRFeatureDefn** (p. 109)). This is an internal reference, and should not be deleted or modified.

Referenced by SetFrom().

11.43.3.18 int OGRFeature::GetFieldIndex (const char * *pszName*) [inline]

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. 112) method.

This method is the same as the C function **OGR_F_GetFieldIndex()** (p. 409).

Parameters

pszName	the name of the field to search for.

Returns

the field index, or -1 if no matching field is found.

Referenced by GetStyleString(), and SetFrom().

11.43.3.19 OGRGeometry * OGRFeature::GetGeometryRef ( ) [inline]

Fetch pointer to feature geometry.

This method is the same as the C function **OGR_F_GetGeometryRef()** (p. 409).

Returns

pointer to internal feature geometry. This object should not be modified.

Referenced by Equal(), OGRLayer::GetExtent(), and SetFrom().

11.43.3.20 OGRField * OGRFeature::GetRawFieldRef (int *iField*) [inline]

Fetch a pointer to the internal field value given the index.

This method is the same as the C function **OGR_F_GetRawFieldRef()** (p. 409).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.

Returns

the returned pointer is to an internal data structure, and should not be freed, or modified.

Referenced by SetFrom().

11.43.3.21 const char * OGRFeature::GetStyleString ( ) [virtual]

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr_featurestyle.h** (p. 489) for services available to parse it.

This method is the same as the C function **OGR_F_GetStyleString()** (p. 410).

Returns

a reference to a representation in string format, or NULL if there isn't one.

References GetFieldAsString(), and GetFieldIndex().

Referenced by Clone(), DumpReadable(), GetFieldAsString(), OGRStyleMgr::InitFromFeature(), and SetFrom().

11.43.3.22 int OGRFeature::IsFieldSet (int *iField*) const

Test if a field has ever been assigned a value or not.

This method is the same as the C function **OGR_F_IsFieldSet()** (p. 410).

Parameters

iField	the field to test.

Returns

TRUE if the field has been set, otherwise false.

References OGRFeatureDefn::GetFieldCount().

Referenced by DumpReadable(), Equal(), GetFieldAsBinary(), GetFieldAsDateTime(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGR_F_IsFieldSet(), SetField(), SetFrom(), and UnsetField().

11.43.3.23 OGRErr OGRFeature::SetFID (long *nFID*) [virtual]

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This method is the same as the C function **OGR_F_SetFID()** (p. 410).

Parameters

nFID	the new feature identifier value to assign.

Returns

On success OGRERR_NONE, or on failure some other value.

Referenced by Clone(), OGRDataSource::CopyLayer(), OGRGenSQLResultsLayer::GetFeature(), and SetFrom().

11.43.3.24 void OGRFeature::SetField (int *iField,* int *nValue*)

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldInteger()** (p. 412).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
nValue	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and OFTString.

Referenced by Clone(), OGRGenSQLResultsLayer::GetFeature(), SetField(), and SetFrom().

11.43.3.25 void OGRFeature::SetField (int *iField,* double *dfValue*)

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldDouble()** (p. 411).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
dfValue	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, and SetField().

11.43.3.26 void OGRFeature::SetField (int *iField*, const char * *pszValue*)

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldString()** (p. 413).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
pszValue	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTTime, OGRParseDate(), and SetField().

11.43.3.27 void OGRFeature::SetField (int *iField*, int *nCount*, int * *panValues*)

Set field to list of integers value.

This method currently on has an effect of OFTIntegerList fields.

This method is the same as the C function **OGR_F_SetFieldIntegerList()** (p. 412).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. 102)-1.
nCount	the number of values in the list being assigned.
panValues	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and SetField().

11.43.3.28 void OGRFeature::SetField (int *iField*, int *nCount*, double * *padfValues*)

Set field to list of doubles value.

This method currently on has an effect of OFTRealList fields.

This method is the same as the C function **OGR_F_SetFieldDoubleList()** (p. 412).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. 102)-1.
nCount	the number of values in the list being assigned.
padfValues	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and SetField().

11.43.3.29 void OGRFeature::SetField (int *iField*, char ** *papszValues*)

Set field to list of strings value.

This method currently on has an effect of OFTStringList fields.

This method is the same as the C function **OGR_F_SetFieldStringList()** (p. 413).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. 102)-1.
papszValues	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTStringList, and SetField().

11.43.3.30 void OGRFeature::SetField (int *iField*, OGRField * *puValue*)

Set field.

The passed value **OGRField** (p. 116) must be of exactly the same type as the target field, or an application crash may occur. The passed value is copied, and will not be affected. It remains the responsibility of the caller.

This method is the same as the C function **OGR_F_SetFieldRaw()** (p. 413).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. 102)-1.
puValue	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

11.43.3.31 void OGRFeature::SetField (int *iField*, int *nBytes*, GByte * *pabyData*)

Set field to binary data.

This method currently on has an effect of OFTBinary fields.

This method is the same as the C function **OGR_F_SetFieldBinary()** (p. 411).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. 102)-1.
nBytes	bytes of data being set.
pabyData	the raw data being applied.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTBinary, and SetField().

11.43.3.32 void OGRFeature::SetField (int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour* = 0, int *nMinute* = 0, int *nSecond* = 0, int *nTZFlag* = 0)

Set field to date.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR_F_SetFieldDateTime()** (p. 411).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. 102)-1.
nYear	(including century)
nMonth	(1-12)
nDay	(1-31)
nHour	(0-23)
nMinute	(0-59)
nSecond	(0-59)
nTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTDate, OFTDateTime, and OFTTime.

11.43.3.33 OGRErr OGRFeature::SetFrom (OGRFeature * *poSrcFeature,* int *bForgiving* = TRUE)

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The poSrcFeature does not need to have the same **OGRFeatureDefn** (p. 109). Field values are copied by corresponding field names. Field types do not have to exactly match. **SetField()** (p. 104) method conversion rules will be applied as needed.

This method is the same as the C function **OGR_F_SetFrom()** (p. 413).

Parameters

poSrcFeature	the feature from which geometry, and field values will be copied.
bForgiving	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References GetFieldCount(), GetFieldDefnRef(), GetFieldIndex(), and OGRFieldDefn::GetNameRef().

Referenced by OGRDataSource::CopyLayer().

11.43.3.34 OGRErr OGRFeature::SetFrom (OGRFeature * *poSrcFeature,* int * *panMap,* int *bForgiving* = TRUE)

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The poSrcFeature does not need to have the same **OGRFeatureDefn** (p. 109). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. 104) method conversion rules will be applied as needed. This is more efficient than **OGR_F_SetFrom()** (p. 413) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This method is the same as the C function **OGR_F_SetFromWithMap()** (p. 414).

Parameters

poSrcFeature	the feature from which geometry, and field values will be copied.
panMap	Array of the indices of the feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
bForgiving	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldCount(), GetFieldDefnRef(), GetGeometryRef(), GetRawFieldRef(), GetStyleString(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTTime, SetFID(), SetField(), SetGeometry(), SetStyleString(), and UnsetField().

11.43.3.35 OGRErr OGRFeature::SetGeometry (OGRGeometry * *poGeomIn*)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometryDirectly()** (p. 108), except that this method does not assume ownership of the passed geometry, but instead makes a copy of it.

This method is the same as the C function **OGR_F_SetGeometry()** (p. 414).

Parameters

poGeomIn	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

Returns

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 109) (checking not yet implemented).

References OGRGeometry::clone().

Referenced by Clone(), and SetFrom().

11.43.3.36 OGRErr OGRFeature::SetGeometryDirectly (OGRGeometry * *poGeomIn*)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometry()** (p. 107), except that this method assumes ownership of the passed geometry.

This method is the same as the C function **OGR_F_SetGeometryDirectly()** (p. 415).

Parameters

poGeomIn	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

Returns

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 109) (checking not yet implemented).

11.43.3.37 void OGRFeature::SetStyleString (const char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. 108) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This method is the same as the C function **OGR_F_SetStyleString()** (p. 415).

Parameters

pszString	the style string to apply to this feature, cannot be NULL.

Referenced by Clone(), OGRStyleMgr::SetFeatureStyleString(), and SetFrom().

11.43.3.38 void OGRFeature::SetStyleStringDirectly (char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. 108) except that it assumes ownership of the passed string.

This method is the same as the C function **OGR_F_SetStyleStringDirectly()** (p. 415).

Parameters

pszString	the style string to apply to this feature, cannot be NULL.

11.43.3.39 OGRGeometry * OGRFeature::StealGeometry ( )

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownship of the geometry from the feature without copying. Sort of an inverse to **SetGeometryDirectly()** (p. 108).

After this call the **OGRFeature** (p. 94) will have a NULL geometry.

Returns

the pointer to the geometry.

11.43.3.40 void OGRFeature::UnsetField (int *iField*)

Clear a field, marking it as unset.

This method is the same as the C function **OGR_F_UnsetField()** (p. 416).

Parameters

iField	the field to unset.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTIntegerList, OFTRealList, OFTString, and OFTStringList.

Referenced by OGRGenSQLResultsLayer::GetFeature(), and SetFrom().

The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrfeature.cpp

11.44 OGRFeatureDefn Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFeatureDefn** (const char *pszName=NULL)
  Constructor.
- const char * **GetName** ()
  *Get name of this **OGRFeatureDefn** (p. 109).*
- int **GetFieldCount** ()
  Fetch number of fields on this feature.
- **OGRFieldDefn** * **GetFieldDefn** (int i)
  Fetch field definition.
- int **GetFieldIndex** (const char *)
  Find field by name.
- void **AddFieldDefn** (**OGRFieldDefn** *)

Add a new field definition.

- OGRErr **DeleteFieldDefn** (int iField)

  Delete an existing field definition.

- OGRErr **ReorderFieldDefns** (int *panMap)

  Reorder the field definitions in the array of the feature definition.

- **OGRwkbGeometryType GetGeomType** ()

  Fetch the geometry base type.

- void **SetGeomType** (**OGRwkbGeometryType**)

  Assign the base geometry type for this layer.

- **OGRFeatureDefn** * **Clone** ()

  Create a copy of this feature definition.

- int **Reference** ()

  Increments the reference count by one.

- int **Dereference** ()

  Decrements the reference count by one.

- int **GetReferenceCount** ()

  Fetch current reference count.

- void **Release** ()

  Drop a reference to this object, and destroy if no longer referenced.

- int **IsGeometryIgnored** ()

  Determine whether the geometry can be omitted when fetching features.

- void **SetGeometryIgnored** (int bIgnore)

  Set whether the geometry can be omitted when fetching features.

- int **IsStyleIgnored** ()

  Determine whether the style can be omitted when fetching features.

- void **SetStyleIgnored** (int bIgnore)

  Set whether the style can be omitted when fetching features.

11.44.1 Detailed Description

Definition of a feature class or feature layer.

This object contains schema information for a set of OGRFeatures. In table based systems, an **OGRFeatureDefn** (p. 109) is essentially a layer. In more object oriented approaches (such as SF CORBA) this can represent a class of features but doesn't necessarily relate to all of a layer, or just one layer.

This object also can contain some other information such as a name, the base geometry type and potentially other metadata.

It is reasonable for different translators to derive classes from **OGRFeatureDefn** (p. 109) with additional translator specific information.

11.44.2 Constructor & Destructor Documentation

11.44.2.1 OGRFeatureDefn::OGRFeatureDefn (const char * *pszName* = NULL)

Constructor.

The **OGRFeatureDefn** (p. 109) maintains a reference count, but this starts at zero. It is mainly intended to represent a count of **OGRFeature** (p. 94)'s based on this definition.

This method is the same as the C function **OGR_FD_Create()** (p. 417).

Parameters

pszName	the name to be assigned to this layer/class. It does not need to be unique.

References wkbUnknown.

Referenced by Clone().

11.44.3 Member Function Documentation

11.44.3.1 void OGRFeatureDefn::AddFieldDefn (OGRFieldDefn * *poNewDefn*)

Add a new field definition.

To add a new field definition to a layer definition, do not use this function directly, but use **OGRLayer::CreateField()** (p. 165) instead.

This method should only be called while there are no **OGRFeature** (p. 94) objects in existance based on this **OGRFeatureDefn** (p. 109). The **OGRFieldDefn** (p. 116) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR_FD_AddFieldDefn()** (p. 416).

Parameters

poNewDefn	the definition of the new field.

Referenced by Clone().

11.44.3.2 OGRFeatureDefn * OGRFeatureDefn::Clone ( )

Create a copy of this feature definition.

Creates a deep copy of the feature definition.

Returns

the copy.

References AddFieldDefn(), GetFieldCount(), GetFieldDefn(), GetGeomType(), GetName(), OGRFeatureDefn(), and SetGeomType().

11.44.3.3 OGRErr OGRFeatureDefn::DeleteFieldDefn (int *iField*)

Delete an existing field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use **OGRLayer::DeleteField()** (p. 166) instead.

This method should only be called while there are no **OGRFeature** (p. 94) objects in existance based on this **OGRFeatureDefn** (p. 109).

This method is the same as the C function **OGR_FD_DeleteFieldDefn()** (p. 417).

Parameters

iField	the index of the field defintion.

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

11.44.3.4 int OGRFeatureDefn::Dereference ( ) [inline]

Decrements the reference count by one.

This method is the same as the C function **OGR_FD_Dereference()** (p. 417).

Returns

the updated reference count.

Referenced by Release().

11.44.3.5 int OGRFeatureDefn::GetFieldCount ( ) [inline]

Fetch number of fields on this feature.

This method is the same as the C function **OGR_FD_GetFieldCount()** (p. 418).

Returns

count of fields.

Referenced by Clone(), OGRFeature::Clone(), OGRDataSource::CopyLayer(), OGRFeature::Equal(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsString(), OGRFeature::IsFieldSet(), OGRFeature::OGRFeature(), OGRLayer::ReorderField(), and OGRLayer::SetIgnoredFields().

11.44.3.6 OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (int *iField*)

Fetch field definition.

This method is the same as the C function **OGR_FD_GetFieldDefn()** (p. 418).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

Parameters

iField	the field to fetch, between 0 and **GetFieldCount()** (p. 112)-1.

Returns

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

Referenced by Clone(), OGRDataSource::CopyLayer(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFeature::SetField(), OGRLayer::SetIgnoredFields(), and OGRFeature::UnsetField().

11.44.3.7 int OGRFeatureDefn::GetFieldIndex (const char * *pszFieldName*)

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This method is the same as the C function **OGR_FD_GetFieldIndex()** (p. 419).

Parameters

pszFieldName	the field name to search for.

Returns

the field index, or -1 if no match found.

Referenced by OGRLayer::SetIgnoredFields().

11.44.3.8 OGRwkbGeometryType OGRFeatureDefn::GetGeomType ( ) [inline]

Fetch the geometry base type.

Note that some drivers are unable to determine a specific geometry type for a layer, in which case wkbUnknown is returned. A value of wkbNone indicates no geometry is available for the layer at all. Many drivers do not properly mark the geometry type as 25D even if some or all geometries are in fact 25D. A few (broken) drivers return wkbPolygon for layers that also include wkbMultiPolygon.

This method is the same as the C function **OGR_FD_GetGeomType()** (p. 419).

Returns

the base type for all geometry related to this definition.

Referenced by Clone(), OGRDataSource::CopyLayer(), and OGRLayer::GetGeomType().

11.44.3.9 const char * OGRFeatureDefn::GetName ( ) [inline]

Get name of this **OGRFeatureDefn** (p. 109).

This method is the same as the C function **OGR_FD_GetName()** (p. 419).

Returns

the name. This name is internal and should not be modified, or freed.

Referenced by Clone(), OGRSFDriver::CopyDataSource(), OGRDataSource::CopyLayer(), OGRFeature::DumpReadable(), and OGRLayer::GetName().

11.44.3.10 int OGRFeatureDefn::GetReferenceCount ( ) [inline]

Fetch current reference count.

This method is the same as the C function **OGR_FD_GetReferenceCount()** (p. 419).

Returns

the current reference count.

11.44.3.11 int OGRFeatureDefn::IsGeometryIgnored ( ) [inline]

Determine whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR_FD_IsGeometryIgnored()** (p. 420).

Returns

ignore state

11.44.3.12 int OGRFeatureDefn::IsStyleIgnored ( ) [inline]

Determine whether the style can be omitted when fetching features.

This method is the same as the C function **OGR_FD_IsStyleIgnored()** (p. 420).

Returns

ignore state

11.44.3.13 int OGRFeatureDefn::Reference ( ) [inline]

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. 94) objects referencing this definition.

This method is the same as the C function **OGR_FD_Reference()** (p. 420).

Returns

the updated reference count.

Referenced by OGRFeature::OGRFeature().

11.44.3.14 OGRErr OGRFeatureDefn::ReorderFieldDefns (int * *panMap*)

Reorder the field definitions in the array of the feature definition.

To reorder the field definitions in a layer definition, do not use this function directly, but use **OGR_L_ReorderFields()** (p. 464) instead.

This method should only be called while there are no **OGRFeature** (p. 94) objects in existance based on this **OGRFeatureDefn** (p. 109).

This method is the same as the C function OGR_FD_ReorderFieldDefns().

Parameters

panMap	an array of **GetFieldCount()** (p. 112) elements which is a permutation of [0, **GetFieldCount()** (p. 112)-1]. panMap is such that, for each field definition at position i after reordering, its position before reordering was panMap[i].

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

11.44.3.15 void OGRFeatureDefn::SetGeometryIgnored (int *bIgnore*) [inline]

Set whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR_FD_SetGeometryIgnored()** (p. 421).

Parameters

bIgnore	ignore state

Referenced by OGRLayer::SetIgnoredFields().

11.44.3.16 void OGRFeatureDefn::SetGeomType (OGRwkbGeometryType *eNewType*)

Assign the base geometry type for this layer.

All geometry objects using this type must be of the defined type or a derived type. The default upon creation is wkbUnknown which allows for any geometry type. The geometry type should generally not be changed after any OGRFeatures have been created against this definition.

This method is the same as the C function **OGR_FD_SetGeomType()** (p. 421).

Parameters

eNewType	the new type to assign.

Referenced by Clone().

11.44.3.17 void OGRFeatureDefn::SetStyleIgnored (int *bIgnore*) [inline]

Set whether the style can be omitted when fetching features.

This method is the same as the C function **OGR_FD_SetStyleIgnored()** (p. 422).

Parameters

bIgnore	ignore state

Referenced by OGRLayer::SetIgnoredFields().

The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrfeaturedefn.cpp

11.45 OGRFeatureQuery Class Reference

Public Member Functions

- char ** **GetUsedFields** ()

11.45.1 Member Function Documentation

11.45.1.1 char ** OGRFeatureQuery::GetUsedFields ( )

Returns lists of fields in expression.

All attribute fields are used in the expression of this feature query are returned as a StringList of field names. This function would primarily be used within drivers to recognise special case conditions depending only on attribute fields that can be very efficiently fetched.

NOTE: If any fields in the expression are from tables other than the primary table then NULL is returned indicating an error. In succesful use, no non-empty expression should return an empty list.

Returns

list of field names. Free list with **CSLDestroy()** (p. 365) when no longer required.

The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrfeaturequery.cpp

11.46 OGRField Union Reference

`#include <ogr_core.h>`

11.46.1 Detailed Description

OGRFeature (p. 94) field attribute value union.

The documentation for this union was generated from the following file:

- **ogr_core.h**

11.47 OGRFieldDefn Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFieldDefn** (const char *, **OGRFieldType**)

  Constructor.
- **OGRFieldDefn** (**OGRFieldDefn** *)

  Constructor.
- void **SetName** (const char *)

  Reset the name of this field.
- const char * **GetNameRef** ()

  Fetch name of this field.
- **OGRFieldType GetType** ()

  Fetch type of this field.
- void **SetType** (**OGRFieldType** eTypeIn)

  *Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 116) that is already part of an **OGRFeatureDefn** (p. 109).*
- **OGRJustification GetJustify** ()

  Get the justification for this field.
- void **SetJustify** (**OGRJustification** eJustifyIn)

  Set the justification for this field.
- int **GetWidth** ()

  Get the formatting width for this field.
- void **SetWidth** (int nWidthIn)

  Set the formatting width for this field in characters.
- int **GetPrecision** ()

  Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.
- void **SetPrecision** (int nPrecisionIn)

Set the formatting precision for this field in characters.

- void **Set** (const char *, **OGRFieldType**, int=0, int=0, **OGRJustification**=OJUndefined)

  Set defining parameters for a field in one call.
- void **SetDefault** (const **OGRField** *)

  Set default field value.
- int **IsIgnored** ()

  Return whether this field should be omitted when fetching features.
- void **SetIgnored** (int bIgnore)

  Set whether this field should be omitted when fetching features.

Static Public Member Functions

- static const char * **GetFieldTypeName** (**OGRFieldType**)

  Fetch human readable name for a field type.

11.47.1 Detailed Description

Definition of an attribute of an **OGRFeatureDefn** (p. 109).

11.47.2 Constructor & Destructor Documentation

11.47.2.1 OGRFieldDefn::OGRFieldDefn (const char * *pszNameIn*, OGRFieldType *eTypeIn*)

Constructor.

Parameters

pszNameIn	the name of the new field.
eTypeIn	the type of the new field.

11.47.2.2 OGRFieldDefn::OGRFieldDefn (OGRFieldDefn * *poPrototype*)

Constructor.

Create by cloning an existing field definition.

Parameters

poPrototype	the field definition to clone.

References GetJustify(), GetNameRef(), GetPrecision(), GetType(), GetWidth(), SetJustify(), SetPrecision(), and SetWidth().

11.47.3 Member Function Documentation

11.47.3.1 const char * OGRFieldDefn::GetFieldTypeName (OGRFieldType *eType*) [static]

Fetch human readable name for a field type.

This static method is the same as the C function **OGR_GetFieldTypeName()** (p. 456).

Parameters

eType	the field type to get name for.

Returns

pointer to an internal static name string. It should not be modified or freed.

References OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by OGRFeature::DumpReadable(), and OGR_GetFieldTypeName().

11.47.3.2 OGRJustification OGRFieldDefn::GetJustify ( ) [inline]

Get the justification for this field.

This method is the same as the C function **OGR_Fld_GetJustify()** (p. 422).

Returns

the justification.

Referenced by OGRFieldDefn().

11.47.3.3 const char * OGRFieldDefn::GetNameRef ( ) [inline]

Fetch name of this field.

This method is the same as the C function **OGR_Fld_GetNameRef()** (p. 423).

Returns

pointer to an internal name string that should not be freed or modified.

Referenced by OGRFeature::DumpReadable(), OGRDataSource::ExecuteSQL(), OGRFieldDefn(), and OGRFeature::SetFrom().

11.47.3.4 int OGRFieldDefn::GetPrecision ( ) [inline]

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR_Fld_GetPrecision()** (p. 423).

Returns

the precision.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

11.47.3.5 OGRFieldType OGRFieldDefn::GetType ( ) [inline]

Fetch type of this field.

This method is the same as the C function **OGR_Fld_GetType()** (p. 423).

Returns

field type.

Referenced by OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFieldDefn(), OGRFeature::SetField(), OGRFeature::SetFrom(), and OGRFeature::UnsetField().

11.47.3.6 int OGRFieldDefn::GetWidth ( ) [inline]

Get the formatting width for this field.

This method is the same as the C function **OGR_Fld_GetWidth()** (p. 424).

Returns

the width, zero means no specified width.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

11.47.3.7 int OGRFieldDefn::IsIgnored ( ) [inline]

Return whether this field should be omitted when fetching features.

This method is the same as the C function **OGR_Fld_IsIgnored()** (p. 424).

Returns

ignore state

11.47.3.8 void OGRFieldDefn::Set (const char * *pszNameIn*, OGRFieldType *eTypeIn*, int *nWidthIn* = 0, int *nPrecisionIn* = 0, OGRJustification *eJustifyIn* = OJUndefined)

Set defining parameters for a field in one call.

This method is the same as the C function **OGR_Fld_Set()** (p. 424).

Parameters

pszNameIn	the new name to assign.
eTypeIn	the new type (one of the OFT values like OFTInteger).
nWidthIn	the preferred formatting width. Defaults to zero indicating undefined.
nPrecisionIn	number of decimals places for formatting, defaults to zero indicating undefined.
eJustifyIn	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetJustify(), SetName(), SetPrecision(), SetType(), and SetWidth().

11.47.3.9 void OGRFieldDefn::SetDefault (const OGRField * *puDefaultIn*)

Set default field value.

Currently use of **OGRFieldDefn** (p. 116) "defaults" is discouraged. This feature may be fleshed out in the future.

References OFTInteger, OFTReal, and OFTString.

11.47.3.10 void OGRFieldDefn::SetIgnored (int *ignore*) [inline]

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGR_Fld_SetIgnored()** (p. 424).

Parameters

ignore	ignore state

Referenced by OGRLayer::SetIgnoredFields().

11.47.3.11 void OGRFieldDefn::SetJustify (OGRJustification *eJustify*) [inline]

Set the justification for this field.

This method is the same as the C function **OGR_Fld_SetJustify()** (p. 425).

Parameters

eJustify	the new justification.

Referenced by OGRFieldDefn(), and Set().

11.47.3.12 void OGRFieldDefn::SetName (const char * *pszNameIn*)

Reset the name of this field.

This method is the same as the C function **OGR_Fld_SetName()** (p. 425).

Parameters

pszNameIn	the new name to apply.

Referenced by Set().

11.47.3.13 void OGRFieldDefn::SetPrecision (int *nPrecision*) [inline]

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR_Fld_SetPrecision()** (p. 425).

Parameters

nPrecision	the new precision.

Referenced by OGRFieldDefn(), and Set().

11.47.3.14 void OGRFieldDefn::SetType (OGRFieldType *eType*) [inline]

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 116) that is already part of an **OGRFeatureDefn** (p. 109).

This method is the same as the C function **OGR_Fld_SetType()** (p. 426).

Parameters

eType	the new field type.

Referenced by Set().

11.47.3.15 void OGRFieldDefn::SetWidth (int *nWidth*) [inline]

Set the formatting width for this field in characters.

This method is the same as the C function **OGR_Fld_SetWidth()** (p. 426).

Parameters

nWidth	the new width.

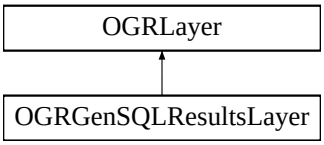
Referenced by OGRFieldDefn(), and Set().

The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrfielddefn.cpp

11.48 OGRGenSQLResultsLayer Class Reference

Inheritance diagram for OGRGenSQLResultsLayer:



Public Member Functions

- virtual **OGRGeometry** * **GetSpatialFilter** ()

  This method returns the current spatial filter for this layer.
- virtual void **ResetReading** ()

  Reset feature reading to start on the first feature.
- virtual **OGRFeature** * **GetNextFeature** ()

  Fetch the next available feature from this layer.
- virtual OGRErr **SetNextByIndex** (long nIndex)

  Move read cursor to the nIndex'th feature in the current resultset.
- virtual **OGRFeature** * **GetFeature** (long nFID)

  Fetch a feature by its identifier.
- virtual **OGRFeatureDefn** * **GetLayerDefn** ()

  Fetch the schema information for this layer.
- virtual **OGRSpatialReference** * **GetSpatialRef** ()

  Fetch the spatial reference system for this layer.
- virtual int **GetFeatureCount** (int bForce=TRUE)

  Fetch the feature count in this layer.
- virtual OGRErr **GetExtent** (**OGREnvelope** *psExtent, int bForce=TRUE)

  Fetch the extent of this layer.
- virtual int **TestCapability** (const char *)

  Test if this layer supported the named capability.

11.48.1 Member Function Documentation

11.48.1.1 OGRErr OGRGenSQLResultsLayer::GetExtent (OGREnvelope * *psExtent,* int *bForce* = TRUE) [virtual]

Fetch the extent of this layer.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (p. 122) without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

This method is the same as the C function **OGR_L_GetExtent()** (p. 459).

Parameters

psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. 167).

References OGRLayer::GetExtent().

11.48.1.2 OGRFeature * OGRGenSQLResultsLayer::GetFeature (long *nFID*) [virtual]

Fetch a feature by its identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters.

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (p. 98)) will be the same as nFID.

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. 122); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads are generally considered interrupted by a **GetFeature()** (p. 122) call.

The returned feature should be free with **OGRFeature::DestroyFeature()** (p. 97).

This method is the same as the C function **OGR_L_GetFeature()** (p. 460).

Parameters

nFID	the feature id of the feature to read.

Returns

a feature now owned by the caller, or NULL on failure.

Reimplemented from **OGRLayer** (p. 167).

References OGRFeature::Clone(), OGRLayer::GetFeature(), OGRFeature::SetFID(), OGRFeature::SetField(), and OGRFeature::UnsetField().

Referenced by GetNextFeature().

11.48.1.3 int OGRGenSQLResultsLayer::GetFeatureCount (int *bForce* = TRUE) [virtual]

Fetch the feature count in this layer.

Returns the number of features in the layer. For dynamic databases the count may not be exact. If bForce is FALSE, and it would be expensive to establish the feature count a value of -1 may be returned indicating that the count isn't know. If bForce is TRUE some implementations will actually scan the entire layer once to count objects.

The returned count takes the spatial filter into account.

Note that some implementations of this method may alter the read cursor of the layer.

This method is the same as the C function **OGR_L_GetFeatureCount()** (p. 460).

Parameters

bForce	Flag indicating whether the count should be computed even if it is expensive.

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. 168).

References OGRLayer::GetFeatureCount().

11.48.1.4 OGRFeatureDefn * OGRGenSQLResultsLayer::GetLayerDefn ( ) [virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. 109) is owned by the **OGRLayer** (p. 162), and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

This method is the same as the C function **OGR_L_GetLayerDefn()** (p. 462).

Returns

feature definition.

Implements **OGRLayer** (p. 169).

11.48.1.5 OGRFeature * OGRGenSQLResultsLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity of the caller to delete with **OGRFeature::DestroyFeature()** (p. 97). It is critical that all features associated with an **OGRLayer** (p. 162) (more specifically an **OGRFeatureDefn** (p. 109)) be deleted before that layer/datasource is deleted.

Only features matching the current spatial filter (set with **SetSpatialFilter()** (p. 174)) will be returned.

This method implements sequential access to the features of a layer. The **ResetReading()** (p. 124) method can be used to start at the beginning again.

This method is the same as the C function **OGR_L_GetNextFeature()** (p. 462).

Returns

a feature, or NULL if no more features are available.

Implements **OGRLayer** (p. 170).

References GetFeature(), and OGRLayer::GetNextFeature().

11.48.1.6 OGRGeometry * OGRGenSQLResultsLayer::GetSpatialFilter ( ) [virtual]

This method returns the current spatial filter for this layer.

The returned pointer is to an internally owned object, and should not be altered or deleted by the caller.

This method is the same as the C function **OGR_L_GetSpatialFilter()** (p. 463).

Returns

spatial filter geometry.

Reimplemented from **OGRLayer** (p. 170).

11.48.1.7 OGRSpatialReference * OGRGenSQLResultsLayer::GetSpatialRef ( ) [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 162) and should not be modified or freed by the application.

This method is the same as the C function **OGR_L_GetSpatialRef()** (p. 463).

Returns

spatial reference, or NULL if there isn't one.

Reimplemented from **OGRLayer** (p. 171).

References OGRLayer::GetSpatialRef().

11.48.1.8 void OGRGenSQLResultsLayer::ResetReading ( ) [virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. 123).

This method is the same as the C function **OGR_L_ResetReading()** (p. 465).

Implements **OGRLayer** (p. 172).

References OGRLayer::ResetReading(), OGRLayer::SetAttributeFilter(), and OGRLayer::SetSpatialFilter().

11.48.1.9 OGRErr OGRGenSQLResultsLayer::SetNextByIndex (long *nIndex*) [virtual]

Move read cursor to the nIndex'th feature in the current resultset.

This method allows positioning of a layer such that the **GetNextFeature()** (p. 123) call will read the requested feature, where nIndex is an absolute index into the current result set. So, setting it to 3 would mean the next feature read with **GetNextFeature()** (p. 123) would have been the 4th feature to have been read if sequential reading took place from the beginning of the layer, including accounting for spatial and attribute filters.

Only in rare circumstances is **SetNextByIndex()** (p. 124) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. 124) and then calls **GetNextFeature()** (p. 123) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. 125) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR_L_SetNextByIndex()** (p. 467).

Parameters

nIndex	the index indicating how many steps into the result set to seek.

Returns

OGRERR_NONE on success or an error code.

Reimplemented from **OGRLayer** (p. 174).

References OGRLayer::SetNextByIndex().

11.48.1.10 int OGRGenSQLResultsLayer::TestCapability (const char * *pszCap*) [virtual]

Test if this layer supported the named capability.

The capability codes that can be tested are represented as strings, but #defined constants exists to ensure correct spelling. Specific layer types may implement class specific capabilities, but this can't generally be discovered by the caller.

- **OLCRandomRead** / "RandomRead": TRUE if the **GetFeature()** (p. 122) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. 124) and **GetNextFeature()** (p. 123) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. 165) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. 173) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCFastSpatialFilter** / "FastSpatialFilter": TRUE if this layer implements spatial filtering efficiently. Layers that effectively read all features, and test them with the **OGRFeature** (p. 94) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain its own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via **GetFeatureCount()** (p. 123)) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **GetExtent()** (p. 122)) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the **SetNextByIndex()** (p. 124) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. 165), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. 166), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. 171) or **ReorderFields()** (p. 172), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. 164), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. 166) method is supported on this layer, otherwise FALSE.
- **OLCStringsAsUTF8** / "StringsAsUTF8": TRUE if values of OFTString fields are assured to be in UTF-8 format. If FALSE the encoding of fields is uncertain, though it might still be UTF-8.
- **OLCTransactions** / "Transactions": TRUE if the StartTransaction(), CommitTransaction() and RollbackTransaction() methods work in a meaningful way, otherwise FALSE.

- **OLCIgnoreFields** / "IgnoreFields": TRUE if fields, geometry and style will be omitted when fetching features as set by **SetIgnoredFields()** (p. 173) method.

This method is the same as the C function **OGR_L_TestCapability()** (p. 469).

Parameters

pszCap	the name of the capability to test.

Returns

TRUE if the layer has the requested capability, or FALSE otherwise. OGRLayers will return FALSE for any unrecognised capabilities.

Implements **OGRLayer** (p. 176).

References OGRLayer::TestCapability().

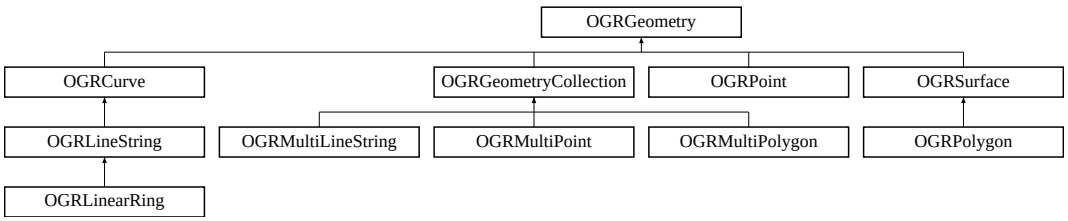
The documentation for this class was generated from the following files:

- ogr_gensql.h
- ogr_gensql.cpp

11.49 OGRGeometry Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRGeometry:



Public Member Functions

- virtual int **getDimension** () const =0

  Get the dimension of this object.
- virtual int **getCoordinateDimension** () const

  Get the dimension of the coordinates in this object.
- virtual OGRBoolean **IsEmpty** () const =0

  Returns TRUE (non-zero) if the object has no points.
- virtual OGRBoolean **IsValid** () const

  Test if the geometry is valid.
- virtual OGRBoolean **IsSimple** () const

  Test if the geometry is simple.
- virtual OGRBoolean **IsRing** () const

  Test if the geometry is a ring.
- virtual void **empty** ()=0

  Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

- virtual **OGRGeometry** * **clone** () const =0

  Make a copy of this object.
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const =0

  Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.
- virtual void **getEnvelope** (**OGREnvelope3D** *psEnvelope) const =0

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.
- virtual int **WkbSize** () const =0

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)=0

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const =0

  Convert a geometry into well known binary format.
- virtual OGRErr **importFromWkt** (char **ppszInput)=0

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **ppszDstText) const =0

  Convert a geometry into well known text format.
- virtual **OGRwkbGeometryType** **getGeometryType** () const =0

  Fetch geometry type.
- virtual const char * **getGeometryName** () const =0

  Fetch WKT name for geometry type.
- virtual void **dumpReadable** (FILE *, const char *=NULL, char **papszOptions=NULL) const

  Dump geometry in well known text format to indicated output file.
- virtual void **flattenTo2D** ()=0

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.
- virtual char * **exportToGML** (const char *const *papszOptions=NULL) const

  Convert a geometry into GML format.
- virtual char * **exportToKML** () const

  Convert a geometry into KML format.
- virtual char * **exportToJson** () const

  Convert a geometry into GeoJSON format.
- virtual void **closeRings** ()

  Force rings to be closed.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- void **assignSpatialReference** (**OGRSpatialReference** *poSR)

  Assign spatial reference to this object.
- **OGRSpatialReference** * **getSpatialReference** (void) const

  Returns spatial reference system for object.
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)=0

  Apply arbitrary coordinate transformation to geometry.
- OGRErr **transformTo** (**OGRSpatialReference** *poSR)

  Transform geometry to new spatial reference system.
- virtual void **segmentize** (double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.
- virtual OGRBoolean **Intersects** (**OGRGeometry** *) const

  Do these features intersect?
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const =0

  Returns TRUE if two geometries are equivalent.
- virtual OGRBoolean **Disjoint** (const **OGRGeometry** *) const

  Test for disjointness.
- virtual OGRBoolean **Touches** (const **OGRGeometry** *) const

Test for touching.

- virtual OGRBoolean **Crosses** (const **OGRGeometry** *) const

  Test for crossing.

- virtual OGRBoolean **Within** (const **OGRGeometry** *) const

  Test for containment.

- virtual OGRBoolean **Contains** (const **OGRGeometry** *) const

  Test for containment.

- virtual OGRBoolean **Overlaps** (const **OGRGeometry** *) const

  Test for overlap.

- virtual **OGRGeometry** * **Boundary** () const

  Compute boundary.

- virtual double **Distance** (const **OGRGeometry** *) const

  Compute distance between two geometries.

- virtual **OGRGeometry** * **ConvexHull** () const

  Compute convex hull.

- virtual **OGRGeometry** * **Buffer** (double dfDist, int nQuadSegs=30) const

  Compute buffer of geometry.

- virtual **OGRGeometry** * **Intersection** (const **OGRGeometry** *) const

  Compute intersection.

- virtual **OGRGeometry** * **Union** (const **OGRGeometry** *) const

  Compute union.

- virtual **OGRGeometry** * **UnionCascaded** () const

  Compute union using cascading.

- virtual **OGRGeometry** * **Difference** (const **OGRGeometry** *) const

  Compute difference.

- virtual **OGRGeometry** * **SymDifference** (const **OGRGeometry** *) const

  Compute symmetric difference.

- virtual OGRErr **Centroid** (**OGRPoint** *poPoint) const

  Compute the geometry centroid.

- virtual **OGRGeometry** * **Simplify** (double dTolerance) const

  Simplify the geometry.

- **OGRGeometry** * **SimplifyPreserveTopology** (double dTolerance) const

  Simplify the geometry while preserving topology.

- virtual **OGRGeometry** * **Polygonize** () const

  Polygonizes a set of sparse edges.

- virtual **OGRGeometry** * **SymmetricDifference** (const **OGRGeometry** *) const

  Compute symmetric difference (deprecated)

- virtual **OGRGeometry** * **getBoundary** () const

  Compute boundary (deprecated)

- virtual void **swapXY** ()

  Swap x and y coordinates.

11.49.1 Detailed Description

Abstract base class for all geometry classes.

Some spatial analysis methods require that OGR is built on the GEOS library to work properly. The precise meaning of methods that describe spatial relationships between geometries is described in the SFCOM, or other simple features interface specifications, like "OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture" (`OGC 06-103r3`)

11.49.2 Member Function Documentation

11.49.2.1 void OGRGeometry::assignSpatialReference (OGRSpatialReference * *poSR*)

Assign spatial reference to this object.

Any existing spatial reference is replaced, but under no circumstances does this result in the object being reprojected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. 230), but does not copy it.

This is similar to the SFCOM IGeometry::put_SpatialReference() method.

This method is the same as the C function **OGR_G_AssignSpatialReference()** (p. 429).

Parameters

poSR	new spatial reference system to apply.

References OGRSpatialReference::Reference(), and OGRSpatialReference::Release().

Referenced by OGRPoint::clone(), OGRLineString::clone(), OGRLinearRing::clone(), OGRPolygon::clone(), OGRGeometryCollection::clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRPoint::transform(), OGRLineString::transform(), OGRPolygon::transform(), and OGRGeometryCollection::transform().

11.49.2.2 OGRGeometry * OGRGeometry::Boundary ( ) const [virtual]

Compute boundary.

A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.

This method is the same as the C function **OGR_G_Boundary()** (p. 429).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Returns

a newly allocated geometry now owned by the caller, or NULL on failure.

Since

OGR 1.8.0

Referenced by getBoundary().

11.49.2.3 OGRGeometry * OGRGeometry::Buffer (double *dfDist,* int *nQuadSegs* = 30) const [virtual]

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The nQuadSegs parameter can be used to control how many segements should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This method is the same as the C function **OGR_G_Buffer()** (p. 429).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

dfDist	the buffer distance to be applied.
nQuadSegs	the number of segments used to approximate a 90 degree (quadrant) of curvature.

Returns

the newly created geometry, or NULL if an error occurs.

11.49.2.4 int OGRGeometry::Centroid (OGRPoint * *poPoint*) const [virtual]

Compute the geometry centroid.

The centroid location is applied to the passed in **OGRPoint** (p. 203) object. The centroid is not necessarily within the geometry.

This method relates to the SFCOM ISurface::get_Centroid() method however the current implementation based on GEOS can operate on other geometry types such as multipoint, linestring, geometrycollection such as multipolygons. OGC SF SQL 1.1 defines the operation for surfaces (polygons). SQL/MM-Part 3 defines the operation for surfaces and multisurfaces (multipolygons).

This function is the same as the C function **OGR_G_Centroid()** (p. 430).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Returns

OGRERR_NONE on success or OGRERR_FAILURE on error.

Since

OGR 1.8.0 as a **OGRGeometry** (p. 126) method (previously was restricted to **OGRPolygon** (p. 211))

References getGeometryType(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::setX(), OGRPoint::setY(), and wkbPoint.

Referenced by OGR_G_Centroid().

11.49.2.5 OGRGeometry * OGRGeometry::clone ( ) const [pure virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implemented in **OGRMultiLineString** (p. 196), **OGRMultiPoint** (p. 199), **OGRMultiPolygon** (p. 201), **OGRGeometryCollection** (p. 148), **OGRPolygon** (p. 213), **OGRLinearRing** (p. 179), **OGRLineString** (p. 184), and **OGRPoint** (p. 205).

Referenced by OGRGeometryCollection::addGeometry(), and OGRFeature::SetGeometry().

11.49.2.6 void OGRGeometry::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented in **OGRGeometryCollection** (p. 148), **OGRPolygon** (p. 214), and **OGRLinearRing** (p. 179).

11.49.2.7 OGRBoolean OGRGeometry::Contains (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for containment.

Tests if actual geometry object contains the passed geometry.

This method is the same as the C function **OGR_G_Contains()** (p. 431).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

11.49.2.8 OGRGeometry * OGRGeometry::ConvexHull ( ) const [virtual]

Compute convex hull.

A new geometry object is created and returned containing the convex hull of the geometry on which the method is invoked.

This method is the same as the C function **OGR_G_ConvexHull()** (p. 431).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Returns

a newly allocated geometry now owned by the caller, or NULL on failure.

11.49.2.9 OGRBoolean OGRGeometry::Crosses (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for crossing.

Tests if this geometry and the other passed into the method are crossing.

This method is the same as the C function **OGR_G_Crosses()** (p. 433).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are crossing, otherwise FALSE.

11.49.2.10 OGRGeometry * OGRGeometry::Difference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the second geometry removed.

This method is the same as the C function **OGR_G_Difference()** (p. 434).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the other geometry removed from "this" geometry.

Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

11.49.2.11 OGRBoolean OGRGeometry::Disjoint (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for disjointness.

Tests if this geometry and the other passed into the method are disjoint.

This method is the same as the C function **OGR_G_Disjoint()** (p. 434).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are disjoint, otherwise FALSE.

11.49.2.12 double OGRGeometry::Distance (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute distance between two geometries.

Returns the shortest distance between the two geometries.

This method is the same as the C function **OGR_G_Distance()** (p. 435).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the other geometry to compare against.

Returns

the distance between the geometries or -1 if an error occurs.

11.49.2.13 void OGRGeometry::dumpReadable (FILE * *fp*, const char * *pszPrefix* = NULL, char ** *papszOptions* = NULL) const [virtual]

Dump geometry in well known text format to indicated output file.

A few options can be defined to change the default dump :

- DISPLAY_GEOMETRY=NO : to hide the dump of the geometry

- DISPLAY_GEOMETRY=WKT or YES (default) : dump the geometry as a WKT
- DISPLAY_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR_G_DumpReadable()** (p. 435).

Parameters

fp	the text file to write the geometry to.
pszPrefix	the prefix to put on each line of output.
papszOptions	NULL terminated list of options (may be NULL)

References dumpReadable(), exportToWkt(), OGRPolygon::getExteriorRing(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRLineString::getNumPoints(), wkbGeometryCollection, wkbGeometryCollection25D, wkbLinearRing, wkbLineString, wkbLineString25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by dumpReadable(), and OGRFeature::DumpReadable().

11.49.2.14 void OGRGeometry::empty ( ) [pure virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 435).

Implemented in **OGRGeometryCollection** (p. 149), **OGRPolygon** (p. 214), **OGRLineString** (p. 184), and **OGRPoint** (p. 205).

11.49.2.15 int OGRGeometry::Equals (OGRGeometry * *poOtherGeom*) const [pure virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR_G_Equals()** (p. 436).

Returns

TRUE if equivalent or FALSE otherwise.

Implemented in **OGRGeometryCollection** (p. 149), **OGRPolygon** (p. 214), **OGRLineString** (p. 184), and **OGRPoint** (p. 205).

Referenced by OGRFeature::Equal(), and OGRGeometryCollection::Equals().

11.49.2.16 char * OGRGeometry::exportToGML (const char *const * *papszOptions* = NULL) const [virtual]

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

The supported options in OGR 1.8.0 are :

- FORMAT=GML3. Otherwise it will default to GML 2.1.2 output.
- GML3_LINESTRING_ELEMENT=curve. (Only valid for FORMAT=GML3) To use gml:Curve element for linestrings. Otherwise gml:LineString will be used .

- GML3_LONGSRS=YES/NO. (Only valid for FORMAT=GML3) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.

This method is the same as the C function **OGR_G_ExportToGMLEx()** (p. 436).

Parameters

papszOptions	NULL-terminated list of options.

Returns

A GML fragment or NULL in case of error.

11.49.2.17 char * OGRGeometry::exportToJson ( ) const [virtual]

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C function **OGR_G_ExportToJson()** (p. 437).

Returns

A GeoJSON fragment or NULL in case of error.

References OGR_G_ExportToJson().

11.49.2.18 char * OGRGeometry::exportToKML ( ) const [virtual]

Convert a geometry into KML format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C function **OGR_G_ExportToKML()** (p. 438).

Returns

A KML fragment or NULL in case of error.

References OGR_G_ExportToKML().

11.49.2.19 OGRErr OGRGeometry::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [pure virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implemented in **OGRGeometryCollection** (p. 149), **OGRPolygon** (p. 214), **OGRLinearRing** (p. 179), **OGRLineString** (p. 185), and **OGRPoint** (p. 205).

Referenced by OGRGeometryCollection::exportToWkb().

11.49.2.20 OGRErr OGRGeometry::exportToWkt (char ** *ppszDstText*) const [pure virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Implemented in **OGRMultiLineString** (p. 196), **OGRMultiPoint** (p. 199), **OGRMultiPolygon** (p. 202), **OGRGeometryCollection** (p. 149), **OGRPolygon** (p. 215), **OGRLineString** (p. 185), and **OGRPoint** (p. 206).

Referenced by dumpReadable(), OGRGeometryCollection::exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiLineString::exportToWkt(), and OGRFeature::GetFieldAsString().

11.49.2.21 void OGRGeometry::flattenTo2D ( ) [pure virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 439).

Implemented in **OGRGeometryCollection** (p. 150), **OGRPolygon** (p. 215), **OGRLineString** (p. 185), and **OGRPoint** (p. 206).

11.49.2.22 OGRGeometry * OGRGeometry::getBoundary ( ) const [virtual]

Compute boundary (deprecated)

Deprecated

See Also

Boundary() (p. 129)

References Boundary().

11.49.2.23 int OGRGeometry::getCoordinateDimension ( ) const [virtual]

Get the dimension of the coordinates in this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method.

This method is the same as the C function **OGR_G_GetCoordinateDimension()** (p. 441).

Returns

in practice this will return 2 or 3. It can also return 0 in the case of an empty point.

Referenced by OGRGeometryCollection::addGeometryDirectly(), OGRLineString::addPoint(), OGRPolygon::addRing(), OGRPolygon::addRingDirectly(), OGRLineString::clone(), OGRLinearRing::closeRings(), OGRLineString::exportToWkb(), OGRPolygon::exportToWkb(), OGRLineString::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRLineString::getGeometryType(), OGRPolygon::getGeometryType(), OGRGeometryCollection::getGeometryType(), OGRMultiPolygon::getGeometryType(), OGRMultiPoint::getGeometryType(), OGRMultiLineString::getGeometryType(), OGRLineString::getPoint(), OGRLineString::segmentize(), OGRLineString::setNumPoints(), OGRLineString::setPoint(), OGRLineString::setPoints(), OGRLineString::Value(), OGRLineString::WkbSize(), and OGRPolygon::WkbSize().

11.49.2.24 int OGRGeometry::getDimension ( ) const [pure virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 135)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 441).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implemented in **OGRGeometryCollection** (p. 151), **OGRPolygon** (p. 215), **OGRLineString** (p. 186), and **OGRPoint** (p. 206).

11.49.2.25 void OGRGeometry::getEnvelope (OGREnvelope * *psEnvelope*) const [pure virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Implemented in **OGRGeometryCollection** (p. 151), **OGRPolygon** (p. 216), **OGRLineString** (p. 186), and **OGRPoint** (p. 206).

Referenced by OGRGeometryCollection::getEnvelope(), OGRLayer::GetExtent(), Intersects(), and OGRGeometryFactory::organizePolygons().

11.49.2.26 void OGRGeometry::getEnvelope (OGREnvelope3D * *psEnvelope*) const [pure virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope3D()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implemented in **OGRGeometryCollection** (p. 151), **OGRPolygon** (p. 216), **OGRLineString** (p. 186), and **OGRPoint** (p. 207).

11.49.2.27 const char * OGRGeometry::getGeometryName ( ) const [pure virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 199), **OGRMultiPolygon** (p. 202), **OGRGeometryCollection** (p. 151), **OGRPolygon** (p. 216), **OGRLinearRing** (p. 180), **OGRLineString** (p. 187), and **OGRPoint** (p. 207).

Referenced by dumpReadable(), and OGRFeature::GetFieldAsString().

11.49.2.28 OGRwkbGeometryType OGRGeometry::getGeometryType ( ) const [pure virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Implemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 200), **OGRMultiPolygon** (p. 203), **OGRGeometryCollection** (p. 152), **OGRPolygon** (p. 217), **OGRLineString** (p. 187), and **OGRPoint** (p. 207).

Referenced by OGRMultiPolygon::addGeometryDirectly(), OGRMultiPoint::addGeometryDirectly(), OGRMultiLineString::addGeometryDirectly(), Centroid(), dumpReadable(), OGRPoint::Equals(), OGRLineString::Equals(), OGRPolygon::Equals(), OGRGeometryCollection::Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get_Area(), OGRGeometryCollection::get_Length(), OGRBuildPolygonFromEdges(), OGRPolygon::PointOnSurface(), and Polygonize().

11.49.2.29 OGRSpatialReference * OGRGeometry::getSpatialReference (void) const [inline]

Returns spatial reference system for object.

This method relates to the SFCOM IGeometry::get_SpatialReference() method.

This method is the same as the C function **OGR_G_GetSpatialReference()** (p. 445).

Returns

a reference to the spatial reference object. The object may be shared with many geometry objects, and should not be modified.

Referenced by OGRPoint::clone(), OGRLineString::clone(), OGRLinearRing::clone(), OGRPolygon::clone(), OGRGeometryCollection::clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), and transformTo().

11.49.2.30 OGRErr OGRGeometry::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1) [pure virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implemented in **OGRGeometryCollection** (p. 153), **OGRPolygon** (p. 218), **OGRLinearRing** (p. 180), **OGRLineString** (p. 189), and **OGRPoint** (p. 208).

Referenced by OGRGeometryFactory::createFromWkb().

11.49.2.31 OGRErr OGRGeometry::importFromWkt (char ** *ppszInput*) [pure virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 200), **OGRMultiPolygon** (p. 203), **OGRGeometryCollection** (p. 153), **OGRPolygon** (p. 218), **OGRLineString** (p. 190), and **OGRPoint** (p. 209).

Referenced by OGRGeometryFactory::createFromWkt().

11.49.2.32 OGRGeometry * OGRGeometry::Intersection (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **Intersects()** (p. 139) method can be used to test if two geometries intersect.

This method is the same as the C function **OGR_G_Intersection()** (p. 447).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the other geometry intersected with "this" geometry.

Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

11.49.2.33 OGRBoolean OGRGeometry::Intersects (OGRGeometry * *poOtherGeom*) const [virtual]

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigerous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two features overlap.

The poOtherGeom argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR_G_Intersects()** (p. 447).

Parameters

poOtherGeom	the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

References getEnvelope().

11.49.2.34 OGRBoolean OGRGeometry::IsEmpty ( ) const [pure virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns

TRUE if object is empty, otherwise FALSE.

Implemented in **OGRGeometryCollection** (p. 153), **OGRPolygon** (p. 218), **OGRLineString** (p. 190), and **OGRPoint** (p. 209).

Referenced by OGRLayer::GetExtent().

11.49.2.35 OGRBoolean OGRGeometry::IsRing ( ) const [virtual]

Test if the geometry is a ring.

This method is the same as the C function **OGR_G_IsRing()** (p. 448).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

Returns

TRUE if the geometry has no points, otherwise FALSE.

11.49.2.36 OGRBoolean OGRGeometry::IsSimple ( ) const [virtual]

Test if the geometry is simple.

This method is the same as the C function **OGR_G_IsSimple()** (p. 448).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

Returns

TRUE if the geometry has no points, otherwise FALSE.

11.49.2.37 OGRBoolean OGRGeometry::IsValid ( ) const [virtual]

Test if the geometry is valid.

This method is the same as the C function **OGR_G_IsValid()** (p. 449).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

Returns

TRUE if the geometry has no points, otherwise FALSE.

11.49.2.38 OGRBoolean OGRGeometry::Overlaps (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for overlap.

Tests if this geometry and the other passed into the method overlap, that is their intersection has a non-zero area.

This method is the same as the C function **OGR_G_Overlaps()** (p. 449).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are overlapping, otherwise FALSE.

Referenced by OGRGeometryFactory::organizePolygons().

11.49.2.39 OGRGeometry * OGRGeometry::Polygonize ( ) const [virtual]

Polygonizes a set of sparse edges.

A new geometry object is created and returned containing a collection of reassembled Polygons: NULL will be returned if the input collection doesn't corresponds to a MultiLinestring, or when reassembling Edges into Polygons is impossible due to topogical inconsistencies.

This method is the same as the C function **OGR_G_Polygonize()** (p. 450).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Returns

a newly allocated geometry now owned by the caller, or NULL on failure.

Since

OGR 1.9.0

References OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRGeometryCollection::getNumGeometries(), wkbGeometryCollection, wkbLineString, and wkbMultiLineString.

11.49.2.40 void OGRGeometry::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 451)

Parameters

dfMaxLength	the maximum distance between 2 points after segmentization

Reimplemented in **OGRGeometryCollection** (p. 154), **OGRPolygon** (p. 219), and **OGRLineString** (p. 191).

11.49.2.41 void OGRGeometry::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented in **OGRGeometryCollection** (p. 154), **OGRPolygon** (p. 219), **OGRLineString** (p. 191), and **OGRPoint** (p. 209).

Referenced by OGRGeometryCollection::setCoordinateDimension().

11.49.2.42 OGRGeometry * OGRGeometry::Simplify (double *dTolerance*) const [virtual]

Simplify the geometry.

This function is the same as the C function **OGR_G_Simplify()** (p. 452).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.8.0

11.49.2.43 OGRGeometry * OGRGeometry::SimplifyPreserveTopology (double *dTolerance*) const

Simplify the geometry while preserving topology.

This function is the same as the C function **OGR_G_SimplifyPreserveTopology()** (p. 452).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.9.0

11.49.2.44 void OGRGeometry::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented in **OGRGeometryCollection** (p. 155), **OGRPolygon** (p. 219), **OGRLineString** (p. 193), and **OGRPoint** (p. 210).

11.49.2.45 OGRGeometry * OGRGeometry::SymDifference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute symmetric difference.

Generates a new geometry which is the symmetric difference of this geometry and the second geometry passed into the method.

This method is the same as the C function **OGR_G_SymDifference()** (p. 453).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the other geometry.

Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

Since

OGR 1.8.0

Referenced by SymmetricDifference().

11.49.2.46 OGRGeometry * OGRGeometry::SymmetricDifference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute symmetric difference (deprecated)

Deprecated

See Also

OGRGeometry::SymDifference() (p. 142)

References SymDifference().

11.49.2.47 OGRBoolean OGRGeometry::Touches (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for touching.

Tests if this geometry and the other passed into the method are touching.

This method is the same as the C function **OGR_G_Touches()** (p. 453).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are touching, otherwise FALSE.

11.49.2.48 OGRErr OGRGeometry::transform (OGRCoordinateTransformation * *poCT*) [pure virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation**

(p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 454).

Parameters

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implemented in **OGRGeometryCollection** (p. 155), **OGRPolygon** (p. 220), **OGRLineString** (p. 193), and **OGRPoint** (p. 210).

Referenced by OGRGeometryCollection::transform(), and transformTo().

11.49.2.49 OGRErr OGRGeometry::transformTo (OGRSpatialReference * *poSR*)

Transform geometry to new spatial reference system.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

This method will only work if the geometry already has an assigned spatial reference system, and if it is transformable to the target coordinate system.

Because this method requires internal creation and initialization of an **OGRCoordinateTransformation** (p. 81) object it is significantly more expensive to use this method to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 81) in advance, and call **transform()** (p. 143) with that transformation. This method exists primarily for convenience when only transforming a single geometry.

This method is the same as the C function **OGR_G_TransformTo()** (p. 454).

Parameters

poSR	spatial reference system to transform to.

Returns

OGRERR_NONE on success, or an error code.

References getSpatialReference(), OGRCreateCoordinateTransformation(), and transform().

11.49.2.50 OGRGeometry * OGRGeometry::Union (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

This method is the same as the C function **OGR_G_Union()** (p. 455).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the other geometry unioned with "this" geometry.

Returns

a new geometry representing the union or NULL if an error occurs.

11.49.2.51 OGRGeometry * OGRGeometry::UnionCascaded ( ) const [virtual]

Compute union using cascading.

This method is the same as the C function **OGR_G_UnionCascaded()** (p. 455).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Returns

a new geometry representing the union or NULL if an error occurs.

Since

OGR 1.8.0

11.49.2.52 OGRBoolean OGRGeometry::Within (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for containment.

Tests if actual geometry object is within the passed geometry.

This method is the same as the C function **OGR_G_Within()** (p. 455).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

11.49.2.53 int OGRGeometry::WkbSize ( ) const [pure virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Implemented in **OGRGeometryCollection** (p. 155), **OGRPolygon** (p. 220), **OGRLinearRing** (p. 180), **OGRLineString** (p. 194), and **OGRPoint** (p. 211).

Referenced by OGRGeometryCollection::exportToWkb(), and OGRGeometryCollection::WkbSize().

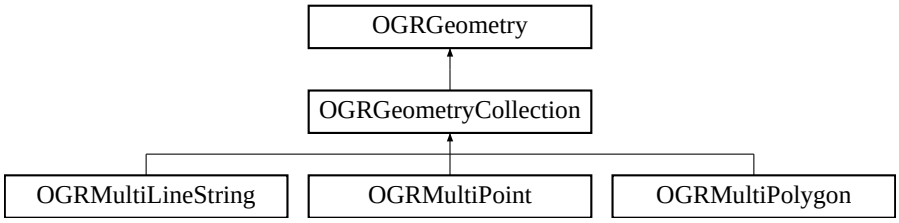
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometry.cpp

11.50 OGRGeometryCollection Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRGeometryCollection:



Public Member Functions

- **OGRGeometryCollection** ()

  Create an empty geometry collection.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual void **empty** ()

  Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)

  Apply arbitrary coordinate transformation to geometry.
- virtual void **flattenTo2D** ()

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.
- virtual OGRBoolean **IsEmpty** () const

  Returns TRUE (non-zero) if the object has no points.
- virtual void **segmentize** (double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const

  Convert a geometry into well known binary format.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **ppszDstText) const

  Convert a geometry into well known text format.
- virtual double **get_Length** () const

  Compute the length of a multicurve.
- virtual double **get_Area** () const

Compute area of geometry collection.

- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const

  Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.
- virtual void **getEnvelope** (**OGREnvelope3D** *psEnvelope) const

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.
- int **getNumGeometries** () const

  Fetch number of geometries in container.
- **OGRGeometry** * **getGeometryRef** (int)

  Fetch geometry from container.
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- virtual OGRErr **addGeometry** (const **OGRGeometry** *)

  Add a geometry to the container.
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

  Add a geometry directly to the container.
- virtual OGRErr **removeGeometry** (int iIndex, int bDelete=TRUE)

  Remove a geometry from the container.
- void **closeRings** ()

  Force rings to be closed.
- virtual void **swapXY** ()

  Swap x and y coordinates.

11.50.1 Detailed Description

A collection of 1 or more geometry objects.

All geometries must share a common spatial reference system, and Subclasses may impose additional restrictions on the contents.

11.50.2 Member Function Documentation

11.50.2.1 OGRErr OGRGeometryCollection::addGeometry (const OGRGeometry * *poNewGeom*) [virtual]

Add a geometry to the container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_AddGeometry()** (p. 426).

Parameters

poNewGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

References addGeometryDirectly(), and OGRGeometry::clone().

Referenced by clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), and OGRMultiLineString::clone().

11.50.2.2 OGRErr OGRGeometryCollection::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. 147) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 426).

There is no SFCOM analog to this method.

Parameters

poNewGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented in **OGRMultiLineString** (p. 195), **OGRMultiPoint** (p. 198), and **OGRMultiPolygon** (p. 201).

References OGRGeometry::getCoordinateDimension().

Referenced by addGeometry().

11.50.2.3 OGRGeometry * OGRGeometryCollection::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 130).

Reimplemented in **OGRMultiLineString** (p. 196), **OGRMultiPoint** (p. 199), and **OGRMultiPolygon** (p. 201).

References addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRGeometryCollection().

11.50.2.4 void OGRGeometryCollection::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 130).

References getGeometryType(), and wkbPolygon.

11.50.2.5 void OGRGeometryCollection::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 435).

Implements **OGRGeometry** (p. 133).

Referenced by OGRMultiPolygon::importFromWkt(), OGRMultiPoint::importFromWkt(), and OGRMultiLineString-::importFromWkt().

11.50.2.6 OGRBoolean OGRGeometryCollection::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR_G_Equals()** (p. 436).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 133).

References OGRGeometry::Equals(), getGeometryRef(), OGRGeometry::getGeometryType(), getGeometryType(), and getNumGeometries().

11.50.2.7 OGRErr OGRGeometryCollection::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR-Geometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

References OGRGeometry::exportToWkb(), getGeometryType(), and OGRGeometry::WkbSize().

11.50.2.8 OGRErr OGRGeometryCollection::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 135).

Reimplemented in **OGRMultiLineString** (p. 196), **OGRMultiPoint** (p. 199), and **OGRMultiPolygon** (p. 202).

References OGRGeometry::exportToWkt(), getGeometryName(), and getNumGeometries().

11.50.2.9 void OGRGeometryCollection::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 439).

Implements **OGRGeometry** (p. 135).

11.50.2.10 double OGRGeometryCollection::get_Area ( ) const [virtual]

Compute area of geometry collection.

The area is computed as the sum of the areas of all members in this collection.

Note

No warning will be issued if a member of the collection does not support the get_Area method.

Returns

computed area.

Reimplemented in **OGRMultiPolygon** (p. 202).

References getGeometryName(), OGRGeometry::getGeometryType(), wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiPolygon, and wkbPolygon.

11.50.2.11 double OGRGeometryCollection::get_Length ( ) const [virtual]

Compute the length of a multicurve.

The length is computed as the sum of the length of all members in this collection.

Note

No warning will be issued if a member of the collection does not support the get_Length method.

Returns

computed area.

References OGRGeometry::getGeometryType(), wkbGeometryCollection, wkbLinearRing, and wkbLineString.

11.50.2.12 int OGRGeometryCollection::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 135)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 441).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 136).

11.50.2.13 void OGRGeometryCollection::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. 136).

References OGRGeometry::getEnvelope(), and IsEmpty().

11.50.2.14 void OGRGeometryCollection::getEnvelope (OGREnvelope3D * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope3D()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. 136).

References OGRGeometry::getEnvelope(), and IsEmpty().

11.50.2.15 const char * OGRGeometryCollection::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. 137).

Reimplemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 199), and **OGRMultiPolygon** (p. 202).

Referenced by exportToWkt(), and get_Area().

11.50.2.16 OGRGeometry * OGRGeometryCollection::getGeometryRef (int *i*)

Fetch geometry from container.

This method returns a pointer to an geometry within the container. The returned geometry remains owned by the container, and should not be modified. The pointer is only valid untill the next change to the geometry container. Use IGeometry::clone() to make a copy.

This method relates to the SFCOM IGeometryCollection::get_Geometry() method.

Parameters

i	the index of the geometry to fetch, between 0 and **getNumGeometries()** (p. 152) - 1.

Returns

pointer to requested geometry.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get_Area(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

11.50.2.17 OGRwkbGeometryType OGRGeometryCollection::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 137).

Reimplemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 200), and **OGRMultiPolygon** (p. 203).

References OGRGeometry::getCoordinateDimension(), wkbGeometryCollection, and wkbGeometryCollection25D.

Referenced by closeRings(), Equals(), and exportToWkb().

11.50.2.18 int OGRGeometryCollection::getNumGeometries ( ) const

Fetch number of geometries in container.

This method relates to the SFCOM IGeometryCollect::get_NumGeometries() method.

Returns

count of children geometries. May be zero.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get_Area(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

11.50.2.19 OGRErr OGRGeometryCollection::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

11.50.2.20 OGRErr OGRGeometryCollection::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

Reimplemented in **OGRMultiLineString** (p. 197), **OGRMultiPoint** (p. 200), and **OGRMultiPolygon** (p. 203).

11.50.2.21 OGRBoolean OGRGeometryCollection::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 139).

Referenced by OGRMultiPoint::exportToWkt(), and getEnvelope().

11.50.2.22 OGRErr OGRGeometryCollection::removeGeometry (int *iGeom,* int *bDelete* = TRUE) [virtual]

Remove a geometry from the container.

Removing a geometry will cause the geometry count to drop by one, and all "higher" geometries will shuffle down one in index.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_RemoveGeometry()** (p. 450).

Parameters

iGeom	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
bDelete	if TRUE the geometry will be deallocated, otherwise it will not. The default is TRUE as the container is considered to own the geometries in it.

Returns

OGRERR_NONE if successful, or OGRERR_FAILURE if the index is out of range.

Referenced by OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), and OGRGeometryFactory::forceToMultiPolygon().

11.50.2.23 void OGRGeometryCollection::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 451)

Parameters

dfMaxLength	the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 141).

11.50.2.24 void OGRGeometryCollection::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 141).

References OGRGeometry::setCoordinateDimension().

11.50.2.25 void OGRGeometryCollection::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. 142).

11.50.2.26 OGRErr OGRGeometryCollection::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 454).

Parameters

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 143).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRGeometry::transform().

11.50.2.27 int OGRGeometryCollection::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 145).

References OGRGeometry::WkbSize().

The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometrycollection.cpp

11.51 OGRGeometryFactory Class Reference

`#include <ogr_geometry.h>`

Static Public Member Functions

- static OGRErr **createFromWkb** (unsigned char *, **OGRSpatialReference** *, **OGRGeometry** **, int=-1)

  Create a geometry object of the appropriate type from it's well known binary representation.
- static OGRErr **createFromWkt** (char **, **OGRSpatialReference** *, **OGRGeometry** **)

  Create a geometry object of the appropriate type from it's well known text representation.
- static OGRErr **createFromFgf** (unsigned char *, **OGRSpatialReference** *, **OGRGeometry** **, int=-1, int *=NULL)

  Create a geometry object of the appropriate type from it's FGF (FDO Geometry Format) binary representation.
- static **OGRGeometry** * **createFromGML** (const char *)

  Create geometry from GML.
- static void **destroyGeometry** (**OGRGeometry** *)

  Destroy geometry object.
- static **OGRGeometry** * **createGeometry** (**OGRwkbGeometryType**)

  Create an empty geometry of desired type.
- static **OGRGeometry** * **forceToPolygon** (**OGRGeometry** *)

  Convert to polygon.
- static **OGRGeometry** * **forceToMultiPolygon** (**OGRGeometry** *)

  Convert to multipolygon.
- static **OGRGeometry** * **forceToMultiPoint** (**OGRGeometry** *)

  Convert to multipoint.
- static **OGRGeometry** * **forceToMultiLineString** (**OGRGeometry** *)

  Convert to multilinestring.
- static **OGRGeometry** * **organizePolygons** (**OGRGeometry** **papoPolygons, int nPolygonCount, int *pbResultValidGeometry, const char **papszOptions=NULL)

  Organize polygons based on geometries.
- static int **haveGEOS** ()

  Test if GEOS enabled.
- static **OGRGeometry** * **approximateArcAngles** (double dfX, double dfY, double dfZ, double dfPrimaryRadius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees)

11.51.1 Detailed Description

Create geometry objects from well known text/binary.

11.51.2 Member Function Documentation

11.51.2.1 OGRGeometry * OGRGeometryFactory::approximateArcAngles (double *dfCenterX,* double *dfCenterY,* double *dfZ,* double *dfPrimaryRadius,* double *dfSecondaryRadius,* double *dfRotation,* double *dfStartAngle,* double *dfEndAngle,* double *dfMaxAngleStepSizeDegrees*) [static]

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in degrees.

If the dfMaxAngleStepSizeDegrees is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR_ARC_STEPSIZE configuration variable.

See Also

CPLSetConfigOption() (p. 330)

Parameters

dfCenterX	center X
dfCenterY	center Y
dfZ	center Z
dfPrimaryRadius	X radius of ellipse.
dfSecondary-Radius	Y radius of ellipse.
dfRotation	rotation of the ellipse clockwise.
dfStartAngle	angle to first point on arc (clockwise of X-positive)
dfEndAngle	angle to last point on arc (clockwise of X-positive)
dfMaxAngle-StepSize-Degrees	the largest step in degrees along the arc, zero to use the default setting.

Returns

OGRLineString (p. 181) geometry representing an approximation of the arc.

Since

OGR 1.8.0

References OGRLineString::setPoint().

Referenced by OGR_G_ApproximateArcAngles().

11.51.2.2 OGRErr OGRGeometryFactory::createFromFgf (unsigned char * *pabyData,* OGRSpatialReference * *poSR,* OGRGeometry ** *ppoReturn,* int *nBytes* = -1, int * *pnBytesConsumed* = NULL) [static]

Create a geometry object of the appropriate type from it's FGF (FDO Geometry Format) binary representation.

Also note that this is a static method, and that there is no need to instantiate an **OGRGeometryFactory** (p. 156) object.

The C function OGR_G_CreateFromFgf() is the same as this method.

Parameters

pabyData	pointer to the input BLOB data.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.

ppoReturn	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
nBytes	the number of bytes available in pabyData.
pnBytes-Consumed	if not NULL, it will be set to the number of bytes consumed (at most nBytes).

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

11.51.2.3 OGRGeometry * OGRGeometryFactory::createFromGML (const char * *pszData*) [static]

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGRGeometry** (p. 126). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The following GML2 elements are parsed : Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, MultiGeometry.

(OGR >= 1.8.0) The following GML3 elements are parsed : Surface, MultiSurface, PolygonPatch, Triangle, Rectangle, Curve, MultiCurve, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the OGR_ARC_STEPSIZE configuration variable.

The C function **OGR_G_CreateFromGML()** (p. 432) is the same as this method.

Parameters

pszData	The GML fragment for the geometry.

Returns

a geometry on succes, or NULL on error.

11.51.2.4 OGRErr OGRGeometryFactory::createFromWkb (unsigned char * *pabyData,* OGRSpatialReference * *poSR,* OGRGeometry ** *ppoReturn,* int *nBytes* = -1) [static]

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGRGeometry::WkbSize()** (p. 145) on the returned geometry to establish the number of bytes it required in WKB format.

Also note that this is a static method, and that there is no need to instantiate an **OGRGeometryFactory** (p. 156) object.

The C function **OGR_G_CreateFromWkb()** (p. 432) is the same as this method.

Parameters

pabyData	pointer to the input BLOB data.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.

ppoReturn	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
nBytes	the number of bytes available in pabyData, or -1 if it isn't known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometry::assignSpatialReference(), createGeometry(), and OGRGeometry::importFromWkb().

Referenced by OGR_G_CreateFromWkb().

11.51.2.5 OGRErr OGRGeometryFactory::createFromWkt (char ** *ppszData,* OGRSpatialReference * *poSR,* OGRGeometry ** *ppoReturn*) [static]

Create a geometry object of the appropriate type from it's well known text representation.

The C function **OGR_G_CreateFromWkt()** (p. 433) is the same as this method.

Parameters

ppszData	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
ppoReturn	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL if the method fails.

Example:

```
const char* wkt= "POINT(0 0)";

// cast because OGR_G_CreateFromWkt will move the pointer
char* pszWkt = (char*) wkt;
OGRSpatialReferenceH ref = OSRNewSpatialReference(NULL);
OGRGeometryH new_geom;
OGRErr err = OGR_G_CreateFromWkt(&pszWkt, ref, &new_geom);
```

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometry::assignSpatialReference(), and OGRGeometry::importFromWkt().

Referenced by OGR_G_CreateFromWkt().

11.51.2.6 OGRGeometry * OGRGeometryFactory::createGeometry (OGRwkbGeometryType *eGeometryType*) [static]

Create an empty geometry of desired type.

This is equivelent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This method is the same as the C function **OGR_G_CreateGeometry()** (p. 433).

Parameters

eGeometryType	the type code of the geometry class to be instantiated.

Returns

the newly create geometry or NULL on failure.

References wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbPoint, and wkbPolygon.

Referenced by createFromWkb(), and OGR_G_CreateGeometry().

11.51.2.7 void OGRGeometryFactory::destroyGeometry (OGRGeometry * *poGeom*) [static]

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This method is the same as the C function **OGR_G_DestroyGeometry()** (p. 434).

Parameters

poGeom	the geometry to deallocate.

Referenced by OGR_G_DestroyGeometry().

11.51.2.8 OGRGeometry * OGRGeometryFactory::forceToMultiLineString (OGRGeometry * *poGeom*) [static]

Convert to multilinestring.

Tries to force the provided geometry to be a multilinestring.

- linestrings are placed in a multilinestring.
- geometry collections will be converted to multilinestring if they only contain linestrings.
- polygons will be changed to a collection of linestrings (one per ring).

The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns

new geometry.

References OGRMultiLineString::addGeometryDirectly(), OGRLineString::addSubLineString(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRLineString::getNumPoints(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, wkbLineString, wkbMultiLineString, wkbMultiPolygon, and wkbPolygon.

Referenced by OGR_G_ForceToMultiLineString().

11.51.2.9 OGRGeometry * OGRGeometryFactory::forceToMultiPoint (OGRGeometry * *poGeom*) [static]

Convert to multipoint.

Tries to force the provided geometry to be a multipoint. Currently this just effects a change on points. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns

new geometry.

References OGRMultiPoint::addGeometryDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, wkbMultiPoint, and wkbPoint.

Referenced by OGR_G_ForceToMultiPoint().

11.51.2.10 OGRGeometry * OGRGeometryFactory::forceToMultiPolygon (OGRGeometry * *poGeom*) [static]

Convert to multipolygon.

Tries to force the provided geometry to be a multipolygon. Currently this just effects a change on polygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns

new geometry.

References OGRMultiPolygon::addGeometryDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, wkbMultiPolygon, and wkbPolygon.

Referenced by OGR_G_ForceToMultiPolygon().

11.51.2.11 OGRGeometry * OGRGeometryFactory::forceToPolygon (OGRGeometry * *poGeom*) [static]

Convert to polygon.

Tries to force the provided geometry to be a polygon. Currently this just effects a change on multipolygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

Parameters

poGeom	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRPolygon::addRing(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), wkbGeometryCollection, wkbMultiPolygon, and wkbPolygon.

Referenced by OGR_G_ForceToPolygon().

11.51.2.12 int OGRGeometryFactory::haveGEOS ( ) [static]

Test if GEOS enabled.

This static method returns TRUE if GEOS support is built into OGR, otherwise it returns FALSE.

Returns

TRUE if available, otherwise FALSE.

Referenced by organizePolygons().

11.51.2.13 OGRGeometry * OGRGeometryFactory::organizePolygons (OGRGeometry ** *papoPolygons*, int *nPolygonCount*, int * *pbIsValidGeometry*, const char ** *papszOptions* = NULL) [static]

Organize polygons based on geometries.

Analyse a set of rings (passed as simple polygons), and based on a geometric analysis convert them into a polygon with inner rings, or a MultiPolygon if dealing with more than one polygon.

All the input geometries must be OGRPolygons with only a valid exterior ring (at least 4 points) and no interior rings.

The passed in geometries become the responsibility of the method, but the papoPolygons "pointer array" remains owned by the caller.

For faster computation, a polygon is considered to be inside another one if a single point of its external ring is included into the other one. (unless 'OGR_DEBUG_ORGANIZE_POLYGONS' configuration option is set to TRUE. In that case, a slower algorithm that tests exact topological relationships is used if GEOS is available.)

In cases where a big number of polygons is passed to this function, the default processing may be really slow. You can skip the processing by adding METHOD=SKIP to the option list (the result of the function will be a multi-polygon with all polygons as toplevel polygons) or only make it analyze counterclockwise polygons by adding METHOD=ONLY_CCW to the option list if you can assume that the outline of holes is counterclockwise defined (this is the convention for shapefiles e.g.)

If the OGR_ORGANIZE_POLYGONS configuration option is defined, its value will override the value of the METHOD option of papszOptions (usefull to modify the behaviour of the shapefile driver)

Parameters

papoPolygons	array of geometry pointers - should all be OGRPolygons. Ownership of the geometries is passed, but not of the array itself.
nPolygonCount	number of items in papoPolygons
pbIsValidGeometry	value will be set TRUE if result is valid or FALSE otherwise.
papszOptions	a list of strings for passing options

Returns

a single resulting geometry (either **OGRPolygon** (p. 211) or **OGRMultiPolygon** (p. 200)).

References OGRPolygon::addRing(), OGRPolygon::exportToWkt(), OGRPolygon::get_Area(), OGRGeometry::getEnvelope(), OGRPolygon::getExteriorRing(), OGRLineString::getNumPoints(), OGRLineString::getPoint(), haveGEOS(), OGRLinearRing::isClockwise(), OGRGeometry::Overlaps(), and wkbPolygon.

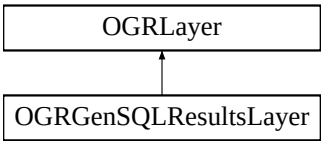
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometryfactory.cpp

11.52 OGRLayer Class Reference

`#include <ogrsf_frmts.h>`

Inheritance diagram for OGRLayer:



Public Member Functions

- virtual **OGRGeometry** * **GetSpatialFilter** ()

  This method returns the current spatial filter for this layer.
- virtual void **SetSpatialFilter** (**OGRGeometry** *)

  Set a new spatial filter.
- virtual void **SetSpatialFilterRect** (double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)

  Set a new rectangular spatial filter.
- virtual OGRErr **SetAttributeFilter** (const char *)

  Set a new attribute query.
- virtual void **ResetReading** ()=0

  Reset feature reading to start on the first feature.
- virtual **OGRFeature** * **GetNextFeature** ()=0

  Fetch the next available feature from this layer.
- virtual OGRErr **SetNextByIndex** (long nIndex)

  Move read cursor to the nIndex'th feature in the current resultset.
- virtual **OGRFeature** * **GetFeature** (long nFID)

  Fetch a feature by its identifier.
- virtual OGRErr **SetFeature** (**OGRFeature** *poFeature)

  Rewrite an existing feature.
- virtual OGRErr **CreateFeature** (**OGRFeature** *poFeature)

  Create and write a new feature within a layer.
- virtual OGRErr **DeleteFeature** (long nFID)

  Delete feature from layer.
- virtual const char * **GetName** ()

  Return the layer name.
- virtual **OGRwkbGeometryType GetGeomType** ()

  Return the layer geometry type.
- virtual **OGRFeatureDefn** * **GetLayerDefn** ()=0

  Fetch the schema information for this layer.
- virtual **OGRSpatialReference** * **GetSpatialRef** ()

  Fetch the spatial reference system for this layer.
- virtual int **GetFeatureCount** (int bForce=TRUE)

  Fetch the feature count in this layer.
- virtual OGRErr **GetExtent** (**OGREnvelope** *psExtent, int bForce=TRUE)

  Fetch the extent of this layer.
- virtual int **TestCapability** (const char *)=0

  Test if this layer supported the named capability.
- virtual const char * **GetInfo** (const char *)

  Fetch metadata from layer.
- virtual OGRErr **CreateField** (**OGRFieldDefn** *poField, int bApproxOK=TRUE)

  Create a new field on a layer.
- virtual OGRErr **DeleteField** (int iField)

  Delete an existing field on a layer.
- virtual OGRErr **ReorderFields** (int *panMap)

  Reorder all the fields of a layer.
- virtual OGRErr **AlterFieldDefn** (int iField, **OGRFieldDefn** *poNewFieldDefn, int nFlags)

  Alter the definition of an existing field on a layer.
- virtual OGRErr **SyncToDisk** ()

  Flush pending changes to disk.
- virtual **OGRStyleTable** * **GetStyleTable** ()

Returns layer style table.

- virtual void **SetStyleTableDirectly** (**OGRStyleTable** ∗poStyleTable)

  Set layer style table.
- virtual void **SetStyleTable** (**OGRStyleTable** ∗poStyleTable)

  Set layer style table.
- virtual const char ∗ **GetFIDColumn** ()

  This method returns the name of the underlying database column being used as the FID column, or "" if not supported.
- virtual const char ∗ **GetGeometryColumn** ()

  This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.
- virtual OGRErr **SetIgnoredFields** (const char ∗∗papszFields)

  Set which fields can be omitted when retrieving features from the layer.
- int **Reference** ()

  Increment layer reference count.
- int **Dereference** ()

  Decrement layer reference count.
- int **GetRefCount** () const

  Fetch reference count.
- OGRErr **ReorderField** (int iOldFieldPos, int iNewFieldPos)

  Reorder an existing field on a layer.

11.52.1 Detailed Description

This class represents a layer of simple features, with access methods.

11.52.2 Member Function Documentation

11.52.2.1 OGRErr OGRLayer::AlterFieldDefn (int *iField,* OGRFieldDefn ∗ *poNewFieldDefn,* int *nFlags*) [virtual]

Alter the definition of an existing field on a layer.

You must use this to alter the definition of an existing field of a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCAlterFieldDefn capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. 456).

Parameters

iField	index of the field whose definition must be altered.
poNewFieldDefn	new field definition
nFlags	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG and ALTER_WIDTH_PRECISION_FLAG to indicate which of the name and/or type and/or width and precision fields from the new field definition must be taken into account.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

11.52.2.2 OGRErr OGRLayer::CreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

The passed feature is written to the layer as a new feature, rather than overwriting an existing one. If the feature has a feature id other than OGRNullFID, then the native implementation may use that as the feature id of the new feature, but not necessarily. Upon successful return the passed feature will have been updated with the new feature id.

This method is the same as the C function **OGR_L_CreateFeature()** (p. 457).

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Referenced by OGRDataSource::CopyLayer().

11.52.2.3 OGRErr OGRLayer::CreateField (OGRFieldDefn * *poField,* int *bApproxOK* = TRUE) [virtual]

Create a new field on a layer.

You must use this to create new fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCCreateField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR_L_CreateField()** (p. 458).

Parameters

poField	field definition to write to disk.
bApproxOK	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

Returns

OGRERR_NONE on success.

Referenced by OGRDataSource::CopyLayer().

11.52.2.4 OGRErr OGRLayer::DeleteFeature (long *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (p. 176) layer method may be called with OLCDeleteFeature to check if the driver supports feature deletion.

This method is the same as the C function **OGR_L_DeleteFeature()** (p. 458).

Parameters

nFID	the feature id to be deleted from the layer

Returns

OGRERR_NONE on success.

11.52.2.5 OGRErr OGRLayer::DeleteField (int *iField*) [virtual]

Delete an existing field on a layer.

You must use this to delete existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the deleted field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCDeleteField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR_L_DeleteField()** (p. 459).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

11.52.2.6 int OGRLayer::Dereference ( )

Decrement layer reference count.

This method is the same as the C function OGR_L_Dereference().

Returns

the reference count after decrementing.

11.52.2.7 OGRErr OGRLayer::GetExtent (OGREnvelope * *psExtent*, int *bForce* = TRUE) [virtual]

Fetch the extent of this layer.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (p. 167) without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

This method is the same as the C function **OGR_L_GetExtent()** (p. 459).

Parameters

psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented in **OGRGenSQLResultsLayer** (p. 122).

References OGRGeometry::getEnvelope(), OGRFeature::GetGeometryRef(), GetGeomType(), GetLayerDefn(), GetNextFeature(), OGRGeometry::IsEmpty(), ResetReading(), and wkbNone.

Referenced by OGRGenSQLResultsLayer::GetExtent().

11.52.2.8 OGRFeature * OGRLayer::GetFeature (long *nFID*) [virtual]

Fetch a feature by its identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters.

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (p. 98)) will be the same as nFID.

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. 167); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads are generally considered interrupted by a **GetFeature()** (p. 167) call.

The returned feature should be free with **OGRFeature::DestroyFeature()** (p. 97).

This method is the same as the C function **OGR_L_GetFeature()** (p. 460).

Parameters

nFID	the feature id of the feature to read.

Returns

a feature now owned by the caller, or NULL on failure.

Reimplemented in **OGRGenSQLResultsLayer** (p. 122).

References OGRFeature::GetFID(), GetNextFeature(), and ResetReading().

Referenced by OGRGenSQLResultsLayer::GetFeature().

11.52.2.9 int OGRLayer::GetFeatureCount (int *bForce* = TRUE) [virtual]

Fetch the feature count in this layer.

Returns the number of features in the layer. For dynamic databases the count may not be exact. If bForce is FALSE, and it would be expensive to establish the feature count a value of -1 may be returned indicating that the count isn't know. If bForce is TRUE some implementations will actually scan the entire layer once to count objects.

The returned count takes the spatial filter into account.

Note that some implementations of this method may alter the read cursor of the layer.

This method is the same as the C function **OGR_L_GetFeatureCount()** (p. 460).

Parameters

bForce	Flag indicating whether the count should be computed even if it is expensive.

Returns

feature count, -1 if count not known.

Reimplemented in **OGRGenSQLResultsLayer** (p. 123).

References GetNextFeature(), and ResetReading().

Referenced by OGRGenSQLResultsLayer::GetFeatureCount().

11.52.2.10 const char * OGRLayer::GetFIDColumn ( ) [virtual]

This method returns the name of the underlying database column being used as the FID column, or "" if not supported.

This method is the same as the C function **OGR_L_GetFIDColumn()** (p. 461).

Returns

fid column name.

11.52.2.11 const char * OGRLayer::GetGeometryColumn ( ) [virtual]

This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.

This method is the same as the C function **OGR_L_GetGeometryColumn()** (p. 461).

Returns

geometry column name.

11.52.2.12 OGRwkbGeometryType OGRLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

This returns the same result as **GetLayerDefn()** (p. 169)->**GetGeomType()** (p. 168), but for a few drivers, calling **GetGeomType()** (p. 168) directly can avoid lengthy layer definition initialization.

This method is the same as the C function **OGR_L_GetGeomType()** (p. 461).

If this method is derived in a driver, it must be done such that it returns the same content as **GetLayerDefn()** (p. 169)->**GetGeomType()** (p. 168).

Returns

the geometry type

Since

OGR 1.8.0

References OGRFeatureDefn::GetGeomType(), and GetLayerDefn().

Referenced by GetExtent().

11.52.2.13 const char * OGRLayer::GetInfo (const char * *pszTag*) [virtual]

Fetch metadata from layer.

This method can be used to fetch various kinds of metadata or layer specific information encoded as a string. It is anticipated that various tag values will be defined with well known semantics, while other tags will be used for driver/application specific purposes.

This method is deprecated and will be replaced with a more general metadata model in the future. At this time no drivers return information via the **GetInfo()** (p. 169) call.

Parameters

pszTag	the tag for which information is being requested.

Returns

the value of the requested tag, or NULL if that tag does not have a value, or is unknown.

Deprecated

11.52.2.14 OGRFeatureDefn * OGRLayer::GetLayerDefn ( ) [pure virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. 109) is owned by the **OGRLayer** (p. 162), and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

This method is the same as the C function **OGR_L_GetLayerDefn()** (p. 462).

Returns

feature definition.

Implemented in **OGRGenSQLResultsLayer** (p. 123).

Referenced by OGRSFDriver::CopyDataSource(), OGRDataSource::CopyLayer(), OGRDataSource::ExecuteSQL(), GetExtent(), GetGeomType(), GetName(), ReorderField(), SetAttributeFilter(), and SetIgnoredFields().

11.52.2.15 const char * OGRLayer::GetName ( ) [virtual]

Return the layer name.

This returns the same content as **GetLayerDefn()** (p. 169)->**GetName()** (p. 169), but for a few drivers, calling **GetName()** (p. 169) directly can avoid lengthy layer definition initialization.

This method is the same as the C function **OGR_L_GetName()** (p. 462).

If this method is derived in a driver, it must be done such that it returns the same content as **GetLayerDefn()** (p. 169)->**GetName()** (p. 169).

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

References GetLayerDefn(), and OGRFeatureDefn::GetName().

Referenced by OGRDataSource::GetLayerByName().

11.52.2.16 OGRFeature * OGRLayer::GetNextFeature ( ) [pure virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity of the caller to delete with **OGRFeature::DestroyFeature()** (p. 97). It is critical that all features associated with an **OGRLayer** (p. 162) (more specifically an **OGRFeatureDefn** (p. 109)) be deleted before that layer/datasource is deleted.

Only features matching the current spatial filter (set with **SetSpatialFilter()** (p. 174)) will be returned.

This method implements sequential access to the features of a layer. The **ResetReading()** (p. 172) method can be used to start at the beginning again.

This method is the same as the C function **OGR_L_GetNextFeature()** (p. 462).

Returns

a feature, or NULL if no more features are available.

Implemented in **OGRGenSQLResultsLayer** (p. 123).

Referenced by OGRDataSource::CopyLayer(), GetExtent(), GetFeature(), GetFeatureCount(), OGRGenSQL-ResultsLayer::GetNextFeature(), and SetNextByIndex().

11.52.2.17 int OGRLayer::GetRefCount ( ) const

Fetch reference count.

This method is the same as the C function OGR_L_GetRefCount().

Returns

the current reference count for the layer object itself.

Referenced by OGRDataSource::GetSummaryRefCount().

11.52.2.18 OGRGeometry * OGRLayer::GetSpatialFilter ( ) [virtual]

This method returns the current spatial filter for this layer.

The returned pointer is to an internally owned object, and should not be altered or deleted by the caller.

This method is the same as the C function **OGR_L_GetSpatialFilter()** (p. 463).

Returns

spatial filter geometry.

Reimplemented in **OGRGenSQLResultsLayer** (p. 124).

11.52.2.19 OGRSpatialReference * OGRLayer::GetSpatialRef ( ) [inline],[virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 162) and should not be modified or freed by the application.

This method is the same as the C function **OGR_L_GetSpatialRef()** (p. 463).

Returns

spatial reference, or NULL if there isn't one.

Reimplemented in **OGRGenSQLResultsLayer** (p. 124).

Referenced by OGRDataSource::CopyLayer(), and OGRGenSQLResultsLayer::GetSpatialRef().

11.52.2.20 OGRStyleTable * OGRLayer::GetStyleTable ( ) [virtual]

Returns layer style table.

This method is the same as the C function OGR_L_GetStyleTable().

Returns

pointer to a style table which should not be modified or freed by the caller.

11.52.2.21 int OGRLayer::Reference ( )

Increment layer reference count.

This method is the same as the C function OGR_L_Reference().

Returns

the reference count after incrementing.

11.52.2.22 OGRErr OGRLayer::ReorderField (int *iOldFieldPos,* int *iNewFieldPos*)

Reorder an existing field on a layer.

This method is a conveniency wrapper of **ReorderFields()** (p. 172) dedicated to move a single field. It is a non-virtual method, so drivers should implement **ReorderFields()** (p. 172) instead.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

The field definition that was at initial position iOldFieldPos will be moved at position iNewFieldPos, and elements between will be shuffled accordingly.

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderField(1, 3) will reorder them as "0","2","3","1","4".

Not all drivers support this method. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR_L_ReorderField()** (p. 464).

Parameters

iOldFieldPos	previous position of the field to move. Must be in the range [0,GetFieldCount()-1].
iNewFieldPos	new position of the field to move. Must be in the range [0,GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGRFeatureDefn::GetFieldCount(), GetLayerDefn(), and ReorderFields().

11.52.2.23 OGRErr OGRLayer::ReorderFields (int * *panMap*) [virtual]

Reorder all the fields of a layer.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was panMap[i].

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderFields([0,2,3,1,4]) will reorder them as "0","2","3","1","4".

Not all drivers support this method. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR_L_ReorderFields()** (p. 464).

Parameters

panMap	an array of **GetLayerDefn()** (p. 169)->GetFieldCount() elements which is a permutation of [0, **GetLayerDefn()** (p. 169)->GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Referenced by ReorderField().

11.52.2.24 void OGRLayer::ResetReading ( ) [pure virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. 170).

This method is the same as the C function **OGR_L_ResetReading()** (p. 465).

Implemented in **OGRGenSQLResultsLayer** (p. 124).

Referenced by OGRDataSource::CopyLayer(), GetExtent(), GetFeature(), GetFeatureCount(), OGRGenSQLResultsLayer::ResetReading(), SetAttributeFilter(), SetNextByIndex(), and SetSpatialFilter().

11.52.2.25 OGRErr OGRLayer::SetAttributeFilter (const char * *pszQuery*) [virtual]

Set a new attribute query.

This method sets the attribute query string to be used when fetching features via the **GetNextFeature()** (p. 170) method. Only features for which the query evaluates as true will be returned.

The query string should be in the format of an SQL WHERE clause. For instance "population > 1000000 and population < 5000000" where population is an attribute in the layer. The query format is normally a restricted form of SQL WHERE clause as described in the "WHERE" section of the OGR SQL tutorial. In some cases (RDBMS backed drivers) the native capabilities of the database may be used to interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

Note that installing a query string will generally result in resetting the current reading position (ala **ResetReading()** (p. 172)).

This method is the same as the C function **OGR_L_SetAttributeFilter()** (p. 465).

Parameters

pszQuery	query in restricted SQL WHERE format, or NULL to clear the current query.

Returns

OGRERR_NONE if successfully installed, or an error code if the query expression is in error, or some other failure occurs.

References GetLayerDefn(), and ResetReading().

Referenced by OGRGenSQLResultsLayer::ResetReading().

11.52.2.26 OGRErr OGRLayer::SetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

This method will write a feature to the layer, based on the feature id within the **OGRFeature** (p. 94).

Use OGRLayer::TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **SetFeature()** (p. 173).

This method is the same as the C function **OGR_L_SetFeature()** (p. 466).

Parameters

poFeature	the feature to write.

Returns

OGRERR_NONE if the operation works, otherwise an appropriate error code.

11.52.2.27 OGRErr OGRLayer::SetIgnoredFields (const char ** *papszFields*) [virtual]

Set which fields can be omitted when retrieving features from the layer.

If the driver supports this functionality (testable using OLCIgnoreFields capability), it will not fetch the specified fields in subsequent calls to **GetFeature()** (p. 167) / **GetNextFeature()** (p. 170) and thus save some processing time and/or bandwidth.

Besides field names of the layers, the following special fields can be passed: "OGR_GEOMETRY" to ignore geometry and "OGR_STYLE" to ignore layer style.

By default, no fields are ignored.

This method is the same as the C function **OGR_L_SetIgnoredFields()** (p. 466)

Parameters

papszFields	an array of field names terminated by NULL item. If NULL is passed, the ignored list is cleared.

Returns

OGRERR_NONE if all field names have been resolved (even if the driver does not support this method)

References OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), GetLayerDefn(), OGRFeatureDefn::SetGeometryIgnored(), OGRFieldDefn::SetIgnored(), and OGRFeatureDefn::SetStyleIgnored().

11.52.2.28 OGRErr OGRLayer::SetNextByIndex (long *nIndex*) [virtual]

Move read cursor to the nIndex'th feature in the current resultset.

This method allows positioning of a layer such that the **GetNextFeature()** (p. 170) call will read the requested feature, where nIndex is an absolute index into the current result set. So, setting it to 3 would mean the next feature read with **GetNextFeature()** (p. 170) would have been the 4th feature to have been read if sequential reading took place from the beginning of the layer, including accounting for spatial and attribute filters.

Only in rare circumstances is **SetNextByIndex()** (p. 174) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. 172) and then calls **GetNextFeature()** (p. 170) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. 176) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR_L_SetNextByIndex()** (p. 467).

Parameters

nIndex	the index indicating how many steps into the result set to seek.

Returns

OGRERR_NONE on success or an error code.

Reimplemented in **OGRGenSQLResultsLayer** (p. 124).

References GetNextFeature(), and ResetReading().

Referenced by OGRGenSQLResultsLayer::SetNextByIndex().

11.52.2.29 void OGRLayer::SetSpatialFilter (OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (p. 170) method. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features who's envelope (as returned by **OGRGeometry::getEnvelope()** (p. 136)) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

This method makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by **OGRLayer::GetSpatialRef()** (p. 171)). In the future this may be generalized.

This method is the same as the C function **OGR_L_SetSpatialFilter()** (p. 467).

Parameters

poFilter	the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

References ResetReading().

Referenced by OGRGenSQLResultsLayer::ResetReading(), and SetSpatialFilterRect().

11.52.2.30 void OGRLayer::SetSpatialFilterRect (double *dfMinX*, double *dfMinY*, double *dfMaxX*, double *dfMaxY*) [virtual]

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **GetNextFeature()** (p. 170) method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as the layer as a whole (as returned by **OGRLayer::GetSpatialRef()** (p. 171)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. 174). It exists as a convenience.

The only way to clear a spatial filter set with this method is to call OGRLayer::SetSpatialFilter(NULL).

This method is the same as the C function **OGR_L_SetSpatialFilterRect()** (p. 468).

Parameters

dfMinX	the minimum X coordinate for the rectangular region.
dfMinY	the minimum Y coordinate for the rectangular region.
dfMaxX	the maximum X coordinate for the rectangular region.
dfMaxY	the maximum Y coordinate for the rectangular region.

References OGRLineString::addPoint(), OGRPolygon::addRing(), and SetSpatialFilter().

11.52.2.31 void OGRLayer::SetStyleTable (OGRStyleTable * *poStyleTable*) [virtual]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTableDirectly()** (p. 175) except that it does not assume ownership of the passed table.

This method is the same as the C function OGR_L_SetStyleTable().

Parameters

poStyleTable	pointer to style table to set

References OGRStyleTable::Clone().

11.52.2.32 void OGRLayer::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [virtual]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTable()** (p. 175) except that it assumes ownership of the passed table.

This method is the same as the C function OGR_L_SetStyleTableDirectly().

Parameters

poStyleTable	pointer to style table to set

11.52.2.33 OGRErr OGRLayer::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

This call is intended to force the layer to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some layers do not implement this method, and will still return OGRERR_NONE. The default implementation just returns OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

In any event, you should always close any opened datasource with **OGRDataSource::DestroyDataSource()** (p. 88) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR_L_SyncToDisk()** (p. 468).

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

Referenced by OGRDataSource::SyncToDisk().

11.52.2.34 int OGRLayer::TestCapability (const char * *pszCap*) [pure virtual]

Test if this layer supported the named capability.

The capability codes that can be tested are represented as strings, but #defined constants exists to ensure correct spelling. Specific layer types may implement class specific capabilities, but this can't generally be discovered by the caller.

- **OLCRandomRead** / "RandomRead": TRUE if the **GetFeature()** (p. 167) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. 172) and **GetNextFeature()** (p. 170) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. 165) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. 173) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCFastSpatialFilter** / "FastSpatialFilter": TRUE if this layer implements spatial filtering efficiently. Layers that effectively read all features, and test them with the **OGRFeature** (p. 94) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain its own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via **GetFeatureCount()** (p. 168)) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **GetExtent()** (p. 167)) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the **SetNextByIndex()** (p. 174) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. 165), otherwise FALSE.

- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. 166), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. 171) or **ReorderFields()** (p. 172), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. 164), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. 166) method is supported on this layer, otherwise FALSE.
- **OLCStringsAsUTF8** / "StringsAsUTF8": TRUE if values of OFTString fields are assured to be in UTF-8 format. If FALSE the encoding of fields is uncertain, though it might still be UTF-8.
- **OLCTransactions** / "Transactions": TRUE if the StartTransaction(), CommitTransaction() and RollbackTransaction() methods work in a meaningful way, otherwise FALSE.
- **OLCIgnoreFields** / "IgnoreFields": TRUE if fields, geometry and style will be omitted when fetching features as set by **SetIgnoredFields()** (p. 173) method.

This method is the same as the C function **OGR_L_TestCapability()** (p. 469).

Parameters

pszCap	the name of the capability to test.

Returns

TRUE if the layer has the requested capability, or FALSE otherwise. OGRLayers will return FALSE for any unrecognised capabilities.

Implemented in **OGRGenSQLResultsLayer** (p. 125).

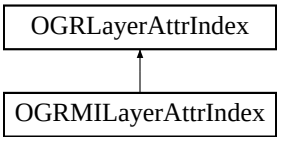
Referenced by OGRDataSource::CopyLayer(), and OGRGenSQLResultsLayer::TestCapability().

The documentation for this class was generated from the following files:

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrlayer.cpp

11.53 OGRLayerAttrIndex Class Reference

Inheritance diagram for OGRLayerAttrIndex:



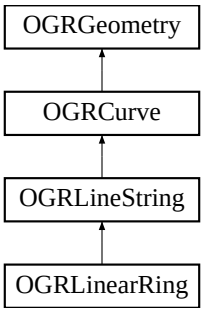
The documentation for this class was generated from the following files:

- ogr_attrind.h
- ogr_attrind.cpp

11.54 OGRLinearRing Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRLinearRing:



Public Member Functions

- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual int **isClockwise** () const

  Returns TRUE if the ring has clockwise winding (or less than 2 points)
- virtual void **closeRings** ()

  Force rings to be closed.
- virtual double **get_Area** () const

  Compute area of ring.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const

  Convert a geometry into well known binary format.

Friends

- class **OGRPolygon**

11.54.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivelent to an **OGRLineString** (p. 181), but has a separate identity to maintain alignment with the OpenGIS simple feature data model. It exists to serve as a component of an **OGRPolygon** (p. 211).

The **OGRLinearRing** (p. 178) has no corresponding free standing well known binary representation, so **importFromWkb()** (p. 180) and **exportToWkb()** (p. 179) will not actually work. There is a non-standard GDAL WKT representation though.

Because **OGRLinearRing** (p. 178) is not a "proper" free standing simple features object, it cannot be directly used on a feature via SetGeometry(), and cannot genearally be used with GEOS for operations like **Intersects()** (p. 139). Instead the polygon should be used, or the **OGRLinearRing** (p. 178) should be converted to an **OGRLineString** (p. 181) for such operations.

11.54.2 Member Function Documentation

11.54.2.1 OGRGeometry * OGRLinearRing::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRLineString** (p. 184).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRLineString::setPoints().

11.54.2.2 void OGRLinearRing::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 130).

References OGRLineString::addPoint(), OGRGeometry::getCoordinateDimension(), OGRLineString::getX(), OGRLineString::getY(), and OGRLineString::getZ().

11.54.2.3 OGRErr OGRLinearRing::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRLineString** (p. 185).

11.54.2.4 double OGRLinearRing::get_Area ( ) const [virtual]

Compute area of ring.

The area is computed according to Green's Theorem:

Area is "Sum(x(i)*(y(i+1) - y(i-1)))/2" for i = 0 to pointCount-1, assuming the last point is a duplicate of the first.

Returns

computed area.

Referenced by OGRPolygon::get_Area().

11.54.2.5 const char * OGRLinearRing::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRLineString** (p. 187).

11.54.2.6 OGRErr OGRLinearRing::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRLineString** (p. 189).

11.54.2.7 int OGRLinearRing::isClockwise ( ) const [virtual]

Returns TRUE if the ring has clockwise winding (or less than 2 points)

Returns

TRUE if clockwise otherwise FALSE.

Referenced by OGRGeometryFactory::organizePolygons().

11.54.2.8 int OGRLinearRing::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Reimplemented from **OGRLineString** (p. 194).

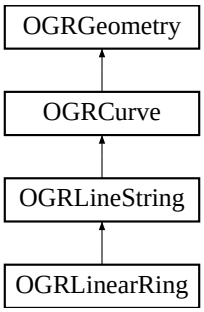
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrlinearring.cpp

11.55 OGRLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRLineString:



Public Member Functions

- **OGRLineString** ()

  Create an empty line string.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char ∗, int=-1)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char ∗) const

  Convert a geometry into well known binary format.
- virtual OGRErr **importFromWkt** (char ∗∗)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char ∗∗ppszDstText) const

  Convert a geometry into well known text format.
- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual **OGRGeometry** ∗ **clone** () const

  Make a copy of this object.
- virtual void **empty** ()

  Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.
- virtual void **getEnvelope** (**OGREnvelope** ∗psEnvelope) const

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

- virtual void **getEnvelope** (**OGREnvelope3D** *psEnvelope) const

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.
- virtual OGRBoolean **IsEmpty** () const

  Returns TRUE (non-zero) if the object has no points.
- virtual double **get_Length** () const

  Returns the length of the curve.
- virtual void **StartPoint** (**OGRPoint** *) const

  Return the curve start point.
- virtual void **EndPoint** (**OGRPoint** *) const

  Return the curve end point.
- virtual void **Value** (double, **OGRPoint** *) const

  Fetch point at given distance along curve.
- int **getNumPoints** () const

  Fetch vertex count.
- void **getPoint** (int, **OGRPoint** *) const

  Fetch a point in line string.
- double **getX** (int i) const

  Get X at vertex.
- double **getY** (int i) const

  Get Y at vertex.
- double **getZ** (int i) const

  Get Z at vertex.
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- void **setNumPoints** (int)

  Set number of points in geometry.
- void **setPoint** (int, **OGRPoint** *)

  Set the location of a vertex in line string.
- void **setPoint** (int, double, double, double)

  Set the location of a vertex in line string.
- void **setPoints** (int, **OGRRawPoint** *, double *=NULL)

  Assign all points in a line string.
- void **setPoints** (int, double *padfX, double *padfY, double *padfZ=NULL)

  Assign all points in a line string.
- void **addPoint** (**OGRPoint** *)

  Add a point to a line string.
- void **addPoint** (double, double, double)

  Add a point to a line string.
- void **getPoints** (**OGRRawPoint** *, double *=NULL) const

  Returns all points of line string.
- void **getPoints** (void *pabyX, int nXStride, void *pabyY, int nYStride, void *pabyZ=NULL, int nZStride=0) const

  Returns all points of line string.
- void **addSubLineString** (const **OGRLineString** *, int nStartVertex=0, int nEndVertex=-1)

  Add a segment of another linestring to this one.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual const char * **getGeometryName** () const

Fetch WKT name for geometry type.

- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)

  Apply arbitrary coordinate transformation to geometry.

- virtual void **flattenTo2D** ()

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

- virtual void **segmentize** (double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.

- virtual void **swapXY** ()

  Swap x and y coordinates.

11.55.1 Detailed Description

Concrete representation of a multi-vertex line.

11.55.2 Member Function Documentation

11.55.2.1 void OGRLineString::addPoint (OGRPoint * *poPoint*)

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

Parameters

poPoint	the point to assign to the new vertex.

References OGRGeometry::getCoordinateDimension(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), and setPoint().

Referenced by OGRLinearRing::closeRings(), OGRBuildPolygonFromEdges(), and OGRLayer::SetSpatialFilterRect().

11.55.2.2 void OGRLineString::addPoint (double *x*, double *y*, double *z*)

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

Parameters

x	the X coordinate to assign to the new point.
y	the Y coordinate to assign to the new point.
z	the Z coordinate to assign to the new point (defaults to zero).

References setPoint().

11.55.2.3 void OGRLineString::addSubLineString (const OGRLineString * *poOtherLine*, int *nStartVertex* = 0, int *nEndVertex* = -1)

Add a segment of another linestring to this one.

Adds the request range of vertices to the end of this line string in an efficient manner. If the nStartVertex is larger than the nEndVertex then the vertices will be reversed as they are copied.

Parameters

poOtherLine	the other **OGRLineString** (p. 181).
nStartVertex	the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.
nEndVertex	the last vertex to copy, defaults to -1 indicating the last vertex of the other line string.

References getNumPoints(), and setNumPoints().

Referenced by OGRGeometryFactory::forceToMultiLineString().

11.55.2.4 OGRGeometry * OGRLineString::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 130).

Reimplemented in **OGRLinearRing** (p. 179).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getSpatialReference(), OGRLineString(), setCoordinateDimension(), and setPoints().

11.55.2.5 void OGRLineString::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 435).

Implements **OGRGeometry** (p. 133).

References setNumPoints().

Referenced by importFromWkt().

11.55.2.6 void OGRLineString::EndPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get_EndPoint() method.

Parameters

poPoint	the point to be assigned the end location.

Implements **OGRCurve** (p. 84).

References getPoint().

Referenced by Value().

11.55.2.7 OGRBoolean OGRLineString::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR_G_Equals()** (p. 436).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 133).

References OGRGeometry::getGeometryType(), getGeometryType(), getNumPoints(), getX(), getY(), and getZ().

Referenced by OGRPolygon::Equals().

11.55.2.8 OGRErr OGRLineString::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

Reimplemented in **OGRLinearRing** (p. 179).

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

11.55.2.9 OGRErr OGRLineString::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 135).

References OGRGeometry::getCoordinateDimension(), and getGeometryName().

Referenced by OGRPolygon::exportToWkt().

11.55.2.10 void OGRLineString::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 439).

Implements **OGRGeometry** (p. 135).

11.55.2.11 double OGRLineString::get_Length ( ) const [virtual]

Returns the length of the curve.

This method relates to the SFCOM ICurve::get_Length() method.

Returns

the length of the curve, zero if the curve hasn't been initialized.

Implements **OGRCurve** (p. 84).

11.55.2.12 int OGRLineString::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 135)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 441).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 136).

11.55.2.13 void OGRLineString::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. 136).

Referenced by getEnvelope(), and OGRPolygon::getEnvelope().

11.55.2.14 void OGRLineString::getEnvelope (OGREnvelope3D * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope3D()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. 136).

References getEnvelope().

11.55.2.15 const char * OGRLineString::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. 137).

Reimplemented in **OGRLinearRing** (p. 180).

Referenced by exportToWkt(), and importFromWkt().

11.55.2.16 OGRwkbGeometryType OGRLineString::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 137).

References OGRGeometry::getCoordinateDimension(), wkbLineString, and wkbLineString25D.

Referenced by Equals(), and exportToWkb().

11.55.2.17 int OGRLineString::getNumPoints ( ) const [inline]

Fetch vertex count.

Returns the number of vertices in the line string.

Returns

vertex count.

Referenced by addSubLineString(), OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGR_G_GetPoint(), OGR_G_GetPointCount(), OGR_G_GetPoints(), OGR_G_GetX(), OGR_G_GetY(), OGR_G_GetZ(), OGRBuildPolygonFromEdges(), and OGRGeometryFactory::organizePolygons().

11.55.2.18 void OGRLineString::getPoint (int *i*, OGRPoint * *poPoint*) const

Fetch a point in line string.

This method relates to the SFCOM ILineString::get_Point() method.

Parameters

i	the vertex to fetch, from 0 to **getNumPoints()** (p. 187)-1.
poPoint	a point to initialize with the fetched point.

References OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), and OGRPoint::setZ().

Referenced by EndPoint(), OGRGeometryFactory::organizePolygons(), and StartPoint().

11.55.2.19 void OGRLineString::getPoints (OGRRawPoint * *paoPointsOut*, double * *padfZ* = NULL) const

Returns all points of line string.

This method copies all points into user list. This list must be at least sizeof(OGRRawPoint) * OGRGeometry::getNumPoints() byte in size. It also copies all Z coordinates.

There is no SFCOM analog to this method.

Parameters

paoPointsOut	a buffer into which the points is written.
padfZ	the Z values that go with the points (optional, may be NULL).

Referenced by getPoints(), and OGR_G_GetPoints().

11.55.2.20 void OGRLineString::getPoints (void * *pabyX*, int *nXStride*, void * *pabyY*, int *nYStride*, void * *pabyZ* = NULL, int *nZStride* = 0) const

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutives elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

There is no SFCOM analog to this method.

Parameters

pabyX	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
nXStride	the number of bytes between 2 elements of pabyX.
pabyY	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
nYStride	the number of bytes between 2 elements of pabyY.
pabyZ	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
nZStride	the number of bytes between 2 elements of pabyZ.

Since

OGR 1.9.0

References getPoints().

11.55.2.21 double OGRLineString::getX (int *iVertex*) const [inline]

Get X at vertex.

Returns the X value at the indicated vertex. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. 187)-1.

Returns

X value.

Referenced by OGRLinearRing::closeRings(), Equals(), OGR_G_GetPoint(), OGR_G_GetX(), and OGRBuildPolygonFromEdges().

11.55.2.22 double OGRLineString::getY (int *iVertex*) const [inline]

Get Y at vertex.

Returns the Y value at the indicated vertex. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. 187)-1.

Returns

X value.

Referenced by OGRLinearRing::closeRings(), Equals(), OGR_G_GetPoint(), OGR_G_GetY(), and OGRBuildPolygonFromEdges().

11.55.2.23 double OGRLineString::getZ (int *iVertex*) const

Get Z at vertex.

Returns the Z (elevation) value at the indicated vertex. If no Z value is available, 0.0 is returned. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. 187)-1.

Returns

Z value.

Referenced by OGRLinearRing::closeRings(), Equals(), OGR_G_GetPoint(), OGR_G_GetZ(), and OGRBuildPolygonFromEdges().

11.55.2.24 OGRErr OGRLineString::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

Reimplemented in **OGRLinearRing** (p. 180).

References setNumPoints(), and wkbLineString.

11.55.2.25 OGRErr OGRLineString::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

References empty(), and getGeometryName().

11.55.2.26 OGRBoolean OGRLineString::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 139).

11.55.2.27 void OGRLineString::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 451)

Parameters

dfMaxLength	the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 141).

References OGRGeometry::getCoordinateDimension().

11.55.2.28 void OGRLineString::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 141).

Referenced by clone(), and OGRPolygon::exportToWkt().

11.55.2.29 void OGRLineString::setNumPoints (int *nNewPointCount*)

Set number of points in geometry.

This method primary exists to preset the number of points in a linestring geometry before **setPoint()** (p. 191) is used to assign them to avoid reallocating the array larger with each call to **addPoint()** (p. 183).

This method has no SFCOM analog.

Parameters

nNewPointCount	the new number of points for geometry.

References OGRGeometry::getCoordinateDimension().

Referenced by addSubLineString(), empty(), importFromWkb(), setPoint(), and setPoints().

11.55.2.30 void OGRLineString::setPoint (int *iPoint,* OGRPoint * *poPoint*)

Set the location of a vertex in line string.

If iPoint is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

Parameters

iPoint	the index of the vertex to assign (zero based).
poPoint	the value to assign to the vertex.

References OGRPoint::getX(), OGRPoint::getY(), and OGRPoint::getZ().

Referenced by addPoint(), and OGRGeometryFactory::approximateArcAngles().

11.55.2.31 void OGRLineString::setPoint (int *iPoint,* double *xIn,* double *yIn,* double *zIn*)

Set the location of a vertex in line string.

If iPoint is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

Parameters

iPoint	the index of the vertex to assign (zero based).
xIn	input X coordinate to assign.
yIn	input Y coordinate to assign.
zIn	input Z coordinate to assign (defaults to zero).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

11.55.2.32 void OGRLineString::setPoints (int *nPointsIn,* OGRRawPoint * *paoPointsIn,* double * *padfZ* = NULL)

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

Parameters

nPointsIn	number of points being passed in paoPointsIn
paoPointsIn	list of points being assigned.
padfZ	the Z values that go with the points (optional, may be NULL).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

Referenced by clone(), OGRLinearRing::clone(), OGRPolygon::importFromWkt(), OGRMultiPolygon::importFromWkt(), OGRMultiLineString::importFromWkt(), and transform().

11.55.2.33 void OGRLineString::setPoints (int *nPointsIn,* double * *padfX,* double * *padfY,* double * *padfZ* = NULL)

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

Parameters

nPointsIn	number of points being passed in padfX and padfY.
padfX	list of X coordinates of points being assigned.
padfY	list of Y coordinates of points being assigned.
padfZ	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints().

11.55.2.34 void OGRLineString::StartPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get_StartPoint() method.

Parameters

poPoint	the point to be assigned the start location.

Implements **OGRCurve** (p. 85).

References getPoint().

Referenced by Value().

11.55.2.35 void OGRLineString::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. 142).

11.55.2.36 OGRErr OGRLineString::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 454).

Parameters

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 143).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), setPoints(), and OGRCoordinateTransformation::TransformEx().

Referenced by OGRPolygon::transform().

11.55.2.37 void OGRLineString::Value (double *dfDistance,* OGRPoint * *poPoint*) const [virtual]

Fetch point at given distance along curve.

This method relates to the SF COM ICurve::get_Value() method.

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. 186) for this curve.
poPoint	the point to be assigned the curve position.

Implements **OGRCurve** (p. 85).

References EndPoint(), OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), OGRPoint::setZ(), and StartPoint().

11.55.2.38 int OGRLineString::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 145).

Reimplemented in **OGRLinearRing** (p. 180).

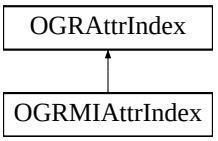
References OGRGeometry::getCoordinateDimension().

The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrlinestring.cpp

11.56 OGRMIAttrIndex Class Reference

Inheritance diagram for OGRMIAttrIndex:

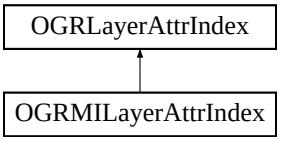


The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

11.57 OGRMILayerAttrIndex Class Reference

Inheritance diagram for OGRMILayerAttrIndex:



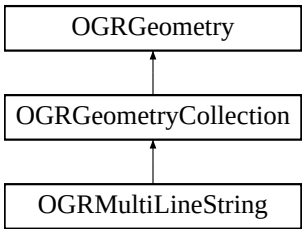
The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

11.58 OGRMultiLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiLineString:



Public Member Functions

- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **) const

  Convert a geometry into well known text format.
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

  Add a geometry directly to the container.

11.58.1 Detailed Description

A collection of OGRLineStrings.

11.58.2 Member Function Documentation

11.58.2.1 OGRErr OGRMultiLineString::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. 147) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 426).

There is no SFCOM analog to this method.

Parameters

poNewGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometry::getGeometryType(), wkbLineString, and wkbLineString25D.

Referenced by OGRGeometryFactory::forceToMultiLineString(), and importFromWkt().

11.58.2.2 OGRGeometry * OGRMultiLineString::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

11.58.2.3 OGRErr OGRMultiLineString::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

11.58.2.4 const char * OGRMultiLineString::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. 151).

Referenced by importFromWkt().

11.58.2.5 OGRwkbGeometryType OGRMultiLineString::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 152).

References OGRGeometry::getCoordinateDimension(), wkbMultiLineString, and wkbMultiLineString25D.

11.58.2.6 OGRErr OGRMultiLineString::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. 153).

References addGeometryDirectly(), OGRGeometryCollection::empty(), getGeometryName(), and OGRLineString::setPoints().

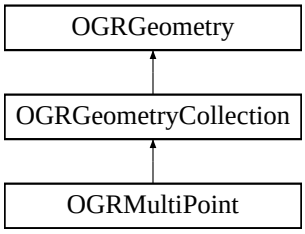
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultilinestring.cpp

11.59 OGRMultiPoint Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPoint:



Public Member Functions

- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **) const

  Convert a geometry into well known text format.
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

  Add a geometry directly to the container.

11.59.1 Detailed Description

A collection of OGRPoints.

11.59.2 Member Function Documentation

11.59.2.1 OGRErr OGRMultiPoint::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. 147) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 426).

There is no SFCOM analog to this method.

Parameters

poNewGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometry::getGeometryType(), wkbPoint, and wkbPoint25D.

Referenced by OGRGeometryFactory::forceToMultiPoint(), and importFromWkt().

11.59.2.2 OGRGeometry * OGRMultiPoint::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

11.59.2.3 OGRErr OGRMultiPoint::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::getCoordinateDimension(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRPoint::IsEmpty(), and OGRGeometryCollection::IsEmpty().

11.59.2.4 const char * OGRMultiPoint::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. 151).

Referenced by exportToWkt(), and importFromWkt().

11.59.2.5 OGRwkbGeometryType OGRMultiPoint::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 152).

References OGRGeometry::getCoordinateDimension(), wkbMultiPoint, and wkbMultiPoint25D.

11.59.2.6 OGRErr OGRMultiPoint::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. 153).

References addGeometryDirectly(), OGRGeometryCollection::empty(), and getGeometryName().

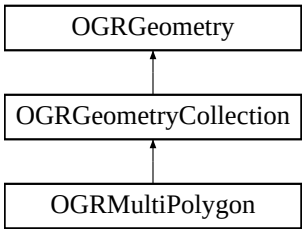
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipoint.cpp

11.60 OGRMultiPolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPolygon:



Public Member Functions

- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **) const

  Convert a geometry into well known text format.
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

  Add a geometry directly to the container.
- virtual double **get_Area** () const

11.60.1 Detailed Description

A collection of non-overlapping OGRPolygons.

Note that the IMultiSurface class hasn't been modelled, nor have any of it's methods.

11.60.2 Member Function Documentation

11.60.2.1 OGRErr OGRMultiPolygon::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. 147) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 426).

There is no SFCOM analog to this method.

Parameters

poNewGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometry::getGeometryType(), wkbPolygon, and wkbPolygon25D.

Referenced by OGRGeometryFactory::forceToMultiPolygon(), and importFromWkt().

11.60.2.2 OGRGeometry * OGRMultiPolygon::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

11.60.2.3 OGRErr OGRMultiPolygon::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

11.60.2.4 double OGRMultiPolygon::get_Area ( ) const [virtual]

Compute area of multipolygon.

The area is computed as the sum of the areas of all polygon members in this collection.

Returns

computed area.

Reimplemented from **OGRGeometryCollection** (p. 150).

References OGRPolygon::get_Area(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

11.60.2.5 const char * OGRMultiPolygon::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. 151).

Referenced by importFromWkt().

11.60.2.6 OGRwkbGeometryType OGRMultiPolygon::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 152).

References OGRGeometry::getCoordinateDimension(), wkbMultiPolygon, and wkbMultiPolygon25D.

11.60.2.7 OGRErr OGRMultiPolygon::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. 153).

References addGeometryDirectly(), OGRPolygon::addRingDirectly(), OGRGeometryCollection::empty(), getGeometryName(), and OGRLineString::setPoints().

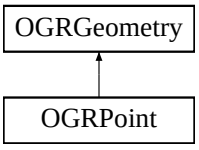
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipolygon.cpp

11.61 OGRPoint Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRPoint:



Public Member Functions

- **OGRPoint** ()

  Create a (0,0) point.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const

  Convert a geometry into well known binary format.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **ppszDstText) const

  Convert a geometry into well known text format.
- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual void **empty** ()

  Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const

  Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.
- virtual void **getEnvelope** (**OGREnvelope3D** *psEnvelope) const

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.
- virtual OGRBoolean **IsEmpty** () const

  Returns TRUE (non-zero) if the object has no points.
- double **getX** () const

  Fetch X coordinate.
- double **getY** () const

  Fetch Y coordinate.
- double **getZ** () const

  Fetch Z coordinate.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- void **setX** (double xIn)

  Assign point X coordinate.
- void **setY** (double yIn)

  Assign point Y coordinate.
- void **setZ** (double zIn)

  Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)

  Apply arbitrary coordinate transformation to geometry.
- virtual void **flattenTo2D** ()

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.
- virtual void **swapXY** ()

  Swap x and y coordinates.

11.61.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

11.61.2 Member Function Documentation

11.61.2.1 OGRGeometry * OGRPoint::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 130).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), OGRPoint(), and setCoordinateDimension().

11.61.2.2 void OGRPoint::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 435).

Implements **OGRGeometry** (p. 133).

Referenced by importFromWkt(), and OGRPoint().

11.61.2.3 OGRBoolean OGRPoint::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR_G_Equals()** (p. 436).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 133).

References OGRGeometry::getGeometryType(), getGeometryType(), getX(), getY(), and getZ().

11.61.2.4 OGRErr OGRPoint::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

References getGeometryType().

11.61.2.5 OGRErr OGRPoint::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 135).

11.61.2.6 void OGRPoint::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 439).

Implements **OGRGeometry** (p. 135).

11.61.2.7 int OGRPoint::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 135)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 441).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 136).

11.61.2.8 void OGRPoint::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. 136).

References getX(), and getY().

11.61.2.9 void OGRPoint::getEnvelope (OGREnvelope3D * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope3D()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. 136).

References getX(), getY(), and getZ().

11.61.2.10 const char * OGRPoint::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. 137).

11.61.2.11 OGRwkbGeometryType OGRPoint::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 137).

References wkbPoint, and wkbPoint25D.

Referenced by Equals(), exportToWkb(), and OGR_G_Centroid().

11.61.2.12 double OGRPoint::getX ( ) const [inline]

Fetch X coordinate.

Relates to the SFCOM IPoint::get_X() method.

Returns

the X coordinate of this point.

Referenced by OGRLineString::addPoint(), OGRGeometry::Centroid(), Equals(), OGRMultiPoint::exportToWkt(), OGRCurve::get_IsClosed(), getEnvelope(), OGRPolygon::PointOnSurface(), and OGRLineString::setPoint().

11.61.2.13 double OGRPoint::getY ( ) const [inline]

Fetch Y coordinate.

Relates to the SFCOM IPoint::get_Y() method.

Returns

the Y coordinate of this point.

Referenced by OGRLineString::addPoint(), OGRGeometry::Centroid(), Equals(), OGRMultiPoint::exportToWkt(), OGRCurve::get_IsClosed(), getEnvelope(), OGRPolygon::PointOnSurface(), and OGRLineString::setPoint().

11.61.2.14 double OGRPoint::getZ ( ) const [inline]

Fetch Z coordinate.

Relates to the SFCOM IPoint::get_Z() method.

Returns

the Z coordinate of this point, or zero if it is a 2D point.

Referenced by OGRLineString::addPoint(), Equals(), OGRMultiPoint::exportToWkt(), getEnvelope(), and OGRLineString::setPoint().

11.61.2.15 OGRErr OGRPoint::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

References wkbPoint.

11.61.2.16 OGRErr OGRPoint::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

References empty().

11.61.2.17 OGRBoolean OGRPoint::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 139).

Referenced by OGRMultiPoint::exportToWkt().

11.61.2.18 void OGRPoint::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 141).

Referenced by clone().

11.61.2.19 void OGRPoint::setX (double *xIn*) [inline]

Assign point X coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

11.61.2.20 void OGRPoint::setY (double *yIn*) [inline]

Assign point Y coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

11.61.2.21 void OGRPoint::setZ (double *zIn*) [inline]

Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).

There is no corresponding SFCOM method.

Referenced by OGRLineString::getPoint(), and OGRLineString::Value().

11.61.2.22 void OGRPoint::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. 142).

11.61.2.23 OGRErr OGRPoint::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 454).

Parameters

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 143).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRCoordinateTransformation::Transform().

11.61.2.24 int OGRPoint::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 145).

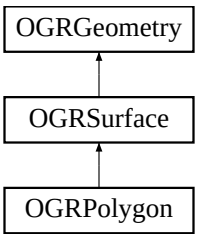
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrpoint.cpp

11.62 OGRPolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRPolygon:



Public Member Functions

- **OGRPolygon** ()

  Create an empty polygon.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual **OGRGeometry** * **clone** () const

  Make a copy of this object.
- virtual void **empty** ()

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)

  Apply arbitrary coordinate transformation to geometry.

- virtual void **flattenTo2D** ()

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

- virtual OGRBoolean **IsEmpty** () const

  Returns TRUE (non-zero) if the object has no points.

- virtual void **segmentize** (double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.

- virtual double **get_Area** () const

  Compute area of polygon.

- virtual int **PointOnSurface** (**OGRPoint** *poPoint) const

  This method relates to the SFCOM ISurface::get_PointOnSurface() method.

- virtual int **WkbSize** () const

  Returns size of related binary representation.

- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)

  Assign geometry from well known binary data.

- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const

  Convert a geometry into well known binary format.

- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.

- virtual OGRErr **exportToWkt** (char **ppszDstText) const

  Convert a geometry into well known text format.

- virtual int **getDimension** () const

  Get the dimension of this object.

- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const

  Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

- virtual void **getEnvelope** (**OGREnvelope3D** *psEnvelope) const

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.

- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.

- void **addRing** (**OGRLinearRing** *)

  Add a ring to a polygon.

- void **addRingDirectly** (**OGRLinearRing** *)

  Add a ring to a polygon.

- **OGRLinearRing** * **getExteriorRing** ()

  Fetch reference to external polygon ring.

- int **getNumInteriorRings** () const

  Fetch the number of internal rings.

- **OGRLinearRing** * **getInteriorRing** (int)

  Fetch reference to indicated internal ring.

- virtual void **closeRings** ()

  Force rings to be closed.

- virtual void **swapXY** ()

  Swap x and y coordinates.

11.62.1 Detailed Description

Concrete class representing polygons.

Note that the OpenGIS simple features polygons consist of one outer ring, and zero or more inner rings. A polygon cannot represent disconnected regions (such as multiple islands in a political body). The **OGRMultiPolygon** (p. 200) must be used for this.

11.62.2 Member Function Documentation

11.62.2.1 void OGRPolygon::addRing (OGRLinearRing * *poNewRing*)

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. The passed **OGRLinearRing** (p. 178) remains the responsibility of the caller (an internal copy is made).

This method has no SFCOM analog.

Parameters

poNewRing	ring to be added to the polygon.

References OGRGeometry::getCoordinateDimension().

Referenced by clone(), OGRGeometryFactory::forceToPolygon(), OGRGeometryFactory::organizePolygons(), and OGRLayer::SetSpatialFilterRect().

11.62.2.2 void OGRPolygon::addRingDirectly (OGRLinearRing * *poNewRing*)

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. Ownership of the passed ring is assumed by the **OGRPolygon** (p. 211), but otherwise this method operates the same as OGRPolygon::AddRing().

This method has no SFCOM analog.

Parameters

poNewRing	ring to be added to the polygon.

References OGRGeometry::getCoordinateDimension().

Referenced by OGRMultiPolygon::importFromWkt(), and OGRBuildPolygonFromEdges().

11.62.2.3 OGRGeometry * OGRPolygon::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 430).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 130).

References addRing(), OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRPolygon().

11.62.2.4 void OGRPolygon::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 130).

11.62.2.5 void OGRPolygon::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 435).

Implements **OGRGeometry** (p. 133).

Referenced by importFromWkt().

11.62.2.6 OGRBoolean OGRPolygon::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR_G_Equals()** (p. 436).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 133).

References OGRLineString::Equals(), getExteriorRing(), OGRGeometry::getGeometryType(), getGeometryType(), getInteriorRing(), and getNumInteriorRings().

11.62.2.7 OGRErr OGRPolygon::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 438).

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 145) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

11.62.2.8 OGRErr OGRPolygon::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 438).

Parameters

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 135).

References OGRLineString::exportToWkt(), OGRGeometry::getCoordinateDimension(), getExteriorRing(), IsEmpty(), and OGRLineString::setCoordinateDimension().

Referenced by OGRGeometryFactory::organizePolygons().

11.62.2.9 void OGRPolygon::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 439).

Implements **OGRGeometry** (p. 135).

11.62.2.10 double OGRPolygon::get_Area ( ) const [virtual]

Compute area of polygon.

The area is computed as the area of the outer ring less the area of all internal rings.

Returns

computed area.

Implements **OGRSurface** (p. 293).

References OGRLinearRing::get_Area(), getExteriorRing(), getInteriorRing(), and getNumInteriorRings().

Referenced by OGRMultiPolygon::get_Area(), and OGRGeometryFactory::organizePolygons().

11.62.2.11 int OGRPolygon::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 135)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 441).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 136).

11.62.2.12 void OGRPolygon::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. 136).

References OGRLineString::getEnvelope(), and IsEmpty().

11.62.2.13 void OGRPolygon::getEnvelope (OGREnvelope3D * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope3D()** (p. 442).

Parameters

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. 136).

References OGRLineString::getEnvelope(), and IsEmpty().

11.62.2.14 OGRLinearRing * OGRPolygon::getExteriorRing ( )

Fetch reference to external polygon ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. 211). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. 130) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get_ExteriorRing() method.

Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. 211) is empty.

Referenced by OGRGeometry::dumpReadable(), Equals(), exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), get_Area(), and OGRGeometryFactory::organizePolygons().

11.62.2.15 const char * OGRPolygon::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 443).

Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. 137).

11.62.2.16 OGRwkbGeometryType OGRPolygon::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 443).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 137).

References OGRGeometry::getCoordinateDimension(), wkbPolygon, and wkbPolygon25D.

Referenced by Equals(), and exportToWkb().

11.62.2.17 OGRLinearRing * OGRPolygon::getInteriorRing (int *iRing*)

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. 211). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. 130) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get_InternalRing() method.

Parameters

iRing	internal ring index from 0 to getNumInternalRings() - 1.

Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. 211) is empty.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get_Area().

11.62.2.18 int OGRPolygon::getNumInteriorRings ( ) const

Fetch the number of internal rings.

Relates to the SFCOM IPolygon::get_NumInteriorRings() method.

Returns

count of internal rings, zero or more.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get_Area().

11.62.2.19 OGRErr OGRPolygon::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR_G_ImportFromWkb()** (p. 446).

Parameters

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

References VSIMalloc2(), and wkbPolygon.

11.62.2.20 OGRErr OGRPolygon::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 156) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR_G_ImportFromWkt()** (p. 446).

Parameters

ppszInput	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 138).

References empty(), and OGRLineString::setPoints().

11.62.2.21 OGRBoolean OGRPolygon::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 139).

Referenced by exportToWkt(), and getEnvelope().

11.62.2.22 int OGRPolygon::PointOnSurface (OGRPoint * *poPoint*) const [virtual]

This method relates to the SFCOM ISurface::get_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

Parameters

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implements **OGRSurface** (p. 294).

References OGRGeometry::getGeometryType(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::setX(), OGRPoint::setY(), and wkbPoint.

11.62.2.23 void OGRPolygon::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 451)

Parameters

dfMaxLength	the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 141).

11.62.2.24 void OGRPolygon::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 141).

11.62.2.25 void OGRPolygon::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. 142).

11.62.2.26 OGRErr OGRPolygon::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 454).

Parameters

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 143).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRLineString::transform().

11.62.2.27 int OGRPolygon::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 456).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 145).

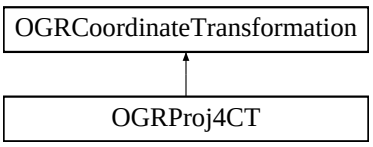
References OGRGeometry::getCoordinateDimension().

The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrpolygon.cpp

11.63 OGRProj4CT Class Reference

Inheritance diagram for OGRProj4CT:



Public Member Functions

- virtual **OGRSpatialReference** ∗ **GetSourceCS** ()
- virtual **OGRSpatialReference** ∗ **GetTargetCS** ()
- virtual int **Transform** (int nCount, double ∗x, double ∗y, double ∗z=NULL)
- virtual int **TransformEx** (int nCount, double ∗x, double ∗y, double ∗z=NULL, int ∗panSuccess=NULL)

Additional Inherited Members

11.63.1 Member Function Documentation

11.63.1.1 OGRSpatialReference ∗ OGRProj4CT::GetSourceCS ( ) [virtual]

Fetch internal source coordinate system.

Implements **OGRCoordinateTransformation** (p. 82).

11.63.1.2 OGRSpatialReference ∗ OGRProj4CT::GetTargetCS ( ) [virtual]

Fetch internal target coordinate system.

Implements **OGRCoordinateTransformation** (p. 82).

11.63.1.3 int OGRProj4CT::Transform (int *nCount*, double ∗ *x*, double ∗ *y*, double ∗ *z* = NULL) [virtual]

Transform points from source to destination space.

This method is the same as the C function OCTTransform().

The method **TransformEx()** (p. 221) allows extended success information to be captured indicating which points failed to transform.

Parameters

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.

Returns

TRUE on success, or FALSE if some or all points fail to transform.

Implements **OGRCoordinateTransformation** (p. 82).

References TransformEx().

11.63.1.4 int OGRProj4CT::TransformEx (int *nCount*, double ∗ *x*, double ∗ *y*, double ∗ *z* = NULL, int ∗ *pabSuccess* = NULL) [virtual]

Transform points from source to destination space.

This method is the same as the C function OCTTransformEx().

Parameters

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.
pabSuccess	array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

Returns

TRUE if some or all points transform successfully, or FALSE if if none transform.

Implements **OGRCoordinateTransformation** (p. 83).

Referenced by Transform().

The documentation for this class was generated from the following file:

- ogrct.cpp

11.64 OGRProj4Datum Struct Reference

The documentation for this struct was generated from the following file:

- ogr_srs_proj4.cpp

11.65 OGRProj4PM Struct Reference

The documentation for this struct was generated from the following file:

- ogr_srs_proj4.cpp

11.66 OGRRawPoint Class Reference

`#include <ogr_geometry.h>`

11.66.1 Detailed Description

Simple container for a position.

The documentation for this class was generated from the following file:

- **ogr_geometry.h**

11.67 OGRSFDriver Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * **GetName** ()=0

  Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".
- virtual **OGRDataSource** * **Open** (const char *pszName, int bUpdate=FALSE)=0

  Attempt to open file with this driver.
- virtual int **TestCapability** (const char *)=0

  Test if capability is available.
- virtual **OGRDataSource** * **CreateDataSource** (const char *pszName, char **=NULL)

  This method attempts to create a new data source based on the passed driver.
- virtual OGRErr **DeleteDataSource** (const char *pszName)

  Delete a datasource.
- virtual **OGRDataSource** * **CopyDataSource** (**OGRDataSource** *poSrcDS, const char *pszNewName, char **papszOptions=NULL)

  This method creates a new datasource by copying all the layers from the source datasource.

11.67.1 Detailed Description

Represents an operational format driver.

One **OGRSFDriver** (p. 222) derived class will normally exist for each file format registered for use, regardless of whether a file has or will be opened. The list of available drivers is normally managed by the **OGRSFDriverRegistrar** (p. 226).

11.67.2 Member Function Documentation

11.67.2.1 OGRDataSource * OGRSFDriver::CopyDataSource (OGRDataSource * *poSrcDS,* const char * *pszNewName,* char ** *papszOptions* = NULL) [virtual]

This method creates a new datasource by copying all the layers from the source datasource.

It is important to call **OGRDataSource::DestroyDataSource()** (p. 88) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This method is the same as the C function **OGR_Dr_CopyDataSource()** (p. 395).

Parameters

poSrcDS	source datasource
pszNewName	the name for the new data source. UTF-8 encoded.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: http://www.gdal.org/ogr/ogr_formats.html

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. 85) handle on success.

References OGRDataSource::CopyLayer(), CreateDataSource(), OGRDataSource::GetDriver(), OGRDataSource::GetLayer(), OGRDataSource::GetLayerCount(), OGRLayer::GetLayerDefn(), OGRFeatureDefn::GetName(), GetName(), OGRDataSource::SetDriver(), and TestCapability().

11.67.2.2 OGRDataSource * OGRSFDriver::CreateDataSource (const char * *pszName,* char ** *papszOptions* = NULL) [virtual]

This method attempts to create a new data source based on the passed driver.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

It is important to call **OGRDataSource::DestroyDataSource()** (p. 88) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This method is the same as the C function **OGR_Dr_CreateDataSource()** (p. 395).

Note

This method does **NOT** attach driver instance to the returned data source, so caller should expect that **OGRDataSource::GetDriver()** (p. 89) will return NULL pointer. In order to attach driver to the returned data source, it is required to use C function OGR_Dr_CreateDataSource. This behavior is related to fix of issue reported in `Ticket #1233`.

Parameters

pszName	the name for the new data source. UTF-8 encoded.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. 85) on success.

Referenced by CopyDataSource(), and OGR_Dr_CreateDataSource().

11.67.2.3 OGRErr OGRSFDriver::DeleteDataSource (const char * *pszDataSource*) [virtual]

Delete a datasource.

Delete (from the disk, in the database, ...) the named datasource. Normally it would be safest if the datasource was not open at the time.

Whether this is a supported operation on this driver case be tested using **TestCapability()** (p. 225) on ODrCDeleteDataSource.

This method is the same as the C function **OGR_Dr_DeleteDataSource()** (p. 396).

Parameters

pszDataSource	the name of the datasource to delete.

Returns

OGRERR_NONE on success, and OGRERR_UNSUPPORTED_OPERATION if this is not supported by this driver.

11.67.2.4 const char * OGRSFDriver::GetName ( ) [pure virtual]

Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".

This method is the same as the C function **OGR_Dr_GetName()** (p. 396).

Returns

driver name. This is an internal string and should not be modified or freed.

Referenced by CopyDataSource(), OGRDataSource::ExecuteSQL(), OGRSFDriverRegistrar::Open(), and OGRSFDriverRegistrar::RegisterDriver().

11.67.2.5 OGRDataSource * OGRSFDriver::Open (const char * *pszName*, int *bUpdate* = FALSE) [pure virtual]

Attempt to open file with this driver.

This method is what **OGRSFDriverRegistrar** (p. 226) uses to implement its **Open()** (p. 225) method. See it for more details.

Note, drivers do not normally set their own m_poDriver value, so a direct call to this method (instead of indirectly via **OGRSFDriverRegistrar** (p. 226)) will usually result in a datasource that does not know what driver it relates to if GetDriver() is called on the datasource. The application may directly call SetDriver() after opening with this method to avoid this problem.

For drivers supporting the VSI virtual file API, it is possible to open a file in a .zip archive (see **VSIInstallZipFileHandler()** (p. 380)), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()** (p. 380)) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()** (p. 377))

This method is the same as the C function **OGR_Dr_Open()** (p. 396).

Parameters

pszName	the name of the file, or data source to try and open.
bUpdate	TRUE if update access is required, otherwise FALSE (the default).

Returns

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an **OGRDataSource** (p. 85). This **OGRDataSource** (p. 85) should be closed by deleting the object when it is no longer needed.

Referenced by OGRSFDriverRegistrar::Open().

11.67.2.6 int OGRSFDriver::TestCapability (const char * *pszCapability*) [pure virtual]

Test if capability is available.

One of the following data source capability names can be passed into this method, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODrCCreateDataSource**: True if this driver can support creating data sources.
- **ODrCDeleteDataSource**: True if this driver supports deleting data sources.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid mispelling.

This method is the same as the C function **OGR_Dr_TestCapability()** (p. 397).

Parameters

pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

Referenced by CopyDataSource().

The documentation for this class was generated from the following files:

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrsfdriver.cpp

11.68 OGRSFDriverRegistrar Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- void **RegisterDriver** (**OGRSFDriver** *poDriver)

  Add a driver to the list of registered drivers.
- void **DeregisterDriver** (**OGRSFDriver** *poDriver)

  Remove the passed driver from the list of registered drivers.
- int **GetDriverCount** (void)

  Fetch the number of registered drivers.
- **OGRSFDriver** * **GetDriver** (int iDriver)

  Fetch the indicated driver.
- **OGRSFDriver** * **GetDriverByName** (const char *)

  Fetch the indicated driver.
- int **GetOpenDSCount** ()

  Return the number of opened datasources.
- **OGRDataSource** * **GetOpenDS** (int)

  Return the iDS th datasource opened.
- void **AutoLoadDrivers** ()

  Auto-load GDAL drivers from shared libraries.

Static Public Member Functions

- static **OGRSFDriverRegistrar** * **GetRegistrar** ()

  Return the driver manager, creating one if none exist.
- static **OGRDataSource** * **Open** (const char *pszName, int bUpdate=FALSE, **OGRSFDriver** **ppoDriver=NULL)

  Open a file / data source with one of the registered drivers.

11.68.1 Detailed Description

Singleton manager for **OGRSFDriver** (p. 222) instances that will be used to try and open datasources. Normally the registrar is populated with standard drivers using the **OGRRegisterAll()** (p. 393) function and does not need to be directly accessed. The driver registrar and all registered drivers may be cleaned up on shutdown using **OGRCleanupAll()** (p. 479).

11.68.2 Member Function Documentation

11.68.2.1 void OGRSFDriverRegistrar::AutoLoadDrivers ( )

Auto-load GDAL drivers from shared libraries.

This function will automatically load drivers from shared libraries. It searches the "driver path" for .so (or .dll) files that start with the prefix "ogr_X.so". It then tries to load them and then tries to call a function within them called RegisterOGRX() where the 'X' is the same as the remainder of the shared library basename, or failing that to call GDALRegisterMe().

There are a few rules for the driver path. If the GDAL_DRIVER_PATH environment variable it set, it is taken to be a list of directories to search separated by colons on unix, or semi-colons on Windows.

If that is not set the following defaults are used:

- Linux/Unix: <prefix>/lib/gdalplugins is searched or /usr/local/lib/gdalplugins if the install prefix is not known.
- MacOSX: <prefix>/PlugIns is searched, or /usr/local/lib/gdalplugins if the install prefix is not known. Also, the framework directory /Library/Application Support/GDAL/PlugIns is searched.
- Win32: <prefix>/lib/gdalplugins if the prefix is known (normally it is not), otherwise the gdalplugins subdirectory of the directory containing the currently running executable is used.

References CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGetExecPath(), CPLGetExtension(), CPLGetSymbol(), and VSIStatL().

Referenced by OGRRegisterAll().

11.68.2.2 void OGRSFDriverRegistrar::DeregisterDriver (OGRSFDriver * *poDriver*)

Remove the passed driver from the list of registered drivers.

This method is the same as the C function **OGRDeregisterDriver()** (p. 480).

Parameters

poDriver	the driver to deregister.

Since

GDAL 1.8.0

Referenced by OGRDeregisterDriver().

11.68.2.3 OGRSFDriver * OGRSFDriverRegistrar::GetDriver (int *iDriver*)

Fetch the indicated driver.

This method is the same as the C function **OGRGetDriver()** (p. 480).

Parameters

iDriver	the driver index, from 0 to **GetDriverCount()** (p. 228)-1.

Returns

the driver, or NULL if iDriver is out of range.

Referenced by OGRGetDriver().

11.68.2.4 OGRSFDriver * OGRSFDriverRegistrar::GetDriverByName (const char * *pszName*)

Fetch the indicated driver.

This method is the same as the C function OGRGetDriverByName

Parameters

pszName	the driver name

Returns

the driver, or NULL if no driver with that name is found

Referenced by OGRGetDriverByName().

11.68.2.5 int OGRSFDriverRegistrar::GetDriverCount (void)

Fetch the number of registered drivers.

This method is the same as the C function **OGRGetDriverCount()** (p. 481).

Returns

the drivers count.

Referenced by OGRGetDriverCount().

11.68.2.6 OGRDataSource * OGRSFDriverRegistrar::GetOpenDS (int *iDS*)

Return the iDS th datasource opened.

This method is the same as the C function **OGRGetOpenDS()** (p. 481).

Parameters

iDS	the index of the dataset to return (between 0 and **GetOpenDSCount()** (p. 228) - 1)

Referenced by OGRGetOpenDS().

11.68.2.7 int OGRSFDriverRegistrar::GetOpenDSCount ( ) [inline]

Return the number of opened datasources.

This method is the same as the C function **OGRGetOpenDSCount()** (p. 481)

Returns

the number of opened datasources.

Referenced by OGRGetOpenDSCount().

11.68.2.8 OGRSFDriverRegistrar * OGRSFDriverRegistrar::GetRegistrar ( ) [static]

Return the driver manager, creating one if none exist.

Fetch registrar.

Returns

the driver manager.

This static method should be used to fetch the singleton registrar. It will create a registrar if there is not already one in existance.

Returns

the current driver registrar.

Referenced by OGRDataSource::ExecuteSQL(), OGRDeregisterDriver(), OGRGetDriverByName(), OGRGetOpenDS(), OGRGetOpenDSCount(), OGRRegisterAll(), OGRRegisterDriver(), OGRReleaseDataSource(), Open(), and OGRDataSource::Release().

11.68.2.9 OGRDataSource * OGRSFDriverRegistrar::Open (const char * *pszName,* int *bUpdate* = FALSE, OGRSFDriver ** *ppoDriver* = NULL) [static]

Open a file / data source with one of the registered drivers.

This method loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This method is static. Applications don't normally need to use any other **OGRSFDriverRegistrar** (p. 226) methods directly, nor do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 226) instance.

If this method fails, **CPLGetLastErrorMsg()** (p. 335) can be used to check if there is an error message explaining why.

For drivers supporting the VSI virtual file API, it is possible to open a file in a .zip archive (see **VSIInstallZipFileHandler()** (p. 380)), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()** (p. 380)) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()** (p. 377))

This method is the same as the C function **OGROpen()** (p. 481).

Parameters

pszName	the name of the file, or data source to open. UTF-8 encoded.
bUpdate	FALSE for read-only access (the default) or TRUE for read-write access.
ppoDriver	if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

Returns

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an **OGRDataSource** (p. 85). This **OGRDataSource** (p. 85) should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSource (p. 85) *poDS;

poDS = OGRSFDriverRegistrar::Open (p. 229)( "polygon.shp" );
if( poDS == NULL )
{
    return;
}

... use the data source ...

OGRDataSource::DestroyDataSource(poDS);
```

References OGRDataSource::GetDriver(), OGRSFDriver::GetName(), GetRegistrar(), OGRSFDriver::Open(), and OGRDataSource::Reference().

Referenced by OGROpen().

11.68.2.10 void OGRSFDriverRegistrar::RegisterDriver (OGRSFDriver * *poDriver*)

Add a driver to the list of registered drivers.

If the passed driver is already registered (based on pointer comparison) then the driver isn't registered. New drivers are added at the end of the list of registered drivers.

This method is the same as the C function **OGRRegisterDriver()** (p. 482).

Parameters

poDriver	the driver to add.

References OGRSFDriver::GetName().

Referenced by OGRRegisterDriver().

The documentation for this class was generated from the following files:

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrsfdriverregistrar.cpp

11.69 OGRSpatialReference Class Reference

`#include <ogr_spatialref.h>`

Public Member Functions

- **OGRSpatialReference** (const char *=NULL)

  Constructor.
- virtual ~**OGRSpatialReference** ()

  OGRSpatialReference *(p. 230) destructor.*
- int **Reference** ()

  Increments the reference count by one.
- int **Dereference** ()

  Decrements the reference count by one.
- int **GetReferenceCount** () const

  Fetch current reference count.
- void **Release** ()

  Decrements the reference count by one, and destroy if zero.
- **OGRSpatialReference** * **Clone** () const

  Make a duplicate of this ***OGRSpatialReference*** *(p. 230).*
- **OGRSpatialReference** * **CloneGeogCS** () const

  Make a duplicate of the GEOGCS node of this ***OGRSpatialReference*** *(p. 230) object.*
- OGRErr **exportToWkt** (char **) const

  Convert this SRS into WKT format.
- OGRErr **exportToPrettyWkt** (char **, int=FALSE) const
- OGRErr **exportToProj4** (char **) const

  Export coordinate system in PROJ.4 format.
- OGRErr **exportToPCI** (char **, char **, double **) const

  Export coordinate system in PCI projection definition.
- OGRErr **exportToUSGS** (long *, long *, double **, long *) const

  Export coordinate system in USGS GCTP projection definition.
- OGRErr **exportToXML** (char **, const char *=NULL) const

  Export coordinate system in XML format.
- OGRErr **exportToPanorama** (long *, long *, long *, long *, double *) const

- OGRErr **exportToERM** (char *pszProj, char *pszDatum, char *pszUnits)
- OGRErr **exportToMICoordSys** (char **) const

  Export coordinate system in Mapinfo style CoordSys format.
- OGRErr **importFromWkt** (char **)

  Import from WKT string.
- OGRErr **importFromProj4** (const char *)

  Import PROJ.4 coordinate string.
- OGRErr **importFromEPSG** (int)

  Initialize SRS based on EPSG GCS or PCS code.
- OGRErr **importFromEPSGA** (int)

  Initialize SRS based on EPSG GCS or PCS code.
- OGRErr **importFromESRI** (char **)

  Import coordinate system from ESRI .prj format(s).
- OGRErr **importFromPCI** (const char *, const char *=NULL, double *=NULL)

  Import coordinate system from PCI projection definition.
- OGRErr **importFromUSGS** (long iProjSys, long iZone, double *padfPrjParams, long iDatum, int bAnglesInPackedDMSFormat=TRUE)

  Import coordinate system from USGS projection definition.
- OGRErr **importFromPanorama** (long, long, long, double *)
- OGRErr **importFromOzi** (const char *, const char *, const char *)
- OGRErr **importFromWMSAUTO** (const char *pszAutoDef)

  Initialize from WMSAUTO string.
- OGRErr **importFromXML** (const char *)

  Import coordinate system from XML format (GML only currently).
- OGRErr **importFromDict** (const char *pszDict, const char *pszCode)
- OGRErr **importFromURN** (const char *)

  Initialize from OGC URN.
- OGRErr **importFromERM** (const char *pszProj, const char *pszDatum, const char *pszUnits)
- OGRErr **importFromUrl** (const char *)

  Set spatial reference from a URL.
- OGRErr **importFromMICoordSys** (const char *)

  Import Mapinfo style CoordSys definition.
- OGRErr **morphToESRI** ()

  Convert in place to ESRI WKT format.
- OGRErr **morphFromESRI** ()

  Convert in place from ESRI WKT format.
- OGRErr **Validate** ()

  Validate SRS tokens.
- OGRErr **StripCTParms** (**OGR_SRSNode** *=NULL)

  Strip OGC CT Parameters.
- OGRErr **StripVertical** ()

  Convert a compound cs into a horizontal CS.
- OGRErr **FixupOrdering** ()

  Correct parameter ordering to match CT Specification.
- OGRErr **Fixup** ()

  Fixup as needed.
- int **EPSGTreatsAsLatLong** ()

  This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.
- const char * **GetAxis** (const char *pszTargetKey, int iAxis, OGRAxisOrientation *peOrientation) const

  Fetch the orientation of one axis.

- OGRErr **SetAxes** (const char *pszTargetKey, const char *pszXAxisName, OGRAxisOrientation eXAxisOrientation, const char *pszYAxisName, OGRAxisOrientation eYAxisOrientation)

  Set the axes for a coordinate system.
- void **SetRoot** (**OGR_SRSNode** *)

  Set the root SRS node.
- **OGR_SRSNode** * **GetAttrNode** (const char *)

  Find named node in tree.
- const char * **GetAttrValue** (const char *, int=0) const

  Fetch indicated attribute of named node.
- OGRErr **SetNode** (const char *, const char *)

  Set attribute value in spatial reference.
- OGRErr **SetLinearUnitsAndUpdateParameters** (const char *pszName, double dfInMeters)

  Set the linear units for the projection.
- OGRErr **SetLinearUnits** (const char *pszName, double dfInMeters)

  Set the linear units for the projection.
- OGRErr **SetTargetLinearUnits** (const char *pszTargetKey, const char *pszName, double dfInMeters)

  Set the linear units for the projection.
- double **GetLinearUnits** (char **=NULL) const

  Fetch linear projection units.
- double **GetTargetLinearUnits** (const char *pszTargetKey, char **ppszRetName=NULL) const

  Fetch linear units for target.
- OGRErr **SetAngularUnits** (const char *pszName, double dfInRadians)

  Set the angular units for the geographic coordinate system.
- double **GetAngularUnits** (char **=NULL) const

  Fetch angular geographic coordinate system units.
- double **GetPrimeMeridian** (char **=NULL) const

  Fetch prime meridian info.
- int **IsGeographic** () const

  Check if geographic coordinate system.
- int **IsProjected** () const

  Check if projected coordinate system.
- int **IsGeocentric** () const

  Check if geocentric coordinate system.
- int **IsLocal** () const

  Check if local coordinate system.
- int **IsVertical** () const

  Check if vertical coordinate system.
- int **IsCompound** () const

  Check if coordinate system is compound.
- int **IsSameGeogCS** (const **OGRSpatialReference** *) const

  Do the GeogCS'es match?
- int **IsSameVertCS** (const **OGRSpatialReference** *) const

  Do the VertCS'es match?
- int **IsSame** (const **OGRSpatialReference** *) const

  Do these two spatial references describe the same system ?
- void **Clear** ()

  Wipe current definition.
- OGRErr **SetLocalCS** (const char *)

  Set the user visible LOCAL_CS name.
- OGRErr **SetProjCS** (const char *)

  Set the user visible PROJCS name.

- OGRErr **SetProjection** (const char *)

  Set a projection name.
- OGRErr **SetGeocCS** (const char *pszGeocName)

  Set the user visible GEOCCS name.
- OGRErr **SetGeogCS** (const char *pszGeogName, const char *pszDatumName, const char *pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char *pszPMName=NULL, double dfPMOffset=0.0, const char *pszUnits=NULL, double dfConvertToRadians=0.0)

  Set geographic coordinate system.
- OGRErr **SetWellKnownGeogCS** (const char *)

  Set a GeogCS based on well known name.
- OGRErr **CopyGeogCSFrom** (const **OGRSpatialReference** *poSrcSRS)

  *Copy GEOGCS from another **OGRSpatialReference** (p. 230).*
- OGRErr **SetVertCS** (const char *pszVertCSName, const char *pszVertDatumName, int nVertDatumClass=2005)

  Set the user visible VERT_CS name.
- OGRErr **SetCompoundCS** (const char *pszName, const **OGRSpatialReference** *poHorizSRS, const **OGRSpatialReference** *poVertSRS)

  Setup a compound coordinate system.
- OGRErr **SetFromUserInput** (const char *)

  Set spatial reference from various text formats.
- OGRErr **SetTOWGS84** (double, double, double, double=0.0, double=0.0, double=0.0, double=0.0)

  Set the Bursa-Wolf conversion to WGS84.
- OGRErr **GetTOWGS84** (double *padfCoef, int nCoeff=7) const

  Fetch TOWGS84 parameters, if available.
- double **GetSemiMajor** (OGRErr *=NULL) const

  Get spheroid semi major axis.
- double **GetSemiMinor** (OGRErr *=NULL) const

  Get spheroid semi minor axis.
- double **GetInvFlattening** (OGRErr *=NULL) const

  Get spheroid inverse flattening.
- OGRErr **SetAuthority** (const char *pszTargetKey, const char *pszAuthority, int nCode)

  Set the authority for a node.
- OGRErr **AutoIdentifyEPSG** ()

  Set EPSG authority info if possible.
- const char * **GetAuthorityCode** (const char *pszTargetKey) const

  Get the authority code for a node.
- const char * **GetAuthorityName** (const char *pszTargetKey) const

  Get the authority name for a node.
- const char * **GetExtension** (const char *pszTargetKey, const char *pszName, const char *pszDefault=NULL) const

  Fetch extension value.
- OGRErr **SetExtension** (const char *pszTargetKey, const char *pszName, const char *pszValue)

  Set extension value.
- int **FindProjParm** (const char *pszParameter, const **OGR_SRSNode** *poPROJCS=NULL) const

  Return the child index of the named projection parameter on its parent PROJCS node.
- OGRErr **SetProjParm** (const char *, double)

  Set a projection parameter value.
- double **GetProjParm** (const char *, double=0.0, OGRErr *=NULL) const

  Fetch a projection parameter value.
- OGRErr **SetNormProjParm** (const char *, double)

  Set a projection parameter with a normalized value.

- double **GetNormProjParm** (const char *, double=0.0, OGRErr *=NULL) const

  Fetch a normalized projection parameter value.
- OGRErr **SetACEA** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetAE** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetBonne** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCEA** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCS** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEckert** (int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEquirectangular** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEquirectangular2** (double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGEOS** (double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGH** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetIGH** ()
- OGRErr **SetGS** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGaussSchreiberTMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGnomonic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetHOM** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)

  Set a Hotine Oblique Mercator projection using azimuth angle.
- OGRErr **SetHOM2PNO** (double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)

  Set a Hotine Oblique Mercator projection using two points on projection centerline.
- OGRErr **SetIWMPolyconic** (double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetKrovak** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLAEA** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC1SP** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCCB** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMC** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMollweide** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetNZMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOS** (double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOrthographic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetPolyconic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **SetPS** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetRobinson** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetSinusoidal** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStereographic** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetSOC** (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTM** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMVariant** (const char *pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMSO** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTPED** (double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetVDG** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetUTM** (int nZone, int bNorth=TRUE)

  Set UTM projection definition.
- int **GetUTMZone** (int *pbNorth=NULL) const

  Get utm zone information.
- OGRErr **SetWagner** (int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStatePlane** (int nZone, int bNAD83=TRUE, const char *pszOverrideUnitName=NULL, double dfOverrideUnit=0.0)

  Set State Plane projection definition.

Static Public Member Functions

- static void **DestroySpatialReference** (**OGRSpatialReference** *poSRS)

  OGRSpatialReference *(p. 230) destructor.*

11.69.1 Detailed Description

This class respresents a OpenGIS Spatial Reference System, and contains methods for converting between this object organization and well known text (WKT) format. This object is reference counted as one instance of the object is normally shared between many **OGRGeometry** (p. 126) objects.

Normally application code can fetch needed parameter values for this SRS using **GetAttrValue()** (p. 243), but in special cases the underlying parse tree (or **OGR_SRSNode** (p. 75) objects) can be accessed more directly.

See `the tutorial` for more information on how to use this class.

11.69.2 Constructor & Destructor Documentation

11.69.2.1 OGRSpatialReference::OGRSpatialReference (const char * *pszWKT* = NULL)

Constructor.

This constructor takes an optional string argument which if passed should be a WKT representation of an SRS. Passing this is equivalent to not passing it, and then calling **importFromWkt()** (p. 260) with the WKT string.

Note that newly created objects are given a reference count of one.

The C function **OSRNewSpatialReference()** (p. 507) does the same thing as this constructor.

Parameters

pszWKT	well known text definition to which the object should be initialized, or NULL (the default).

References importFromWkt().

11.69.2.2 OGRSpatialReference::~OGRSpatialReference ( ) [virtual]

OGRSpatialReference (p. 230) destructor.

The C function **OSRDestroySpatialReference()** (p. 499) does the same thing as this method. Preferred C++ method : **OGRSpatialReference::DestroySpatialReference()** (p. 237)

Deprecated

11.69.3 Member Function Documentation

11.69.3.1 OGRErr OGRSpatialReference::AutoIdentifyEPSG ( )

Set EPSG authority info if possible.

This method inspects a WKT definition, and adds EPSG authority nodes where an aspect of the coordinate system can be easily and safely corresponded with an EPSG identifier. In practice, this method will evolve over time. In theory it can add authority nodes for any object (ie. spheroid, datum, GEOGCS, units, and PROJCS) that could have an authority node. Mostly this is useful to inserting appropriate PROJCS codes for common formulations (like UTM n WGS84).

If it success the **OGRSpatialReference** (p. 230) is updated in place, and the method return OGRERR_NONE. If the method fails to identify the general coordinate system OGRERR_UNSUPPORTED_SRS is returned but no error message is posted via **CPLError()** (p. 334).

This method is the same as the C function **OSRAutoIdentifyEPSG()** (p. 497).

Returns

OGRERR_NONE or OGRERR_UNSUPPORTED_SRS.

References GetAuthorityCode(), GetAuthorityName(), GetUTMZone(), IsGeographic(), IsProjected(), and SetAuthority().

11.69.3.2 void OGRSpatialReference::Clear ( )

Wipe current definition.

Returns **OGRSpatialReference** (p. 230) to a state with no definition, as it exists when first created. It does not affect reference counts.

Referenced by CopyGeogCSFrom(), importFromERM(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromURN(), importFromWkt(), importFromWMSAUTO(), importFromXML(), SetCompoundCS(), SetFromUserInput(), SetGeogCS(), SetStatePlane(), and SetVertCS().

11.69.3.3 OGRSpatialReference * OGRSpatialReference::Clone ( ) const

Make a duplicate of this **OGRSpatialReference** (p. 230).

This method is the same as the C function **OSRClone()** (p. 498).

Returns

a new SRS, which becomes the responsibility of the caller.

References OGR_SRSNode::Clone().

Referenced by exportToPrettyWkt().

11.69.3.4 OGRSpatialReference * OGRSpatialReference::CloneGeogCS ( ) const

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. 230) object.

Returns

a new SRS, which becomes the responsibility of the caller.

References OGR_SRSNode::AddChild(), OGR_SRSNode::Clone(), CPLAtof(), GetAttrNode(), IsGeocentric(), SetAngularUnits(), and SetRoot().

Referenced by morphFromESRI().

11.69.3.5 OGRErr OGRSpatialReference::CopyGeogCSFrom (const OGRSpatialReference * *poSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. 230).

The GEOGCS information is copied into this **OGRSpatialReference** (p. 230) from another. If this object has a PROJCS root already, the GEOGCS is installed within it, otherwise it is installed as the root.

Parameters

poSrcSRS	the spatial reference to copy the GEOGCS information from.

Returns

OGRERR_NONE on success or an error code.

References Clear(), OGR_SRSNode::Clone(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::InsertChild(), IsGeocentric(), and SetRoot().

Referenced by importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), morphFromESRI(), SetGeogCS(), and SetWellKnownGeogCS().

11.69.3.6 int OGRSpatialReference::Dereference ( )

Decrements the reference count by one.

The method does the same thing as the C function **OSRDereference()** (p. 498).

Returns

the updated reference count.

Referenced by Release().

11.69.3.7 void OGRSpatialReference::DestroySpatialReference (OGRSpatialReference * *poSRS*) [static]

OGRSpatialReference (p. 230) destructor.

This static method will destroy a **OGRSpatialReference** (p. 230). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

This function is the same as **OSRDestroySpatialReference()** (p. 499)

Parameters

poSRS	the object to delete

Since

GDAL 1.7.0

11.69.3.8 int OGRSpatialReference::EPSGTreatsAsLatLong ( )

This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

Currently this returns TRUE for all geographic coordinate systems with an EPSG code set, and AXIS values set defining it as lat, long. Note that coordinate systems with an EPSG code and no axis settings will be assumed to not be lat/long.

FALSE will be returned for all coordinate systems that are not geographic, or that do not have an EPSG code set.

This method is the same as the C function **OSREPSGTreatsAsLatLong()** (p. 499).

Returns

TRUE or FALSE.

References GetAttrNode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and IsGeographic().

11.69.3.9 OGRErr OGRSpatialReference::exportToERM (char * *pszProj,* char * *pszDatum,* char * *pszUnits*)

Convert coordinate system to ERMapper format.

Parameters

pszProj	32 character buffer to receive projection name.
pszDatum	32 character buffer to recieve datum name.
pszUnits	32 character buffer to receive units name.

Returns

OGRERR_NONE on success, OGRERR_SRS_UNSUPPORTED if not translation is found, or OGRERR_FAILURE on other failures.

References GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetLinearUnits(), GetUTMZone(), importFromDict(), IsGeographic(), and IsProjected().

11.69.3.10 OGRErr OGRSpatialReference::exportToMICoordSys (char ** *ppszResult*) const

Export coordinate system in Mapinfo style CoordSys format.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

This method is the same as the C function **OSRExportToMICoordSys()** (p. 499).

Parameters

ppszResult	pointer to which dynamically allocated Mapinfo CoordSys definition will be assigned.

Returns

OGRERR_NONE on success, OGRERR_FAILURE on failure, OGRERR_UNSUPPORTED_OPERATION if MITAB library was not linked in.

11.69.3.11 OGRErr OGRSpatialReference::exportToPanorama (long * *piProjSys,* long * *piDatum,* long * *piEllips,* long * *piZone,* double * *padfPrjParams*) const

Export coordinate system in "Panorama" GIS projection definition.

This method is the equivalent of the C function OSRExportToPanorama().

Parameters

piProjSys	Pointer to variable, where the projection system code will be returned.
piDatum	Pointer to variable, where the coordinate system code will be returned.
piEllips	Pointer to variable, where the spheroid code will be returned.
piZone	Pointer to variable, where the zone for UTM projection system will be returned.
padfPrjParams	an existing 7 double buffer into which the projection parameters will be placed. See **importFromPanorama()** (p. 252) for the list of parameters.

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), and IsLocal().

11.69.3.12 OGRErr OGRSpatialReference::exportToPCI (char ** *ppszProj,* char ** *ppszUnits,* double ** *ppadfPrjParams*) const

Export coordinate system in PCI projection definition.

Converts the loaded coordinate reference system into PCI projection definition to the extent possible. The strings returned in ppszProj, ppszUnits and ppadfPrjParams array should be deallocated by the caller with CPLFree() when no longer needed.

LOCAL_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR_NONE.

This method is the equivelent of the C function **OSRExportToPCI()** (p. 499).

Parameters

ppszProj	pointer to which dynamically allocated PCI projection definition will be assigned.
ppszUnits	pointer to which dynamically allocated units definition will be assigned.
ppadfPrjParams	pointer to which dynamically allocated array of 17 projection parameters will be assigned. See **importFromPCI()** (p. 254) for the list of parameters.

Returns

OGRERR_NONE on success or an error code on failure.

References CPLAtof(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetInvFlattening(), GetLinearUnits(), GetNormProjParm(), GetSemiMajor(), GetTOWGS84(), GetUTMZone(), OGR_SRSNode::GetValue(), and IsLocal().

11.69.3.13 OGRErr OGRSpatialReference::exportToPrettyWkt (char ** *ppszResult,* int *bSimplify* = FALSE) const

Convert this SRS into a a nicely formatted WKT string for display to a person.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

This method is the same as the C function **OSRExportToPrettyWkt()** (p. 499).

Parameters

ppszResult	the resulting string is returned in this pointer.
bSimplify	TRUE if the AXIS, AUTHORITY and EXTENSION nodes should be stripped off

Returns

currently OGRERR_NONE is always returned, but the future it is possible error conditions will develop.

References Clone(), and OGR_SRSNode::StripNodes().

11.69.3.14 OGRErr OGRSpatialReference::exportToProj4 (char ** *ppszProj4*) const

Export coordinate system in PROJ.4 format.

Converts the loaded coordinate reference system into PROJ.4 format to the extent possible. The string returned in ppszProj4 should be deallocated by the caller with CPLFree() when no longer needed.

LOCAL_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR_NONE.

This method is the equivelent of the C function **OSRExportToProj4()** (p. 500).

Parameters

ppszProj4	pointer to which dynamically allocated PROJ.4 definition will be assigned.

Returns

OGRERR_NONE on success or an error code on failure.

References CPLAtof(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetExtension(), GetInvFlattening(), GetLinearUnits(), OGR_SRSNode::GetNode(), GetNormProjParm(), GetSemiMajor(), GetSemiMinor(), GetUTMZone(), OGR_SRSNode::GetValue(), IsGeocentric(), and IsGeographic().

11.69.3.15 OGRErr OGRSpatialReference::exportToUSGS (long * *piProjSys,* long * *piZone,* double ** *ppadfPrjParams,* long * *piDatum*) const

Export coordinate system in USGS GCTP projection definition.

This method is the equivalent of the C function **OSRExportToUSGS()** (p. 500).

Parameters

piProjSys	Pointer to variable, where the projection system code will be returned.
piZone	Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.
ppadfPrjParams	Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See **importFromUSGS()** (p. 256) for the list of parameters. Caller responsible to free this array.
piDatum	Pointer to variable, where the datum code will be returned.

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), and IsLocal().

11.69.3.16 OGRErr OGRSpatialReference::exportToWkt (char ** *ppszResult*) const

Convert this SRS into WKT format.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

This method is the same as the C function **OSRExportToWkt()** (p. 500).

Parameters

ppszResult	the resulting string is returned in this pointer.

Returns

currently OGRERR_NONE is always returned, but the future it is possible error conditions will develop.

References OGR_SRSNode::exportToWkt().

11.69.3.17 OGRErr OGRSpatialReference::exportToXML (char ** *ppszRawXML,* const char * *pszDialect* = NULL) const

Export coordinate system in XML format.

Converts the loaded coordinate reference system into XML format to the extent possible. The string returned in ppszRawXML should be deallocated by the caller with CPLFree() when no longer needed.

LOCAL_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR_NONE.

This method is the equivelent of the C function **OSRExportToXML()** (p. 500).

Parameters

ppszRawXML	pointer to which dynamically allocated XML definition will be assigned.
pszDialect	currently ignored. The dialect used is GML based.

Returns

OGRERR_NONE on success or an error code on failure.

References IsGeographic(), and IsProjected().

11.69.3.18 int OGRSpatialReference::FindProjParm (const char * *pszParameter,* const OGR_SRSNode * *poPROJCS =* NULL) const

Return the child index of the named projection parameter on its parent PROJCS node.

Parameters

pszParameter	projection parameter to look for
poPROJCS	projection CS node to look in. If NULL is passed, the PROJCS node of the SpatialReference object will be searched.

Returns

the child index of the named projection parameter. -1 on failure

References GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by GetProjParm(), and morphToESRI().

11.69.3.19 OGRErr OGRSpatialReference::Fixup ( )

Fixup as needed.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 230), and some imported WKT, are not valid according to the OGC CT specification. This method attempts to fill in any missing defaults that are required, and fixup ordering problems (using **OSRFixupOrdering()** (p. 500)) so that the resulting WKT is valid.

This method should be expected to evolve over time to as problems are discovered. The following are amoung the fixup actions this method will take:

- Fixup the ordering of nodes to match the BNF WKT ordering, using the **FixupOrdering()** (p. 242) method.
- Add missing linear or angular units nodes.

This method is the same as the C function **OSRFixup()** (p. 500).

Returns

OGRERR_NONE on success or an error code if something goes wrong.

References CPLAtof(), OGR_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), SetAngularUnits(), and SetLinearUnits().

Referenced by morphToESRI().

11.69.3.20 OGRErr OGRSpatialReference::FixupOrdering ( )

Correct parameter ordering to match CT Specification.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 230), and some imported WKT fail to maintain the order of parameters required according to the BNF definitions in the OpenGIS SF-SQL and CT Specifications. This method attempts to massage things back into the required order.

This method is the same as the C function **OSRFixupOrdering()** (p. 500).

Returns

OGRERR_NONE on success or an error code if something goes wrong.

Referenced by Fixup(), importFromEPSGA(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromUSGS(), and morphFromESRI().

11.69.3.21 double OGRSpatialReference::GetAngularUnits (char ** *ppszName* = NULL) const

Fetch angular geographic coordinate system units.

If no units are available, a value of "degree" and SRS_UA_DEGREE_CONV will be assumed. This method only checks directly under the GEOGCS node for units.

This method does the same thing as the C function **OSRGetAngularUnits()** (p. 501).

Parameters

ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. 230) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (p. 230) call.

Returns

the value to multiply by angular distances to transform them to radians.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by morphToESRI().

11.69.3.22 OGR_SRSNode * OGRSpatialReference::GetAttrNode (const char * *pszNodePath*)

Find named node in tree.

This method does a pre-order traversal of the node tree searching for a node with this exact value (case insensitive), and returns it. Leaf nodes are not considered, under the assumption that they are just attribute value nodes.

If a node appears more than once in the tree (such as UNIT for instance), the first encountered will be returned. Use GetNode() on a subtree to be more specific.

Parameters

pszNodePath	the name of the node to search for. May contain multiple components such as "GEOGCS|UNIT".

Returns

a pointer to the node found, or NULL if none.

References OGR_SRSNode::GetNode().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), EPSGTreatsAsLatLong(), exportToPCI(), exportToProj4(), FindProjParm(), Fixup(), GetAngularUnits(), GetAttrValue(), GetInvFlattening(), GetPrimeMeridian(), GetProjParm(), GetSemiMajor(), GetTargetLinearUnits(), GetTOWGS84(), importFromEPSG(), importFromESRI(), importFromProj4(), IsGeographic(), IsProjected(), IsSame(), IsVertical(), morphFromESRI(), morphToESRI(), SetAngularUnits(), SetAuthority(), SetGeocCS(), SetGeogCS(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetProjCS(), SetProjection(), SetProjParm(), SetStatePlane(), SetTargetLinearUnits(), SetTOWGS84(), and SetVertCS().

11.69.3.23 const char * OGRSpatialReference::GetAttrValue (const char * *pszNodeName,* int *iAttr* = 0) const

Fetch indicated attribute of named node.

This method uses **GetAttrNode()** (p. 243) to find the named node, and then extracts the value of the indicated child. Thus a call to GetAttrValue("UNIT",1) would return the second child of the UNIT node, which is normally the length of the linear unit in meters.

This method does the same thing as the C function **OSRGetAttrValue()** (p. 501).

Parameters

pszNodeName	the tree node to look for (case insensitive).
iAttr	the child of the node to fetch (zero based).

Returns

the requested value, or NULL if it fails for any reason.

References GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), IsSame(), IsSameGeogCS(), IsSameVertCS(), morphFromESRI(), morphToESRI(), and SetUTM().

11.69.3.24 const char * OGRSpatialReference::GetAuthorityCode (const char * *pszTargetKey*) const

Get the authority code for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the code value.

While in theory values may be non-numeric, for the EPSG authority all code values should be integral.

This method is the same as the C function **OSRGetAuthorityCode()** (p. 501).

Parameters

pszTargetKey	the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS|UNIT" or NULL to search for an authority node on the root element.

Returns

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR_SRSNode::FindChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), exportToERM(), exportToPCI(), exportToProj4(), and morphToESRI().

11.69.3.25 const char * OGRSpatialReference::GetAuthorityName (const char * *pszTargetKey*) const

Get the authority name for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the authority name value.

The most common authority is "EPSG".

This method is the same as the C function **OSRGetAuthorityName()** (p. 501).

Parameters

pszTargetKey	the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS|UNIT" or NULL to search for an authority node on the root element.

Returns

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR_SRSNode::FindChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToPCI(), exportToProj4(), importFromEPSGA(), and morphToESRI().

11.69.3.26 const char * OGRSpatialReference::GetAxis (const char * *pszTargetKey,* int *iAxis,* OGRAxisOrientation * *peOrientation*) const

Fetch the orientation of one axis.

Fetches the the request axis (iAxis - zero based) from the indicated portion of the coordinate system (pszTargetKey) which should be either "GEOGCS" or "PROJCS".

No CPLError is issued on routine failures (such as not finding the AXIS).

This method is equivalent to the C function **OSRGetAxis()** (p. 501).

Parameters

pszTargetKey	the coordinate system part to query ("PROJCS" or "GEOGCS").
iAxis	the axis to query (0 for first, 1 for second).
peOrientation	location into which to place the fetch orientation, may be NULL.

Returns

the name of the axis or NULL on failure.

References OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

11.69.3.27 const char * OGRSpatialReference::GetExtension (const char * *pszTargetKey,* const char * *pszName,* const char * *pszDefault* = `NULL`) const

Fetch extension value.

Fetch the value of the named EXTENSION item for the identified target node.

Parameters

pszTargetKey	the name or path to the parent node of the EXTENSION.
pszName	the name of the extension being fetched.
pszDefault	the value to return if the extension is not found.

Returns

node value if successful or pszDefault on failure.

References OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToProj4().

11.69.3.28 double OGRSpatialReference::GetInvFlattening (OGRErr * *pnErr* = `NULL`) const

Get spheroid inverse flattening.

This method does the same thing as the C function **OSRGetInvFlattening()** (p. 501).

Parameters

pnErr	if non-NULL set to OGRERR_FAILURE if no inverse flattening can be found.

Returns

inverse flattening, or SRS_WGS84_INVFLATTENING if it can't be found.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

11.69.3.29 double OGRSpatialReference::GetLinearUnits (char ** *ppszName* = NULL) const

Fetch linear projection units.

If no units are available, a value of "Meters" and 1.0 will be assumed. This method only checks directly under the PROJCS, GEOCCS or LOCAL_CS node for units.

This method does the same thing as the C function **OSRGetLinearUnits()** (p. 502)/

Parameters

ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. 230) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (p. 230) call.

Returns

the value to multiply by linear distances to transform them to meters.

References GetTargetLinearUnits().

Referenced by exportToERM(), exportToPCI(), exportToProj4(), importFromESRI(), importFromProj4(), IsSame(), morphToESRI(), SetLinearUnitsAndUpdateParameters(), and SetStatePlane().

11.69.3.30 double OGRSpatialReference::GetNormProjParm (const char * *pszName*, double *dfDefaultValue* = 0.0, OGRErr * *pnErr* = NULL) const

Fetch a normalized projection parameter value.

This method is the same as **GetProjParm()** (p. 247) except that the value of the parameter is "normalized" into degrees or meters depending on whether it is linear or angular.

This method is the same as the C function **OSRGetNormProjParm()** (p. 502).

Parameters

pszName	the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. 490).
dfDefaultValue	the value to return if this parameter doesn't exist.
pnErr	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References GetProjParm().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), morphToESRI(), and SetStatePlane().

11.69.3.31 double OGRSpatialReference::GetPrimeMeridian (char ** *ppszName* = NULL) const

Fetch prime meridian info.

Returns the offset of the prime meridian from greenwich in degrees, and the prime meridian name (if requested). If no PRIMEM value exists in the coordinate system definition a value of "Greenwich" and an offset of 0.0 is assumed.

If the prime meridian name is returned, the pointer is to an internal copy of the name. It should not be freed, altered or depended on after the next OGR call.

This method is the same as the C function **OSRGetPrimeMeridian()** (p. 502).

Parameters

ppszName	return location for prime meridian name. If NULL, name is not returned.

Returns

the offset to the GEOGCS prime meridian from greenwich in decimal degrees.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

11.69.3.32 double OGRSpatialReference::GetProjParm (const char * *pszName*, double *dfDefaultValue* = 0.0, OGRErr * *pnErr* = NULL) const

Fetch a projection parameter value.

NOTE: This code should be modified to translate non degree angles into degrees based on the GEOGCS unit. This has not yet been done.

This method is the same as the C function **OSRGetProjParm()** (p. 502).

Parameters

pszName	the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. 490).
dfDefaultValue	the value to return if this parameter doesn't exist.
pnErr	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References CPLAtof(), FindProjParm(), GetAttrNode(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by GetNormProjParm(), GetUTMZone(), importFromProj4(), IsSame(), morphFromESRI(), morphToESRI(), and SetLinearUnitsAndUpdateParameters().

11.69.3.33 int OGRSpatialReference::GetReferenceCount ( ) const [inline]

Fetch current reference count.

Returns

the current reference count.

11.69.3.34 double OGRSpatialReference::GetSemiMajor (OGRErr * *pnErr* = NULL) const

Get spheroid semi major axis.

This method does the same thing as the C function **OSRGetSemiMajor()** (p. 502).

Parameters

pnErr	if non-NULL set to OGRERR_FAILURE if semi major axis can be found.

Returns

semi-major axis, or SRS_WGS84_SEMIMAJOR if it can't be found.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

11.69.3.35 double OGRSpatialReference::GetSemiMinor (OGRErr * *pnErr* = NULL) const

Get spheroid semi minor axis.

This method does the same thing as the C function **OSRGetSemiMinor()** (p. 502).

Parameters

pnErr	if non-NULL set to OGRERR_FAILURE if semi minor axis can be found.

Returns

semi-minor axis, or WGS84 semi minor if it can't be found.

References GetInvFlattening(), and GetSemiMajor().

Referenced by exportToProj4().

11.69.3.36 double OGRSpatialReference::GetTargetLinearUnits (const char * *pszTargetKey*, char ** *ppszName* = NULL) const

Fetch linear units for target.

If no units are available, a value of "Meters" and 1.0 will be assumed.

This method does the same thing as the C function **OSRGetTargetLinearUnits()** (p. 503)/

Parameters

pszTargetKey	the key to look on. ie. "PROJCS" or "VERT_CS".
ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. 230) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (p. 230) call.

Returns

the value to multiply by linear distances to transform them to meters.

Since

OGR 1.9.0

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and IsVertical().

Referenced by GetLinearUnits().

11.69.3.37 OGRErr OGRSpatialReference::GetTOWGS84 (double * *padfCoeff,* int *nCoeffCount* = 7) const

Fetch TOWGS84 parameters, if available.

Parameters

padfCoeff	array into which up to 7 coefficients are placed.
nCoeffCount	size of padfCoeff - defaults to 7.

Returns

OGRERR_NONE on success, or OGRERR_FAILURE if there is no TOWGS84 node available.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToPCI(), and IsSameGeogCS().

11.69.3.38 int OGRSpatialReference::GetUTMZone (int * *pbNorth* = NULL) const

Get utm zone information.

This is the same as the C function **OSRGetUTMZone()** (p. 503).

In SWIG bindings (Python, Java, etc) the **GetUTMZone()** (p. 249) method returns a zone which is negative in the southern hemisphere instead of having the pbNorth flag used in the C and C++ interface.

Parameters

pbNorth	pointer to in to set to TRUE if northern hemisphere, or FALSE if southern.

Returns

UTM zone number or zero if this isn't a UTM definition.

References GetAttrValue(), GetNormProjParm(), and GetProjParm().

Referenced by AutoIdentifyEPSG(), exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and morphToESRI().

11.69.3.39 OGRErr OGRSpatialReference::importFromDict (const char * *pszDictFile,* const char * *pszCode*)

Read SRS from WKT dictionary.

This method will attempt to find the indicated coordinate system identity in the indicated dictionary file. If found, the WKT representation is imported and used to initialize this **OGRSpatialReference** (p. 230).

More complete information on the format of the dictionary files can be found in the epsg.wkt file in the GDAL data tree. The dictionary files are searched for in the "GDAL" domain using CPLFindFile(). Normally this results in searching /usr/local/share/gdal or somewhere similar.

This method is the same as the C function OSRImportFromDict().

Parameters

pszDictFile	the name of the dictionary file to load.
pszCode	the code to lookup in the dictionary.

Returns

OGRERR_NONE on success, or OGRERR_SRS_UNSUPPORTED if the code isn't found, and OGRERR_SRS_FAILURE if something more dramatic goes wrong.

References importFromWkt().

Referenced by exportToERM(), importFromEPSGA(), importFromERM(), and SetFromUserInput().

11.69.3.40 OGRErr OGRSpatialReference::importFromEPSG (int *nCode*)

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code. The coordinate system definitions are normally read from the EPSG derived support files such as pcs.csv, gcs.csv, pcs.override.csv, gcs.override.csv and falling back to search for a PROJ.4 epsg init file or a definition in epsg.wkt.

These support files are normally searched for in /usr/local/share/gdal or in the directory identified by the GDAL_DATA configuration option. See CPLFindFile() for details.

This method is relatively expensive, and generally involves quite a bit of text file scanning. Reasonable efforts should be made to avoid calling it many times for the same coordinate system.

This method is similar to **importFromEPSGA()** (p. 250) except that EPSG preferred axis ordering will *not* be applied for geographic coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long contrary to typical GIS use).

This method is the same as the C function **OSRImportFromEPSG()** (p. 503).

Parameters

nCode	a GCS or PCS code from the horizontal coordinate system table.

Returns

OGRERR_NONE on success, or an error code on failure.

References GetAttrNode(), importFromEPSGA(), and OGR_SRSNode::StripNodes().

Referenced by importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), morphFromESRI(), SetFromUserInput(), SetStatePlane(), and SetWellKnownGeogCS().

11.69.3.41 OGRErr OGRSpatialReference::importFromEPSGA (int *nCode*)

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code.

This method is similar to **importFromEPSG()** (p. 250) except that EPSG preferred axis ordering *will* be applied for geographic coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long contrary to typical GIS use). See **OGRSpatialReference::importFromEPSG()** (p. 250) for more details on operation of this method.

This method is the same as the C function **OSRImportFromEPSGA()** (p. 503).

Parameters

nCode	a GCS or PCS code from the horizontal coordinate system table.

Returns

OGRERR_NONE on success, or an error code on failure.

References FixupOrdering(), GetAuthorityName(), importFromDict(), importFromProj4(), IsGeographic(), IsProjected(), and SetAuthority().

Referenced by importFromEPSG(), SetFromUserInput(), and SetWellKnownGeogCS().

11.69.3.42 OGRErr OGRSpatialReference::importFromERM (const char * *pszProj,* const char * *pszDatum,* const char * *pszUnits*)

Create OGR WKT from ERMapper projection definitions.

Generates an **OGRSpatialReference** (p. 230) definition from an ERMapper datum and projection name. Based on the ecw_cs.wkt dictionary file from gdal/data.

Parameters

pszProj	the projection name, such as "NUTM11" or "GEOGRAPHIC".
pszDatum	the datum name, such as "NAD83".
pszUnits	the linear units "FEET" or "METERS".

Returns

OGRERR_NONE on success or OGRERR_UNSUPPORTED_SRS if not found.

References Clear(), CopyGeogCSFrom(), importFromDict(), importFromEPSG(), IsLocal(), and SetLinearUnits().

11.69.3.43 OGRErr OGRSpatialReference::importFromESRI (char ** *papszPrj*)

Import coordinate system from ESRI .prj format(s).

This function will read the text loaded from an ESRI .prj file, and translate it into an **OGRSpatialReference** (p. 230) definition. This should support many (but by no means all) old style (Arc/Info 7.x) .prj files, as well as the newer pseudo-OGC WKT .prj files. Note that new style .prj files are in OGC WKT format, but require some manipulation to correct datum names, and units on some projection parameters. This is addressed within **importFromESRI()** (p. 251) by an automatical call to **morphFromESRI()** (p. 263).

Currently only GEOGRAPHIC, UTM, STATEPLANE, GREATBRITIAN_GRID, ALBERS, EQUIDISTANT_CONIC, TRANSVERSE (mercator), POLAR, MERCATOR and POLYCONIC projections are supported from old style files.

At this time there is no equivelent exportToESRI() method. Writing old style .prj files is not supported by **OGRSpatialReference** (p. 230). However the **morphToESRI()** (p. 264) and **exportToWkt()** (p. 241) methods can be used to generate output suitable to write to new style (Arc 8) .prj files.

This function is the equilvelent of the C function **OSRImportFromESRI()** (p. 503).

Parameters

papszPrj	NULL terminated list of strings containing the definition.

Returns

OGRERR_NONE on success or an error code in case of failure.

References CopyGeogCSFrom(), OGR_SRSNode::DestroyChild(), GetAttrNode(), GetLinearUnits(), importFromEPSG(), importFromWkt(), IsLocal(), IsProjected(), morphFromESRI(), SetACEA(), SetEC(), SetLAEA(), SetLCC(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMercator(), SetPolyconic(), SetPS(), SetStatePlane(), SetTM(), SetUTM(), and SetWellKnownGeogCS().

11.69.3.44 OGRErr OGRSpatialReference::importFromMICoordSys (const char * *pszCoordSys*)

Import Mapinfo style CoordSys definition.

The **OGRSpatialReference** (p. 230) is initialized from the passed Mapinfo style CoordSys definition string.

This method is the equivalent of the C function **OSRImportFromMICoordSys()** (p. 504).

Parameters

pszCoordSys	Mapinfo style CoordSys definition string.

Returns

OGRERR_NONE on success, OGRERR_FAILURE on failure, OGRERR_UNSUPPORTED_OPERATION if MITAB library was not linked in.

11.69.3.45 OGRErr OGRSpatialReference::importFromOzi (const char * *pszDatum,* const char * *pszProj,* const char * *pszProjParms*)

Import coordinate system from OziExplorer projection definition.

This method will import projection definition in style, used by OziExplorer software.

This function is the equivalent of the C function OSRImportFromOzi().

Parameters

pszDatum	Datum string. This is a fifth string in the OziExplorer .MAP file.
pszProj	Projection string. Search for line starting with "Map Projection" name in the OziExplorer .MAP file and supply it as a whole in this parameter.
pszProjParms	String containing projection parameters. Search for "Projection Setup" name in the OziExplorer .MAP file and supply it as a whole in this parameter.

Returns

OGRERR_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), CPLAtof(), FixupOrdering(), importFromEPSG(), IsLocal(), IsProjected(), SetACEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMercator(), SetSinusoidal(), SetTM(), and SetWellKnownGeogCS().

11.69.3.46 OGRErr OGRSpatialReference::importFromPanorama (long *iProjSys,* long *iDatum,* long *iEllips,* double * *padfPrjParams*)

Import coordinate system from "Panorama" GIS projection definition.

This method will import projection definition in style, used by "Panorama" GIS.

This function is the equivalent of the C function OSRImportFromPanorama().

Parameters

iProjSys	Input projection system code, used in GIS "Panorama". Supported Projections 1: Gauss-Kruger (Transverse Mercator) 2: Lambert Conformal Conic 2SP 5: Stereographic 6: Azimuthal Equidistant (Postel) 8: Mercator 10: Polyconic 13: Polar Stereographic 15: Gnomonic 17: Universal Transverse Mercator (UTM) 18: Wagner I (Kavraisky VI) 19: Mollweide 20: Equidistant Conic 24: Lambert Azimuthal Equal Area 27: Equirectangular 28: Cylindrical Equal Area (Lambert) 29: International Map of the World Polyconic
iDatum	Input coordinate system. Supported Datums 1: Pulkovo, 1942 2: WGS, 1984 3: OSGB 1936 (British National Grid) 9: Pulkovo, 1995
iEllips	Input spheroid. Supported Spheroids 1: Krassovsky, 1940 2: WGS, 1972 3: International, 1924 (Hayford, 1909) 4: Clarke, 1880 5: Clarke, 1866 (NAD1927) 6: Everest, 1830 7: Bessel, 1841 8: Airy, 1830 9: WGS, 1984 (GPS)
padfPrjParams	Array of 8 coordinate system parameters:

```
[0]  Latitude of the first standard parallel (radians)
[1]  Latitude of the second standard parallel (radians)
[2]  Latitude of center of projection (radians)
[3]  Longitude of center of projection (radians)
[4]  Scaling factor
[5]  False Easting
[6]  False Northing
[7]  Zone number
```

Particular projection uses different parameters, unused ones may be set to zero. If NULL supplied instead of array pointer default values will be used (i.e., zeroes).

Returns

OGRERR_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), FixupOrdering(), importFromEPSG(), IsLocal(), IsProjected(), SetAE(), SetAuthority(), SetCEA(), SetEC(), SetEquirectangular(), SetGeogCS(), SetGnomonic(), SetIWMPolyconic(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMercator(), SetMollweide(), SetPolyconic(), SetPS(), SetStereographic(), SetTM(), SetUTM(), SetWagner(), and SetWellKnownGeogCS().

11.69.3.47 OGRErr OGRSpatialReference::importFromPCI (const char * *pszProj*, const char * *pszUnits* = NULL, double * *padfPrjParams* = NULL)

Import coordinate system from PCI projection definition.

PCI software uses 16-character string to specify coordinate system and datum/ellipsoid. You should supply at least this string to the **importFromPCI()** (p. 254) function.

This function is the equivalent of the C function **OSRImportFromPCI()** (p. 504).

Parameters

pszProj	NULL terminated string containing the definition. Looks like "ppppppppppp Ennn" or "ppppppppppp Dnnn", where "ppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" — a datum code.
pszUnits	Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.
padfPrjParams	Array of 17 coordinate system parameters:

[0] Spheroid semi major axis [1] Spheroid semi minor axis [2] Reference Longitude [3] Reference Latitude [4] First Standard Parallel [5] Second Standard Parallel [6] False Easting [7] False Northing [8] Scale Factor [9] Height above sphere surface [10] Longitude of 1st point on center line [11] Latitude of 1st point on center line [12] Longitude of 2nd point on center line [13] Latitude of 2nd point on center line [14] Azimuth east of north for center line [15] Landsat satellite number [16] Landsat path number

Particular projection uses different parameters, unused ones may be set to zero. If NULL suppliet instead of array pointer default values will be used (i.e., zeroes).

Returns

OGRERR_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), CPLAtof(), FixupOrdering(), importFromEPSG(), IsGeographic(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetAngularUnits(), SetAuthority(), SetCS(), SetEC(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMC(), SetMercator(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetTOWGS84(), SetUTM(), and SetVDG().

11.69.3.48 OGRErr OGRSpatialReference::importFromProj4 (const char * *pszProj4*)

Import PROJ.4 coordinate string.

The **OGRSpatialReference** (p. 230) is initialized from the passed PROJ.4 style coordinate system string. In addition to many +proj formulations which have OGC equivelents, it is also possible to import "+init=epsg:n" style definitions. These are passed to **importFromEPSG()** (p. 250). Other init strings (such as the state plane zones) are not currently supported.

Example: pszProj4 = "+proj=utm +zone=11 +datum=WGS84"

Some parameters, such as grids, recognised by PROJ.4 may not be well understood and translated into the **OGRSpatialReference** (p. 230) model. It is possible to add the +wktext parameter which is a special keyword that OGR

recognises as meaning "embed the entire PROJ.4 string in the WKT and use it literally when converting back to PROJ.4 format".

For example: "+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000 +y_0=6023150 +ellps=intl +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"

will be translated as :

```
PROJCS["unnamed",
   GEOGCS["International 1909 (Hayford)",
       DATUM["unknown",
           SPHEROID["intl",6378388,297]],
       PRIMEM["Greenwich",0],
       UNIT["degree",0.0174532925199433]],
   PROJECTION["New_Zealand_Map_Grid"],
   PARAMETER["latitude_of_origin",-41],
   PARAMETER["central_meridian",173],
   PARAMETER["false_easting",2510000],
   PARAMETER["false_northing",6023150],
   UNIT["Meter",1],
   EXTENSION["PROJ4","+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000
              +y_0=6023150 +ellps=intl  +units=m +nadgrids=nzgd2kgrid0005.gsb
       +wktext"]]
```

This method is the equivalent of the C function **OSRImportFromProj4()** (p. 504).

Parameters

pszProj4	the PROJ.4 style string.

Returns

OGRERR_NONE on success or OGRERR_CORRUPT_DATA on failure.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), CopyGeogCSFrom(), CPLAtof(), CPLAtofM(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), OGR_SRSNode::GetNode(), GetProjParm(), OGR_SRSNode::GetValue(), importFromEPSG(), IsGeocentric(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetExtension(), SetGaussSchreiberTMercator(), SetGeocCS(), SetGeogCS(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetIGH(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetMC(), SetMercator(), SetMollweide(), SetNode(), SetNormProjParm(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetTM(), SetTMSO(), SetTOWGS84(), SetTPED(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

Referenced by importFromEPSGA(), and SetFromUserInput().

11.69.3.49 OGRErr OGRSpatialReference::importFromUrl (const char * *pszUrl*)

Set spatial reference from a URL.

This method will download the spatial reference at a given URL and feed it into SetFromUserInput for you.

This method does the same thing as the **OSRImportFromUrl()** (p. 504) function.

Parameters

pszUrl	text definition to try to deduce SRS from.

Returns

OGRERR_NONE on success, or an error code with the curl error message if it is unable to dowload data.

References CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHTTPResult::pabyData, CPLHTTPResult::pszErrBuf, and SetFromUserInput().

Referenced by SetFromUserInput().

11.69.3.50 OGRErr OGRSpatialReference::importFromURN (const char * *pszURN*)

Initialize from OGC URN.

Initializes this spatial reference from a coordinate system defined by an OGC URN prefixed with "urn:ogc:def:crs:" per recommendation paper 06-023r1. Currently EPSG and OGC authority values are supported, including OGC auto codes, but not including CRS1 or CRS88 (NAVD88).

This method is also support through **SetFromUserInput()** (p. 268) which can normally be used for URNs.

Parameters

pszURN	the urn string.

Returns

OGRERR_NONE on success or an error code.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), and SetNode().

Referenced by SetFromUserInput().

11.69.3.51 OGRErr OGRSpatialReference::importFromUSGS (long *iProjSys,* long *iZone,* double * *padfPrjParams,* long *iDatum,* int *bAnglesInPackedDMSFormat* = TRUE)

Import coordinate system from USGS projection definition.

This method will import projection definition in style, used by USGS GCTP software. GCTP operates on angles in packed DMS format (see **CPLDecToPackedDMS()** (p. 316) function for details), so all angle values (latitudes, longitudes, azimuths, etc.) specified in the padfPrjParams array should be in the packed DMS format, unless bAnglesInPackedDMSFormat is set to FALSE.

This function is the equivalent of the C function **OSRImportFromUSGS()** (p. 504). Note that the bAnglesInPackedDMSFormat parameter is only present in the C++ method. The C function assumes bAnglesInPackedFormat = TRUE.

Parameters

iProjSys	Input projection system code, used in GCTP.
iZone	Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. iZone ignored for all other projections.
padfPrjParams	Array of 15 coordinate system parameters. These parameters differs for different projections.

<h4>Projection Transformation Package Projection Parameters</h4>

Code & Projection Id	Array Element 0	1	2	3	4	5	6	7
0 Geographic								
1 U T M	Lon/Z	Lat/Z						
2 State Plane								
3 Albers Equal Area	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
5 Mercator	SMajor	SMinor			CentMer	TrueScale	FE	FN
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN
8 Equid. Conic A	SMajor	SMinor	STDPAR		CentMer	OriginLat	FE	FN
Equid. Conic B	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN
10 Stereographic	Sphere				CentLon	CenterLat	FE	FN
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
12 Azimuthal	Sphere				CentLon	CenterLat	FE	FN
13 Gnomonic	Sphere				CentLon	CenterLat	FE	FN
14 Orthographic	Sphere				CentLon	CenterLat	FE	FN
15 Gen. Vert. Near Per	Sphere		Height		CentLon	CenterLat	FE	FN
16 Sinusoidal	Sphere				CentMer		FE	FN
17 Equirectangular	Sphere				CentMer	TrueScale	FE	FN
18 Miller Cylindrical	Sphere				CentMer		FE	FN
19 Van der Grinten	Sphere				CentMer	OriginLat	FE	FN
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN
Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
21 Robinson	Sphere				CentMer		FE	FN
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLong		FE	FN
Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN
23 Alaska Conformal	SMajor	SMinor					FE	FN
24 Interrupted Goode	Sphere							
25 Mollweide	Sphere				CentMer		FE	FN
26 Interrupt Mollweide	Sphere							
27 Hammer	Sphere				CentMer		FE	FN
28 Wagner IV	Sphere				CentMer		FE	FN
29 Wagner VII	Sphere				CentMer		FE	FN
30 Oblated Equal Area	Sphere		Shapem	Shapen	CentLon	CenterLat	FE	FN

---------- | Array Element | Code & Projection Id |---------- | 8 | 9 | 10 | 11 | 12 | ---------- 0 Geographic | | | | | | 1 U T M | | | | | | 2 State Plane | | | | | | 3 Albers Equal Area | | | | | | 4 Lambert Conformal C | | | | | | 5 Mercator | | | | | | 6 Polar Stereographic | | | | | | 7 Polyconic | | | | | | 8 Equid. Conic A |zero | | | | | Equid. Conic B |one | | | | | 9 Transverse Mercator | | | | | | 10 Stereographic | | | | | | 11 Lambert Azimuthal | | | | | | 12 Azimuthal | | | | | | 13 Gnomonic | | | | | | 14 Orthographic | | | | | | 15 Gen. Vert. Near Per | | | | | | 16 Sinusoidal | | | | | | 17 Equirectangular | | | | | | 18 Miller Cylindrical | | | | | | 19 Van der Grinten | | | | | | 20 Hotin Oblique Merc A |Long1|Lat1|Long2|Lat2|zero| Hotin Oblique Merc B | | | |one | 21 Robinson | | | | | | 22 Space Oblique Merc A |PSRev|LRat|PFlag| |zero| Space Oblique Merc B | | | |one | 23 Alaska Conformal | | | | | | 24 Interrupted Goode | | | | | | 25 Mollweide | | | | | | 26 Interrupt Mollweide | | | | | | 27 Hammer | | | | | | 28 Wagner IV | | | | | | 29 Wagner VII | | | | | | 30 Oblated Equal Area |Angle| | | | ----------

where

```
   Lon/Z     Longitude of any point in the UTM zone or zero.  If zero,
             a zone code must be specified.
   Lat/Z     Latitude of any point in the UTM zone or zero.  If zero, a
             zone code must be specified.
   SMajor    Semi-major axis of ellipsoid.  If zero, Clarke 1866 in meters
             is assumed.
   SMinor    Eccentricity squared of the ellipsoid if less than zero,
             if zero, a spherical form is assumed, or if greater than
             zero, the semi-minor axis of ellipsoid.
   Sphere    Radius of reference sphere.  If zero, 6370997 meters is used.
   STDPAR    Latitude of the standard parallel
   STDPR1    Latitude of the first standard parallel
   STDPR2    Latitude of the second standard parallel
   CentMer   Longitude of the central meridian
   OriginLat Latitude of the projection origin
   FE        False easting in the same units as the semi-major axis
   FN        False northing in the same units as the semi-major axis
   TrueScale Latitude of true scale
   LongPol   Longitude down below pole of map
   Factor    Scale factor at central meridian (Transverse Mercator) or
             center of projection (Hotine Oblique Mercator)
   CentLon   Longitude of center of projection
   CenterLat Latitude of center of projection
   Height    Height of perspective point
   Long1     Longitude of first point on center line (Hotine Oblique
             Mercator, format A)
   Long2     Longitude of second point on center line (Hotine Oblique
             Mercator, format A)
   Lat1      Latitude of first point on center line (Hotine Oblique
             Mercator, format A)
   Lat2      Latitude of second point on center line (Hotine Oblique
             Mercator, format A)
   AziAng    Azimuth angle east of north of center line (Hotine Oblique
             Mercator, format B)
   AzmthPt   Longitude of point on central meridian where azimuth occurs
             (Hotine Oblique Mercator, format B)
   IncAng    Inclination of orbit at ascending node, counter-clockwise
             from equator (SOM, format A)
   AscLong   Longitude of ascending orbit at equator (SOM, format A)
   PSRev     Period of satellite revolution in minutes (SOM, format A)
   LRat      Landsat ratio to compensate for confusion at northern end
             of orbit (SOM, format A -- use 0.5201613)
   PFlag     End of path flag for Landsat:  0 = start of path,
             1 = end of path (SOM, format A)
   Satnum    Landsat Satellite Number (SOM, format B)
   Path      Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and
             WRS-2 for Landsat 4, 5 and 6.)  (SOM, format B)
   Shapem    Oblated Equal Area oval shape parameter m
   Shapen    Oblated Equal Area oval shape parameter n
   Angle     Oblated Equal Area oval rotation angle

Array elements 13 and 14 are set to zero. All array elements with blank
fields are set to zero too.
```

Parameters

iDatum	Input spheroid.

If the datum code is negative, the first two values in the parameter array (parm) are used to define the values as follows:

- If padfPrjParams[0] is a non-zero value and padfPrjParams[1] is greater than one, the semimajor axis is set

to padfPrjParams[0] and the semiminor axis is set to padfPrjParams[1].

- If padfPrjParams[0] is nonzero and padfPrjParams[1] is greater than zero but less than or equal to one, the semimajor axis is set to padfPrjParams[0] and the semiminor axis is computed from the eccentricity squared value padfPrjParams[1]:

  semiminor = sqrt(1.0 - ES) ∗ semimajor

  where

  ES = eccentricity squared

- If padfPrjParams[0] is nonzero and padfPrjParams[1] is equal to zero, the semimajor axis and semiminor axis are set to padfPrjParams[0].

- If padfPrjParams[0] equals zero and padfPrjParams[1] is greater than zero, the default Clarke 1866 is used to assign values to the semimajor axis and semiminor axis.

- If padfPrjParams[0] and padfPrjParams[1] equals zero, the semimajor axis is set to 6370997.0 and the semiminor axis is set to zero.

If a datum code is zero or greater, the semimajor and semiminor axis are defined by the datum code as found in the following table:

```
<h4>Supported Datums</h4>

 0: Clarke 1866 (default)
 1: Clarke 1880
 2: Bessel
 3: International 1967
 4: International 1909
 5: WGS 72
 6: Everest
 7: WGS 66
 8: GRS 1980/WGS 84
 9: Airy
10: Modified Everest
11: Modified Airy
12: Walbeck
13: Southeast Asia
14: Australian National
15: Krassovsky
16: Hough
17: Mercury 1960
18: Modified Mercury 1968
19: Sphere of Radius 6370997 meters
```

Parameters

bAnglesIn-PackedDMS-Format	TRUE if the angle values specified in the padfPrjParams array should be in the packed DMS format

Returns

OGRERR_NONE on success or an error code in case of failure.

References FixupOrdering(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetAuthority(), SetEC(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), SetMercator(), SetMollweide(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

11.69.3.52 OGRErr OGRSpatialReference::importFromWkt (char ** *ppszInput*)

Import from WKT string.

This method will wipe the existing SRS definition, and reassign it based on the contents of the passed WKT string. Only as much of the input string as needed to construct this SRS is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

This method is the same as the C function **OSRImportFromWkt()** (p. 504).

Parameters

ppszInput	Pointer to pointer to input. The pointer is updated to point to remaining unused input text.

Returns

OGRERR_NONE if import succeeds, or OGRERR_CORRUPT_DATA if it fails for any reason.

References OGR_SRSNode::AddChild(), Clear(), and OGR_SRSNode::importFromWkt().

Referenced by importFromDict(), importFromESRI(), OGRSpatialReference(), OSRNewSpatialReference(), SetFromUserInput(), and SetWellKnownGeogCS().

11.69.3.53 OGRErr OGRSpatialReference::importFromWMSAUTO (const char * *pszDefinition*)

Initialize from WMSAUTO string.

Note that the WMS 1.3 specification does not include the units code, while apparently earlier specs do. We try to guess around this.

Parameters

pszDefinition	the WMSAUTO string

Returns

OGRERR_NONE on success or an error code.

References Clear(), CPLAtof(), SetAuthority(), SetEquirectangular(), SetLinearUnits(), SetMollweide(), SetOrthographic(), SetTM(), SetUTM(), and SetWellKnownGeogCS().

Referenced by SetFromUserInput().

11.69.3.54 OGRErr OGRSpatialReference::importFromXML (const char * *pszXML*)

Import coordinate system from XML format (GML only currently).

This method is the same as the C function **OSRImportFromXML()** (p. 505)

Parameters

pszXML	XML string to import

Returns

OGRERR_NONE on success or OGRERR_CORRUPT_DATA on failure.

References Clear(), CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by SetFromUserInput().

11.69.3.55 int OGRSpatialReference::IsCompound ( ) const

Check if coordinate system is compound.

This method is the same as the C function **OSRIsCompound()** (p. 505).

Returns

TRUE if this is rooted with a COMPD_CS node.

References OGR_SRSNode::GetValue().

11.69.3.56 int OGRSpatialReference::IsGeocentric ( ) const

Check if geocentric coordinate system.

This method is the same as the C function **OSRIsGeocentric()** (p. 505).

Returns

TRUE if this contains a GEOCCS node indicating a it is a geocentric coordinate system.

Since

OGR 1.9.0

References OGR_SRSNode::GetValue().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), exportToProj4(), importFromProj4(), and SetGeogCS().

11.69.3.57 int OGRSpatialReference::IsGeographic ( ) const

Check if geographic coordinate system.

This method is the same as the C function **OSRIsGeographic()** (p. 505).

Returns

TRUE if this spatial reference is geographic ... that is the root is a GEOGCS node.

References GetAttrNode(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToProj4(), exportToXML(), importFromEPSGA(), importFromPCI(), SetCompoundCS(), SetVertCS(), and SetWellKnownGeogCS().

11.69.3.58 int OGRSpatialReference::IsLocal ( ) const

Check if local coordinate system.

This method is the same as the C function **OSRIsLocal()** (p. 505).

Returns

TRUE if this spatial reference is local ... that is the root is a LOCAL_CS node.

Referenced by exportToPanorama(), exportToPCI(), exportToUSGS(), importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and IsSame().

11.69.3.59 int OGRSpatialReference::IsProjected ( ) const

Check if projected coordinate system.

This method is the same as the C function **OSRIsProjected()** (p. 505).

Returns

TRUE if this contains a PROJCS node indicating a it is a projected coordinate system.

References GetAttrNode(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), exportToERM(), exportToXML(), importFromEPSGA(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), IsSame(), SetCompoundCS(), and SetVertCS().

11.69.3.60 int OGRSpatialReference::IsSame (const OGRSpatialReference * *poOtherSRS*) const

Do these two spatial references describe the same system ?

Parameters

poOtherSRS	the SRS being compared to.

Returns

TRUE if equivalent or FALSE otherwise.

References GetAttrNode(), GetAttrValue(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR_SRSNode::GetValue(), IsLocal(), IsProjected(), IsSameGeogCS(), IsSameVertCS(), and IsVertical().

11.69.3.61 int OGRSpatialReference::IsSameGeogCS (const OGRSpatialReference * *poOther*) const

Do the GeogCS'es match?

This method is the same as the C function **OSRIsSameGeogCS()** (p. 506).

Parameters

poOther	the SRS being compared against.

Returns

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), GetAttrValue(), and GetTOWGS84().

Referenced by IsSame(), and morphFromESRI().

11.69.3.62 int OGRSpatialReference::IsSameVertCS (const OGRSpatialReference * *poOther*) const

Do the VertCS'es match?

This method is the same as the C function **OSRIsSameVertCS()** (p. 506).

Parameters

poOther	the SRS being compared against.

Returns

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), and GetAttrValue().

Referenced by IsSame().

11.69.3.63 int OGRSpatialReference::IsVertical ( ) const

Check if vertical coordinate system.

This method is the same as the C function **OSRIsVertical()** (p. 506).

Returns

TRUE if this contains a VERT_CS node indicating a it is a vertical coordinate system.

Since

OGR 1.8.0

References GetAttrNode(), and OGR_SRSNode::GetValue().

Referenced by GetTargetLinearUnits(), IsSame(), SetCompoundCS(), and SetTargetLinearUnits().

11.69.3.64 OGRErr OGRSpatialReference::morphFromESRI ( )

Convert in place from ESRI WKT format.

The value notes of this coordinate system are modified in various manners to adhere more closely to the WKT standard. This mostly involves translating a variety of ESRI names for projections, arguments and datums to "standard" names, as defined by Adam Gawne-Cain's reference translation of EPSG to WKT for the CT specification.

Starting with GDAL 1.9.0, missing parameters in TOWGS84, DATUM or GEOGCS nodes can be added to the WKT, comparing existing WKT parameters to GDAL's databases. Note that this optional procedure is very conservative and should not introduce false information into the WKT defintion (altough caution should be advised when activating it). Needs the Configuration Option GDAL_FIX_ESRI_WKT be set to one of the following values (TOWGS84 is recommended for proper datum shift calculations):

GDAL_FIX_ESRI_WKT values

	TOWGS84		Adds missing TOWGS84 parameters (necessary for datum transformations), based on named datum and spheroid values.
	DATUM		Adds ESPG AUTHORITY nodes and sets SPHEROID name to OGR spec.
	GEOGCS		Adds ESPG AUTHORITY nodes and sets GEOGCS, DATUM and SPHEROID names to OGR spec. Effectively replaces GEOGCS node with the result of importFromEPSG(n), using EPSG code n corresponding to the existing GEOGCS. Does not impact PROJCS values.

This does the same as the C function **OSRMorphFromESRI()** (p. 506).

Returns

OGRERR_NONE unless something goes badly wrong.

References OGR_SRSNode::AddChild(), OGR_SRSNode::applyRemapper(), OGR_SRSNode::Clone(), CloneGeogCS(), CopyGeogCSFrom(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), GetAttrValue(), OGR_SRSNode::GetChild(), GetProjParm(), OGR_SRSNode::GetValue(), importFromEPSG(), OGR_SRSNode::InsertChild(), IsSameGeogCS(), SetNode(), SetProjParm(), OGR_SRSNode::SetValue(), and StripCTParms().

Referenced by importFromESRI(), and SetFromUserInput().

11.69.3.65 OGRErr OGRSpatialReference::morphToESRI ( )

Convert in place to ESRI WKT format.

The value nodes of this coordinate system are modified in various manners more closely map onto the ESRI concept of WKT format. This includes renaming a variety of projections and arguments, and stripping out nodes note recognised by ESRI (like AUTHORITY and AXIS).

This does the same as the C function **OSRMorphToESRI()** (p. 507).

Returns

OGRERR_NONE unless something goes badly wrong.

References OGR_SRSNode::AddChild(), OGR_SRSNode::applyRemapper(), OGR_SRSNode::DestroyChild(), FindProjParm(), Fixup(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetNormProjParm(), GetProjParm(), GetUTMZone(), OGR_SRSNode::GetValue(), SetNode(), OGR_SRSNode::SetValue(), and StripCTParms().

11.69.3.66 int OGRSpatialReference::Reference ( )

Increments the reference count by one.

The reference count is used keep track of the number of **OGRGeometry** (p. 126) objects referencing this SRS.

The method does the same thing as the C function **OSRReference()** (p. 507).

Returns

the updated reference count.

Referenced by OGRGeometry::assignSpatialReference().

11.69.3.67 void OGRSpatialReference::Release ( )

Decrements the reference count by one, and destroy if zero.

The method does the same thing as the C function **OSRRelease()** (p. 507).

References Dereference().

Referenced by OGRGeometry::assignSpatialReference().

11.69.3.68 OGRErr OGRSpatialReference::SetACEA (double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Albers Conic Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.69 OGRErr OGRSpatialReference::SetAE (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Azimuthal Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.70 OGRErr OGRSpatialReference::SetAngularUnits (const char * *pszUnitsName,* double *dfInRadians*)

Set the angular units for the geographic coordinate system.

This method creates a UNIT subnode with the specified values as a child of the GEOGCS node.

This method does the same as the C function **OSRSetAngularUnits()** (p. 507).

Parameters

pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 490) such as SRS_UA_DEGREE.
dfInRadians	the value to multiple by an angle in the indicated units to transform to radians. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 490).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::SetValue().

Referenced by CloneGeogCS(), Fixup(), and importFromPCI().

11.69.3.71 OGRErr OGRSpatialReference::SetAuthority (const char * *pszTargetKey,* const char * *pszAuthority,* int *nCode*)

Set the authority for a node.

This method is the same as the C function **OSRSetAuthority()** (p. 508).

Parameters

pszTargetKey	the partial or complete path to the node to set an authority on. ie. "PROJCS", "GEOGCS" or "GEOGCS|UNIT".
pszAuthority	authority name, such as "EPSG".
nCode	code for value with this authority.

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), and GetAttrNode().

Referenced by AutoIdentifyEPSG(), importFromEPSGA(), importFromPanorama(), importFromPCI(), importFromUSGS(), and importFromWMSAUTO().

11.69.3.72 OGRErr OGRSpatialReference::SetAxes (const char * *pszTargetKey,* const char * *pszXAxisName,* OGRAxisOrientation *eXAxisOrientation,* const char * *pszYAxisName,* OGRAxisOrientation *eYAxisOrientation*)

Set the axes for a coordinate system.

Set the names, and orientations of the axes for either a projected (PROJCS) or geographic (GEOGCS) coordinate system.

This method is equivalent to the C function OSRSetAxes().

Parameters

pszTargetKey	either "PROJCS" or "GEOGCS", must already exist in SRS.
pszXAxisName	name of first axis, normally "Long" or "Easting".
eXAxis-Orientation	normally OAO_East.
pszYAxisName	name of second axis, normally "Lat" or "Northing".
eYAxis-Orientation	normally OAO_North.

Returns

OGRERR_NONE on success or an error code.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), and OSRAxisEnumToName().

11.69.3.73 OGRErr OGRSpatialReference::SetBonne (double *dfStdP1,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Bonne

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.74 OGRErr OGRSpatialReference::SetCEA (double *dfStdP1,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Cylindrical Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

11.69.3.75 OGRErr OGRSpatialReference::SetCompoundCS (const char * *pszName,* const OGRSpatialReference * *poHorizSRS,* const OGRSpatialReference * *poVertSRS*)

Setup a compound coordinate system.

This method is the same as the C function **OSRSetCompoundCS()** (p. 508).

This method is replace the current SRS with a COMPD_CS coordinate system consisting of the passed in horizontal and vertical coordinate systems.

Parameters

pszName	the name of the compound coordinate system.
poHorizSRS	the horizontal SRS (PROJCS or GEOGCS).
poVertSRS	the vertical SRS (VERT_CS).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), IsGeographic(), IsProjected(), and IsVertical().

11.69.3.76 OGRErr OGRSpatialReference::SetCS (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Cassini-Soldner

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromProj4().

11.69.3.77 OGRErr OGRSpatialReference::SetEC (double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equidistant Conic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.78 OGRErr OGRSpatialReference::SetEckert (int *nVariation,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert I-VI

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.79 OGRErr OGRSpatialReference::SetEquirectangular (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromWMSAUTO().

11.69.3.80 OGRErr OGRSpatialReference::SetEquirectangular2 (double *dfCenterLat,* double *dfCenterLong,* double *dfPseudoStdParallel1,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular generalized form :

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.81 OGRErr OGRSpatialReference::SetExtension (const char * *pszTargetKey,* const char * *pszName,* const char * *pszValue*)

Set extension value.

Set the value of the named EXTENSION item for the identified target node.

Parameters

pszTargetKey	the name or path to the parent node of the EXTENSION.
pszName	the name of the extension being fetched.
pszValue	the value to set

Returns

OGRERR_NONE on success

References OGR_SRSNode::AddChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and OGR_SRSNode::SetValue().

Referenced by importFromProj4().

11.69.3.82 OGRErr OGRSpatialReference::SetFromUserInput (const char * *pszDefinition*)

Set spatial reference from various text formats.

This method will examine the provided input, and try to deduce the format, and then use it to initialize the spatial reference system. It may take the following forms:

1. Well Known Text definition - passed on to **importFromWkt()** (p. 260).
2. "EPSG:n" - number passed on to **importFromEPSG()** (p. 250).
3. "EPSGA:n" - number passed on to **importFromEPSGA()** (p. 250).
4. "AUTO:proj_id,unit_id,lon0,lat0" - WMS auto projections.
5. "urn:ogc:def:crs:EPSG::n" - ogc urns
6. PROJ.4 definitions - passed on to **importFromProj4()** (p. 254).
7. filename - file read for WKT, XML or PROJ.4 definition.

8. well known name accepted by **SetWellKnownGeogCS()** (p. 282), such as NAD27, NAD83, WGS84 or WGS72.
9. WKT (directly or in a file) in ESRI format should be prefixed with ESRI:: to trigger an automatic **morphFromESRI()** (p. 263).
10. "IGNF:xxx" - "+init=IGNF:xxx" passed on to **importFromProj4()** (p. 254).

It is expected that this method will be extended in the future to support XML and perhaps a simplified "minilanguage" for indicating common UTM and State Plane definitions.

This method is intended to be flexible, but by it's nature it is imprecise as it must guess information about the format intended. When possible applications should call the specific method appropriate if the input is known to be in a particular format.

This method does the same thing as the **OSRSetFromUserInput()** (p. 510) function.

Parameters

pszDefinition	text definition to try to deduce SRS from.

Returns

OGRERR_NONE on success, or an error code if the name isn't recognised, the definition is corrupt, or an EPSG value can't be successfully looked up.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), importFromDict(), importFromEPSG(), importFromEPSGA(), importFromProj4(), importFromUrl(), importFromURN(), importFromWkt(), importFromWMSAUTO(), importFromXML(), morphFromESRI(), SetNode(), and SetWellKnownGeogCS().

Referenced by importFromUrl().

11.69.3.83 OGRErr OGRSpatialReference::SetGaussSchreiberTMercator (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.84 OGRErr OGRSpatialReference::SetGeocCS (const char * *pszName*)

Set the user visible GEOCCS name.

This method is the same as the C function **OSRSetGeocCS()** (p. 510).

This method will ensure a GEOCCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the DATUM and PRIMEM nodes from the GEOGCS will be tarnsferred over to the GEOGCS.

Parameters

pszName	the user visible name to assign. Not used as a key.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_SRSNode::Clone(), GetAttrNode(), OGR_SRSNode::GetNode(), OGR_SRSNode::GetValue(), OGR_SRSNode::InsertChild(), and SetNode().

Referenced by importFromProj4().

11.69.3.85 OGRErr OGRSpatialReference::SetGeogCS (const char * *pszGeogName,* const char * *pszDatumName,* const char * *pszSpheroidName,* double *dfSemiMajor,* double *dfInvFlattening,* const char * *pszPMName* = NULL, double *dfPMOffset* = 0.0, const char * *pszAngularUnits* = NULL, double *dfConvertToRadians* = 0.0)

Set geographic coordinate system.

This method is used to set the datum, ellipsoid, prime meridian and angular units for a geographic coordinate system. It can be used on it's own to establish a geographic spatial reference, or applied to a projected coordinate system to establish the underlying geographic coordinate system.

This method does the same as the C function **OSRSetGeogCS()** (p. 510).

Parameters

pszGeogName	user visible name for the geographic coordinate system (not to serve as a key).
pszDatumName	key name for this datum. The OpenGIS specification lists some known values, and otherwise EPSG datum names with a standard transformation are considered legal keys.
pszSpheroid-Name	user visible spheroid name (not to serve as a key)
dfSemiMajor	the semi major axis of the spheroid.
dfInvFlattening	the inverse flattening for the spheroid. This can be computed from the semi minor axis as 1/f = 1.0 / (1.0 - semiminor/semimajor).
pszPMName	the name of the prime merdidian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.
dfPMOffset	the longitude of greenwich relative to this prime meridian.
pszAngularUnits	the angular units name (see **ogr_srs_api.h** (p. 490) for some standard names). If NULL a value of "degrees" will be assumed.
dfConvertTo-Radians	value to multiply angular units by to transform them to radians. A value of SRS_UL_DEGREE_CONV will be used if pszAngularUnits is NULL.

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), Clear(), CopyGeogCSFrom(), CPLAtof(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::InsertChild(), IsGeocentric(), SetGeogCS(), and SetRoot().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and SetGeogCS().

11.69.3.86 OGRErr OGRSpatialReference::SetGEOS (double *dfCentralMeridian,* double *dfSatelliteHeight,* double *dfFalseEasting,* double *dfFalseNorthing*)

Geostationary Satellite

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.87 OGRErr OGRSpatialReference::SetGH (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Goode Homolosine

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.88 OGRErr OGRSpatialReference::SetGnomonic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gnomonic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.89 OGRErr OGRSpatialReference::SetGS (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gall Stereograpic

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.90 OGRErr OGRSpatialReference::SetHOM (double *dfCenterLat,* double *dfCenterLong,* double *dfAzimuth,* double *dfRectToSkew,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Set a Hotine Oblique Mercator projection using azimuth angle.

Hotine Oblique Mercator

This method does the same thing as the C function **OSRSetHOM()** (p. 511).

Parameters

dfCenterLat	Latitude of the projection origin.
dfCenterLong	Longitude of the projection origin.
dfAzimuth	Azimuth, measured clockwise from North, of the projection centerline.
dfRectToSkew	?.
dfScale	Scale factor applies to the projection origin.
dfFalseEasting	False easting.
dfFalseNorthing	False northing.

Returns

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.91 OGRErr OGRSpatialReference::SetHOM2PNO (double *dfCenterLat,* double *dfLat1,* double *dfLong1,* double *dfLat2,* double *dfLong2,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Set a Hotine Oblique Mercator projection using two points on projection centerline.

This method does the same thing as the C function **OSRSetHOM2PNO()** (p. 511).

Parameters

dfCenterLat	Latitude of the projection origin.
dfLat1	Latitude of the first point on center line.
dfLong1	Longitude of the first point on center line.
dfLat2	Latitude of the second point on center line.
dfLong2	Longitude of the second point on center line.
dfScale	Scale factor applies to the projection origin.
dfFalseEasting	False easting.
dfFalseNorthing	False northing.

Returns

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromUSGS().

11.69.3.92 OGRErr OGRSpatialReference::SetIGH ( )

Interrupted Goode Homolosine

References SetProjection().

Referenced by importFromProj4().

11.69.3.93 OGRErr OGRSpatialReference::SetIWMPolyconic (double *dfLat1,* double *dfLat2,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

International Map of the World Polyconic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

11.69.3.94 OGRErr OGRSpatialReference::SetKrovak (double *dfCenterLat,* double *dfCenterLong,* double *dfAzimuth,* double *dfPseudoStdParallelLat,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.95 OGRErr OGRSpatialReference::SetLAEA (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.96 OGRErr OGRSpatialReference::SetLCC (double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.97 OGRErr OGRSpatialReference::SetLCC1SP (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromProj4().

11.69.3.98 OGRErr OGRSpatialReference::SetLCCB (double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References SetNormProjParm(), and SetProjection().

11.69.3.99 OGRErr OGRSpatialReference::SetLinearUnits (const char * *pszUnitsName,* double *dfInMeters*)

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS, GEOCCS or LOCAL_CS node.

This method does the same as the C function **OSRSetLinearUnits()** (p. 512).

Parameters

pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 490) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 490).

Returns

OGRERR_NONE on success.

References SetTargetLinearUnits().

Referenced by Fixup(), importFromERM(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), importFromWMSAUTO(), SetLinearUnitsAndUpdateParameters(), SetStatePlane(), and SetUTM().

11.69.3.100 OGRErr OGRSpatialReference::SetLinearUnitsAndUpdateParameters (const char * *pszName,* double *dfInMeters*)

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS or LOCAL_CS node. It works the same as the **SetLinearUnits()** (p. 273) method, but it also updates all existing linear projection parameter values from the old units to the new units.

Parameters

pszName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 490) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 490).

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR_SRSNode::GetValue(), SetLinearUnits(), and SetProjParm().

Referenced by importFromESRI(), and importFromPCI().

11.69.3.101 OGRErr OGRSpatialReference::SetLocalCS (const char * *pszName*)

Set the user visible LOCAL_CS name.

This method is the same as the C function **OSRSetLocalCS()** (p. 513).

This method will ensure a LOCAL_CS node is created as the root, and set the provided name on it. It must be used before **SetLinearUnits()** (p. 273).

Parameters

pszName	the user visible name to assign. Not used as a key.

Returns

OGRERR_NONE on success.

References GetAttrNode(), and SetNode().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromUSGS(), and SetStatePlane().

11.69.3.102 OGRErr OGRSpatialReference::SetMC (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Miller Cylindrical

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.103 OGRErr OGRSpatialReference::SetMercator (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.104 OGRErr OGRSpatialReference::SetMollweide (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mollweide

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

11.69.3.105 OGRErr OGRSpatialReference::SetNode (const char * *pszNodePath,* const char * *pszNewNodeValue*)

Set attribute value in spatial reference.

Missing intermediate nodes in the path will be created if not already in existance. If the attribute has no children one will be created and assigned the value otherwise the zeroth child will be assigned the value.

This method does the same as the C function **OSRSetAttrValue()** (p. 508).

Parameters

pszNodePath	full path to attribute to be set. For instance "PROJCS|GEOGCS|UNIT".
pszNewNode-Value	value to be assigned to node, such as "meter". This may be NULL if you just want to force creation of the intermediate path.

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), SetRoot(), and OGR_SRSNode::SetValue().

Referenced by importFromProj4(), importFromURN(), morphFromESRI(), morphToESRI(), SetFromUserInput(), SetGeocCS(), SetLocalCS(), SetProjCS(), SetProjection(), and SetUTM().

11.69.3.106 OGRErr OGRSpatialReference::SetNormProjParm (const char * *pszName,* double *dfValue*)

Set a projection parameter with a normalized value.

This method is the same as **SetProjParm()** (p. 277) except that the value of the parameter passed in is assumed to be in "normalized" form (decimal degrees for angular values, meters for linear values. The values are converted in a form suitable for the GEOGCS and linear units in effect.

This method is the same as the C function **OSRSetNormProjParm()** (p. 513).

Parameters

pszName	the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. 490), such as SRS_PP_CENTRAL_MERIDIAN.
dfValue	value to assign.

Returns

OGRERR_NONE on success.

References SetProjParm().

Referenced by importFromProj4(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStatePlane(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

11.69.3.107 OGRErr OGRSpatialReference::SetNZMG (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

New Zealand Map Grid

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.108 OGRErr OGRSpatialReference::SetOrthographic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Orthographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

11.69.3.109 OGRErr OGRSpatialReference::SetOS (double *dfOriginLat,* double *dfCMeridian,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Oblique Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromProj4().

11.69.3.110 OGRErr OGRSpatialReference::SetPolyconic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polyconic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.111 OGRErr OGRSpatialReference::SetProjCS (const char * *pszName*)

Set the user visible PROJCS name.

This method is the same as the C function **OSRSetProjCS()** (p. 514).

This method will ensure a PROJCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the GEOGCS node will be demoted to be a child of the new PROJCS root.

Parameters

pszName	the user visible name to assign. Not used as a key.

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetValue(), OGR_SRSNode::InsertChild(), and SetNode().

11.69.3.112 OGRErr OGRSpatialReference::SetProjection (const char * *pszProjection*)

Set a projection name.

This method is the same as the C function **OSRSetProjection()** (p. 514).

Parameters

pszProjection	the projection name, which should be selected from the macros in **ogr_srs_api.h** (p. 490), such as SRS_PT_TRANSVERSE_MERCATOR.

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetValue(), OGR_SRSNode::InsertChild(), and SetNode().

Referenced by SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetIGH(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

11.69.3.113 OGRErr OGRSpatialReference::SetProjParm (const char * *pszParmName,* double *dfValue*)

Set a projection parameter value.

Adds a new PARAMETER under the PROJCS with the indicated name and value.

This method is the same as the C function **OSRSetProjParm()** (p. 514).

Please check `http://www.remotesensing.org/geotiff/proj_list` pages for legal parameter names for specific projections.

Parameters

pszParmName	the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. 490), such as SRS_PP_CENTRAL_MERIDIAN.
dfValue	value to assign.

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and OGR_SRSNode::SetValue().

Referenced by morphFromESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

11.69.3.114 OGRErr OGRSpatialReference::SetPS (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polar Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.115 OGRErr OGRSpatialReference::SetRobinson (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Robinson

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.116 void OGRSpatialReference::SetRoot (OGR_SRSNode * *poNewRoot*)

Set the root SRS node.

If the object has an existing tree of OGR_SRSNodes, they are destroyed as part of assigning the new root. Ownership of the passed **OGR_SRSNode** (p. 75) is is assumed by the **OGRSpatialReference** (p. 230).

Parameters

poNewRoot	object to assign as root.

Referenced by CloneGeogCS(), CopyGeogCSFrom(), SetGeogCS(), SetNode(), SetVertCS(), and StripVertical().

11.69.3.117 OGRErr OGRSpatialReference::SetSinusoidal (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Sinusoidal

References SetNormProjParm(), and SetProjection().

Referenced by importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.118 OGRErr OGRSpatialReference::SetSOC (double *dfLatitudeOfOrigin,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References SetNormProjParm(), and SetProjection().

11.69.3.119 OGRErr OGRSpatialReference::SetStatePlane (int *nZone,* int *bNAD83* = `TRUE`, const char * *pszOverrideUnitName* = `NULL`, double *dfOverrideUnit* = `0.0`)

Set State Plane projection definition.

State Plane

This will attempt to generate a complete definition of a state plane zone based on generating the entire SRS from the EPSG tables. If the EPSG tables are unavailable, it will produce a stubbed LOCAL_CS definition and return OGRERR_FAILURE.

This method is the same as the C function **OSRSetStatePlaneWithUnits()** (p. 515).

Parameters

nZone	State plane zone number, in the USGS numbering scheme (as dinstinct from the Arc/Info and Erdas numbering scheme.
bNAD83	TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used.
pszOverrideUnit-Name	Linear unit name to apply overriding the legal definition for this zone.
dfOverrideUnit	Linear unit conversion factor to apply overriding the legal definition for this zone.

Returns

OGRERR_NONE on success, or OGRERR_FAILURE on failure, mostly likely due to the EPSG tables not being accessable.

References Clear(), CPLAtof(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), GetLinearUnits(), GetNormProjParm(), importFromEPSG(), SetLinearUnits(), SetLocalCS(), and SetNormProjParm().

Referenced by importFromESRI(), importFromPCI(), and importFromUSGS().

11.69.3.120 OGRErr OGRSpatialReference::SetStereographic (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.121 OGRErr OGRSpatialReference::SetTargetLinearUnits (const char * *pszTargetKey,* const char * *pszUnitsName,* double *dfInMeters*)

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the target node.

This method does the same as the C function **OSRSetTargetLinearUnits()** (p. 515).

Parameters

pszTargetKey	the keyword to set the linear units for. ie. "PROJCS" or "VERT_CS"
pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 490) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 490).

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), IsVertical(), and OGR_SRSNode::SetValue().

Referenced by SetLinearUnits().

11.69.3.122 OGRErr OGRSpatialReference::SetTM (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

11.69.3.123 OGRErr OGRSpatialReference::SetTMG (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Tunesia Mining Grid

References SetNormProjParm(), and SetProjection().

11.69.3.124 OGRErr OGRSpatialReference::SetTMSO (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.125 OGRErr OGRSpatialReference::SetTMVariant (const char * *pszVariantName,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Transverse Mercator variants.

References SetNormProjParm(), and SetProjection().

11.69.3.126 OGRErr OGRSpatialReference::SetTOWGS84 (double *dfDX,* double *dfDY,* double *dfDZ,* double *dfEX* = 0.0, double *dfEY* = 0.0, double *dfEZ* = 0.0, double *dfPPM* = 0.0)

Set the Bursa-Wolf conversion to WGS84.

This will create the TOWGS84 node as a child of the DATUM. It will fail if there is no existing DATUM node. Unlike most **OGRSpatialReference** (p. 230) methods it will insert itself in the appropriate order, and will replace an existing TOWGS84 node if there is one.

The parameters have the same meaning as EPSG transformation 9606 (Position Vector 7-param. transformation).

This method is the same as the C function **OSRSetTOWGS84()** (p. 516).

Parameters

dfDX	X child in meters.
dfDY	Y child in meters.
dfDZ	Z child in meters.
dfEX	X rotation in arc seconds (optional, defaults to zero).
dfEY	Y rotation in arc seconds (optional, defaults to zero).
dfEZ	Z rotation in arc seconds (optional, defaults to zero).
dfPPM	scaling factor (parts per million).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::InsertChild().

Referenced by importFromPCI(), and importFromProj4().

11.69.3.127 OGRErr OGRSpatialReference::SetTPED (double *dfLat1,* double *dfLong1,* double *dfLat2,* double *dfLong2,* double *dfFalseEasting,* double *dfFalseNorthing*)

Two Point Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

11.69.3.128 OGRErr OGRSpatialReference::SetUTM (int *nZone,* int *bNorth* = TRUE)

Set UTM projection definition.

Universal Transverse Mercator

This will generate a projection definition with the full set of transverse mercator projection parameters for the given UTM zone. If no PROJCS[] description is set yet, one will be set to look like "UTM Zone %d, {Northern, Southern} Hemisphere".

This method is the same as the C function **OSRSetUTM()** (p. 516).

Parameters

nZone	UTM zone.
bNorth	TRUE for northern hemisphere, or FALSE for southern hemisphere.

Returns

OGRERR_NONE on success.

References GetAttrValue(), SetLinearUnits(), SetNode(), SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

11.69.3.129 OGRErr OGRSpatialReference::SetVDG (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

VanDerGrinten

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

11.69.3.130 OGRErr OGRSpatialReference::SetVertCS (const char * *pszVertCSName,* const char * *pszVertDatumName,* int *nVertDatumType* = 2005)

Set the user visible VERT_CS name.

This method is the same as the C function **OSRSetVertCS()** (p. 517).

This method will ensure a VERT_CS node is created if needed. If the existing coordinate system is GEOGCS or PROJCS rooted, then it will be turned into a COMPD_CS.

Parameters

pszVertCSName	the user visible name of the vertical coordinate system. Not used as a key.
pszVertDatum-Name	the user visible name of the vertical datum. It is helpful if this matches the EPSG name.
nVertDatumType	the OGC vertical datum type, usually 2005.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_SRSNode::AddChild(), Clear(), GetAttrNode(), OGR_SRSNode::GetValue(), IsGeographic(), IsProjected(), and SetRoot().

11.69.3.131 OGRErr OGRSpatialReference::SetWagner (int *nVariation,* double *dfCenterLat,* double *dfFalseEasting,* double *dfFalseNorthing*)

Wagner I – VII

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromUSGS().

11.69.3.132 OGRErr OGRSpatialReference::SetWellKnownGeogCS (const char * *pszName*)

Set a GeogCS based on well known name.

This may be called on an empty **OGRSpatialReference** (p. 230) to make a geographic coordinate system, or on something with an existing PROJCS node to set the underlying geographic coordinate system of a projected coordinate system.

The following well known text values are currently supported:

- "WGS84": same as "EPSG:4326" but has no dependence on EPSG data files.
- "WGS72": same as "EPSG:4322" but has no dependence on EPSG data files.
- "NAD27": same as "EPSG:4267" but has no dependence on EPSG data files.
- "NAD83": same as "EPSG:4269" but has no dependence on EPSG data files.
- "EPSG:n": same as doing an ImportFromEPSG(n).

Parameters

pszName	name of well known geographic coordinate system.

Returns

OGRERR_NONE on success, or OGRERR_FAILURE if the name isn't recognised, the target object is already initialized, or an EPSG value can't be successfully looked up.

References CopyGeogCSFrom(), importFromEPSG(), importFromEPSGA(), importFromWkt(), and IsGeographic().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromProj4(), importFromUSGS(), importFromWMSAUTO(), and SetFromUserInput().

11.69.3.133 OGRErr OGRSpatialReference::StripCTParms (OGR_SRSNode * *poCurrent* = NULL)

Strip OGC CT Parameters.

This method will remove all components of the coordinate system that are specific to the OGC CT Specification. That is it will attempt to strip it down to being compatible with the Simple Features 1.0 specification.

This method is the same as the C function **OSRStripCTParms()** (p. 517).

Parameters

poCurrent	node to operate on. NULL to operate on whole tree.

Returns

OGRERR_NONE on success or an error code.

References OGR_SRSNode::GetValue(), OGR_SRSNode::StripNodes(), and StripVertical().

Referenced by morphFromESRI(), and morphToESRI().

11.69.3.134 OGRErr OGRSpatialReference::StripVertical ( )

Convert a compound cs into a horizontal CS.

If this SRS is of type COMPD_CS[] then the vertical CS and the root COMPD_CS nodes are stripped resulting and only the horizontal coordinate system portion remains (normally PROJCS, GEOGCS or LOCAL_CS).

If this is not a compound coordinate system then nothing is changed.

Since

OGR 1.8.0

References OGR_SRSNode::Clone(), OGR_SRSNode::GetChild(), and SetRoot().

Referenced by StripCTParms().

11.69.3.135 OGRErr OGRSpatialReference::Validate ( )

Validate SRS tokens.

This method attempts to verify that the spatial reference system is well formed, and consists of known tokens. The validation is not comprehensive.

This method is the same as the C function **OSRValidate()** (p. 517).

Returns

OGRERR_NONE if all is fine, OGRERR_CORRUPT_DATA if the SRS is not well formed, and OGRERR_UNSUPPORTED_SRS if the SRS is well formed, but contains non-standard PROJECTION[] values.

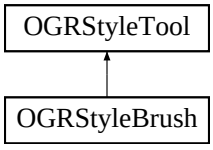
The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogr_fromepsg.cpp
- ogr_srs_dict.cpp
- ogr_srs_erm.cpp
- ogr_srs_esri.cpp
- ogr_srs_ozi.cpp
- ogr_srs_panorama.cpp
- ogr_srs_pci.cpp
- ogr_srs_proj4.cpp
- ogr_srs_usgs.cpp
- ogr_srs_validate.cpp
- ogr_srs_xml.cpp
- ogrspatialreference.cpp

11.70 OGRStyleBrush Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleBrush:



11.70.1 Detailed Description

This class represents a style brush

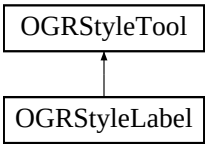
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.71 OGRStyleLabel Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleLabel:



11.71.1 Detailed Description

This class represents a style label

The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.72 OGRStyleMgr Class Reference

`#include <ogr_featurestyle.h>`

Public Member Functions

- **OGRStyleMgr** (**OGRStyleTable** *poDataSetStyleTable=NULL)

  Constructor.
- ~**OGRStyleMgr** ()

  Destructor.
- GBool **SetFeatureStyleString** (**OGRFeature** *, const char *pszStyleString=NULL, GBool bNoMatching=FALSE)

  Set a style in a feature.
- const char * **InitFromFeature** (**OGRFeature** *)

  Initialize style manager from the style string of a feature.
- GBool **InitStyleString** (const char *pszStyleString=NULL)

  Initialize style manager from the style string.
- const char * **GetStyleName** (const char *pszStyleString=NULL)

  Get the name of a style from the style table.
- const char * **GetStyleByName** (const char *pszStyleName)

  find a style in the current style table.

- GBool **AddStyle** (const char *pszStyleName, const char *pszStyleString=NULL)

  Add a style to the current style table.
- const char * **GetStyleString** (**OGRFeature** *=NULL)

  Get the style string from the style manager.
- GBool **AddPart** (**OGRStyleTool** *)

  Add a part (style tool) to the current style.
- GBool **AddPart** (const char *)

  Add a part (style string) to the current style.
- int **GetPartCount** (const char *pszStyleString=NULL)

  Get the number of parts in a style.
- **OGRStyleTool** * **GetPart** (int hPartId, const char *pszStyleString=NULL)

  Fetch a part (style tool) from the current style.

11.72.1 Detailed Description

This class represents a style manager

11.72.2 Constructor & Destructor Documentation

11.72.2.1 OGRStyleMgr::OGRStyleMgr (OGRStyleTable * *poDataSetStyleTable* = NULL)

Constructor.

This method is the same as the C function **OGR_SM_Create()** (p. 471)

Parameters

poDataSetStyle-Table	(currently unused, reserved for future use), pointer to **OGRStyleTable** (p. 289). Pass NULL for now.

11.72.2.2 OGRStyleMgr::~OGRStyleMgr ( )

Destructor.

This method is the same as the C function **OGR_SM_Destroy()** (p. 471)

11.72.3 Member Function Documentation

11.72.3.1 GBool OGRStyleMgr::AddPart (OGRStyleTool * *poStyleTool*)

Add a part (style tool) to the current style.

This method is the same as the C function **OGR_SM_AddPart()** (p. 470).

Parameters

poStyleTool	the style tool defining the part to add.

Returns

TRUE on success, FALSE on errors.

11.72.3.2 GBool OGRStyleMgr::AddPart (const char * *pszPart*)

Add a part (style string) to the current style.

Parameters

pszPart	the style string defining the part to add.

Returns

TRUE on success, FALSE on errors.

11.72.3.3 GBool OGRStyleMgr::AddStyle (const char * *pszStyleName,* const char * *pszStyleString* = NULL)

Add a style to the current style table.

This method is the same as the C function **OGR_SM_AddStyle()** (p. 470).

Parameters

pszStyleName	the name of the style to add.
pszStyleString	the style string to use, or NULL to use the style stored in the manager.

Returns

TRUE on success, FALSE on errors.

References OGRStyleTable::AddStyle().

11.72.3.4 OGRStyleTool * OGRStyleMgr::GetPart (int *nPartId,* const char * *pszStyleString* = NULL)

Fetch a part (style tool) from the current style.

This method is the same as the C function **OGR_SM_GetPart()** (p. 471).

This method instanciates a new object that should be freed with **OGR_ST_Destroy()** (p. 473).

Parameters

nPartId	the part number (0-based index).
pszStyleString	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns

OGRStyleTool (p. 292) of the requested part (style tools) or NULL on error.

11.72.3.5 int OGRStyleMgr::GetPartCount (const char * *pszStyleString* = NULL)

Get the number of parts in a style.

This method is the same as the C function **OGR_SM_GetPartCount()** (p. 472).

Parameters

pszStyleString	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns

the number of parts (style tools) in the style.

11.72.3.6 const char * OGRStyleMgr::GetStyleByName (const char * *pszStyleName*)

find a style in the current style table.

Parameters

pszStyleName	the name of the style to add.

Returns

the style string matching the name or NULL if not found or error.

References OGRStyleTable::Find().

Referenced by InitStyleString().

11.72.3.7 const char * OGRStyleMgr::GetStyleName (const char * *pszStyleString* = `NULL`)

Get the name of a style from the style table.

Parameters

pszStyleString	the style to search for, or NULL to use the style currently stored in the manager.

Returns

The name if found, or NULL on error.

References OGRStyleTable::GetStyleName().

Referenced by SetFeatureStyleString().

11.72.3.8 const char * OGRStyleMgr::GetStyleString (OGRFeature * *poFeature* = `NULL`)

Get the style string from the style manager.

Parameters

poFeature	feature object from which to read the style or NULL to get the style string stored in the manager.

Returns

the style string stored in the feature or the style string stored in the style manager if poFeature is NULL

NOTE: this method will call **OGRStyleMgr::InitFromFeature()** (p. 287) if poFeature is not NULL and replace the style string stored in the style manager

References InitFromFeature().

11.72.3.9 const char * OGRStyleMgr::InitFromFeature (OGRFeature * *poFeature*)

Initialize style manager from the style string of a feature.

This method is the same as the C function **OGR_SM_InitFromFeature()** (p. 472).

Parameters

poFeature	feature object from which to read the style.

Returns

a reference to the style string read from the feature, or NULL in case of error..

References OGRFeature::GetStyleString(), and InitStyleString().

Referenced by GetStyleString().

11.72.3.10 GBool OGRStyleMgr::InitStyleString (const char * *pszStyleString* = NULL)

Initialize style manager from the style string.

This method is the same as the C function **OGR_SM_InitStyleString()** (p. 472).

Parameters

pszStyleString	the style string to use (can be NULL).

Returns

TRUE on success, FALSE on errors.

References GetStyleByName().

Referenced by InitFromFeature().

11.72.3.11 GBool OGRStyleMgr::SetFeatureStyleString (OGRFeature * *poFeature*, const char * *pszStyleString* = NULL, GBool *bNoMatching* = FALSE)

Set a style in a feature.

Parameters

poFeature	the feature object to store the style in
pszStyleString	the style to store
bNoMatching	TRUE to lookup the style in the style table and add the name to the feature

Returns

TRUE on success, FALSE on error.

References GetStyleName(), and OGRFeature::SetStyleString().

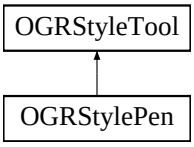
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.73 OGRStylePen Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStylePen:



11.73.1 Detailed Description

This class represents a style pen

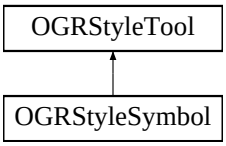
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.74 OGRStyleSymbol Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleSymbol:



11.74.1 Detailed Description

This class represents a style symbol

The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.75 OGRStyleTable Class Reference

`#include <ogr_featurestyle.h>`

Public Member Functions

- GBool **AddStyle** (const char ∗pszName, const char ∗pszStyleString)

  Add a new style in the table. No comparison will be done on the Style string, only on the name.
- GBool **RemoveStyle** (const char ∗pszName)

  Remove a style in the table by its name.
- GBool **ModifyStyle** (const char ∗pszName, const char ∗pszStyleString)

  Modify a style in the table by its name If the style does not exist, it will be added.
- GBool **SaveStyleTable** (const char ∗pszFilename)

  Save a style table to a file.
- GBool **LoadStyleTable** (const char ∗pszFilename)

Load a style table from a file.

- const char * **Find** (const char *pszStyleString)

  Get a style string by name.

- GBool **IsExist** (const char *pszName)

  Get the index of a style in the table by its name.

- const char * **GetStyleName** (const char *pszName)

  Get style name by style string.

- void **Print** (FILE *fpOut)

  Print a style table to a FILE pointer.

- void **Clear** ()

  Clear a style table.

- **OGRStyleTable** * **Clone** ()

  Duplicate style table.

11.75.1 Detailed Description

This class represents a style table

11.75.2 Member Function Documentation

11.75.2.1 GBool OGRStyleTable::AddStyle (const char * *pszName,* const char * *pszStyleString*)

Add a new style in the table. No comparison will be done on the Style string, only on the name.

Parameters

pszName	the name the style to add.
pszStyleString	the style string to add.

Returns

TRUE on success, FALSE on error

References IsExist().

Referenced by OGRStyleMgr::AddStyle(), and ModifyStyle().

11.75.2.2 OGRStyleTable * OGRStyleTable::Clone ( )

Duplicate style table.

The newly created style table is owned by the caller, and will have it's own reference to the **OGRStyleTable** (p. 289).

Returns

new style table, exactly matching this style table.

Referenced by OGRLayer::SetStyleTable(), and OGRDataSource::SetStyleTable().

11.75.2.3 const char * OGRStyleTable::Find (const char * *pszName*)

Get a style string by name.

Parameters

pszName	the name of the style string to find.

Returns

the style string matching the name, NULL if not found or error.

References IsExist().

Referenced by OGRStyleMgr::GetStyleByName().

11.75.2.4 const char * OGRStyleTable::GetStyleName (const char * *pszStyleString*)

Get style name by style string.

Parameters

pszStyleString	the style string to look up.

Returns

the Name of the matching style string or NULL on error.

Referenced by OGRStyleMgr::GetStyleName().

11.75.2.5 int OGRStyleTable::IsExist (const char * *pszName*)

Get the index of a style in the table by its name.

Parameters

pszName	the name to look for.

Returns

The index of the style if found, -1 if not found or error.

Referenced by AddStyle(), Find(), and RemoveStyle().

11.75.2.6 GBool OGRStyleTable::LoadStyleTable (const char * *pszFilename*)

Load a style table from a file.

Parameters

pszFilename	the name of the file to load from.

Returns

TRUE on success, FALSE on error

11.75.2.7 GBool OGRStyleTable::ModifyStyle (const char * *pszName,* const char * *pszStyleString*)

Modify a style in the table by its name If the style does not exist, it will be added.

Parameters

pszName	the name of the style to modify.
pszStyleString	the style string.

Returns

TRUE on success, FALSE on error

References AddStyle(), and RemoveStyle().

11.75.2.8 void OGRStyleTable::Print (FILE * *fpOut*)

Print a style table to a FILE pointer.

Parameters

fpOut	the FILE pointer to print to.

11.75.2.9 GBool OGRStyleTable::RemoveStyle (const char * *pszName*)

Remove a style in the table by its name.

Parameters

pszName	the name of the style to remove.

Returns

TRUE on success, FALSE on error

References IsExist().

Referenced by ModifyStyle().

11.75.2.10 GBool OGRStyleTable::SaveStyleTable (const char * *pszFilename*)

Save a style table to a file.

Parameters

pszFilename	the name of the file to save to.

Returns

TRUE on success, FALSE on error

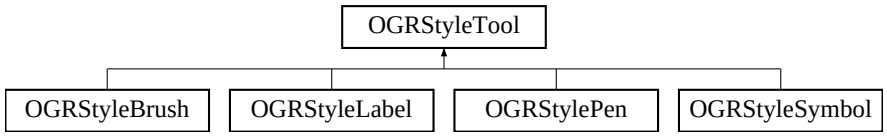
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.76 OGRStyleTool Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleTool:



11.76.1 Detailed Description

This class represents a style tool

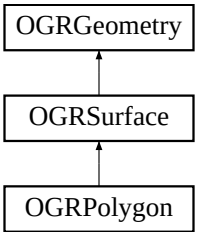
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

11.77 OGRSurface Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRSurface:



Public Member Functions

- virtual double **get_Area** () const =0

  Get the area of the surface object.
- virtual OGRErr **PointOnSurface** (**OGRPoint** ∗poPoint) const =0

  This method relates to the SFCOM ISurface::get_PointOnSurface() method.

11.77.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons.

11.77.2 Member Function Documentation

11.77.2.1 double OGRSurface::get_Area ( ) const [pure virtual]

Get the area of the surface object.

For polygons the area is computed as the area of the outer ring less the area of all internal rings.

This method relates to the SFCOM ISurface::get_Area() method.

Returns

the area of the feature in square units of the spatial reference system in use.

Implemented in **OGRPolygon** (p. 215).

11.77.2.2 OGRErr OGRSurface::PointOnSurface (OGRPoint * *poPoint*) const [pure virtual]

This method relates to the SFCOM ISurface::get_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

Parameters

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implemented in **OGRPolygon** (p. 219).

The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrsurface.cpp

11.78 OZIDatums Struct Reference

The documentation for this struct was generated from the following file:

- ogr_srs_ozi.cpp

11.79 ParseContext Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minixml.cpp

11.80 PCIDatums Struct Reference

The documentation for this struct was generated from the following file:

- ogr_srs_pci.cpp

11.81 projUV Struct Reference

The documentation for this struct was generated from the following file:

- ogrct.cpp

11.82 SFRegion Class Reference

The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

11.83 StackContext Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minixml.cpp

11.84 swq_col_def Struct Reference

The documentation for this struct was generated from the following file:

- swq.h

11.85 swq_expr_node Class Reference

The documentation for this class was generated from the following files:

- swq.h
- swq_expr_node.cpp

11.86 swq_field_list Class Reference

The documentation for this class was generated from the following file:

- swq.h

11.87 swq_join_def Struct Reference

The documentation for this struct was generated from the following file:

- swq.h

11.88 swq_op_registrar Class Reference

The documentation for this class was generated from the following files:

- swq.h
- swq_op_registrar.cpp

11.89 swq_operation Class Reference

The documentation for this class was generated from the following file:

- swq.h

11.90 swq_order_def Struct Reference

The documentation for this struct was generated from the following file:

- swq.h

11.91 swq_parse_context Class Reference

The documentation for this class was generated from the following file:

- swq.h

11.92 swq_select Class Reference

The documentation for this class was generated from the following files:

- swq.h
- swq_select.cpp

11.93 swq_summary Struct Reference

The documentation for this struct was generated from the following file:

- swq.h

11.94 swq_table_def Struct Reference

The documentation for this struct was generated from the following file:

- swq.h

11.95 tm_unz_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.h

11.96 tm_zip_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.h

11.97 unz_file_info_internal_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.cpp

11.98 unz_file_info_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.h

11.99 unz_file_pos_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.h

11.100 unz_global_info_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.h

11.101 unz_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_unzip.cpp

11.102 VSIArchiveContent Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsi_virtual.h

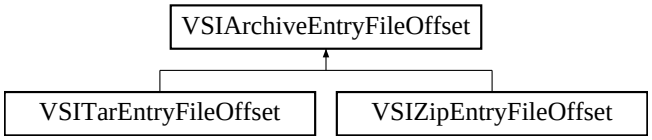
11.103 VSIArchiveEntry Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsi_virtual.h

11.104 VSIArchiveEntryFileOffset Class Reference

Inheritance diagram for VSIArchiveEntryFileOffset:

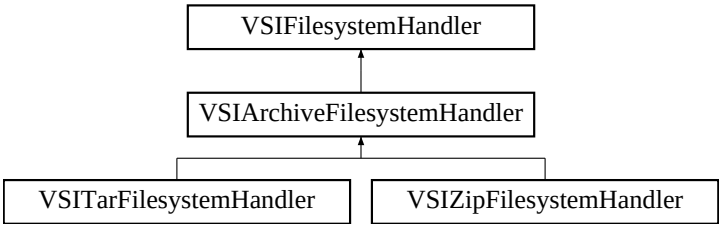


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

11.105 VSIArchiveFilesystemHandler Class Reference

Inheritance diagram for VSIArchiveFilesystemHandler:

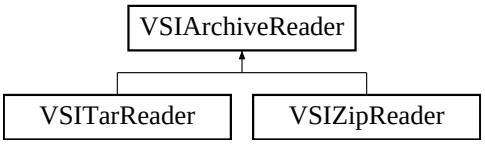


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

11.106 VSIArchiveReader Class Reference

Inheritance diagram for VSIArchiveReader:

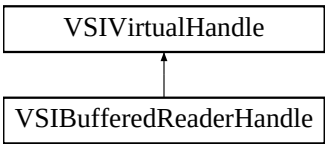


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

11.107 VSIBufferedReaderHandle Class Reference

Inheritance diagram for VSIBufferedReaderHandle:



The documentation for this class was generated from the following file:

- cpl_vsil_buffered_reader.cpp

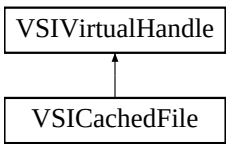
11.108 VSICacheChunk Class Reference

The documentation for this class was generated from the following file:

- cpl_vsil_cache.cpp

11.109 VSICachedFile Class Reference

Inheritance diagram for VSICachedFile:

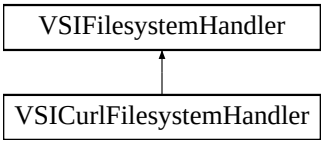


The documentation for this class was generated from the following file:

- cpl_vsil_cache.cpp

11.110 VSICurlFilesystemHandler Class Reference

Inheritance diagram for VSICurlFilesystemHandler:

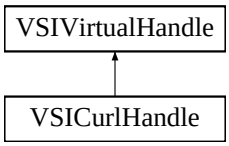


The documentation for this class was generated from the following file:

- cpl_vsil_curl.cpp

11.111 VSICurlHandle Class Reference

Inheritance diagram for VSICurlHandle:



The documentation for this class was generated from the following file:

- cpl_vsil_curl.cpp

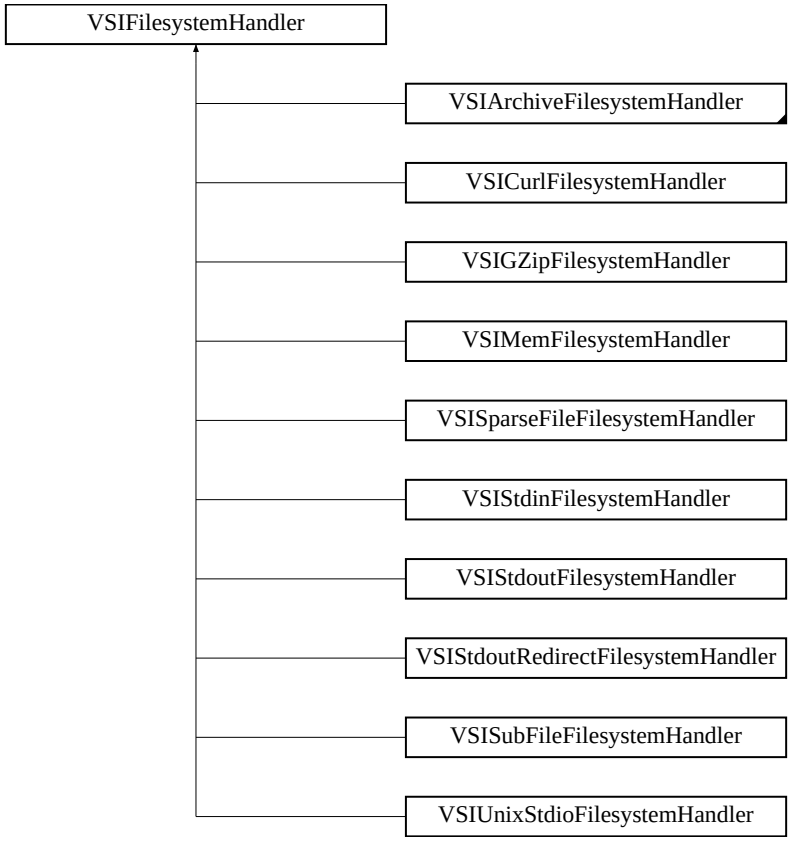
11.112 VSIFileManager Class Reference

The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil.cpp

11.113 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:

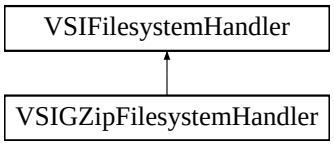


The documentation for this class was generated from the following file:

- cpl_vsi_virtual.h

11.114 VSIGZipFilesystemHandler Class Reference

Inheritance diagram for VSIGZipFilesystemHandler:

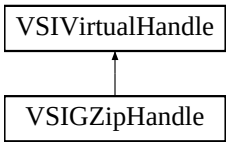


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.115 VSIGZipHandle Class Reference

Inheritance diagram for VSIGZipHandle:

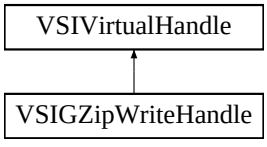


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.116 VSIGZipWriteHandle Class Reference

Inheritance diagram for VSIGZipWriteHandle:



The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

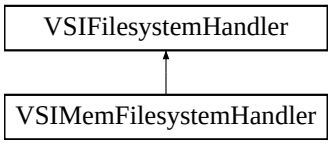
11.117 VSIMemFile Class Reference

The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

11.118 VSIMemFilesystemHandler Class Reference

Inheritance diagram for VSIMemFilesystemHandler:

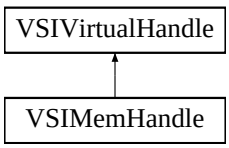


The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

11.119 VSIMemHandle Class Reference

Inheritance diagram for VSIMemHandle:

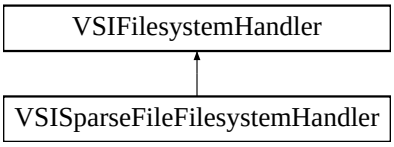


The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

11.120 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

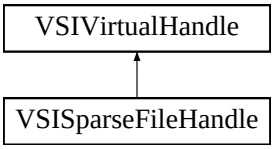


The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

11.121 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:

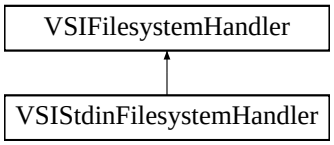


The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

11.122 VSIStdinFilesystemHandler Class Reference

Inheritance diagram for VSIStdinFilesystemHandler:

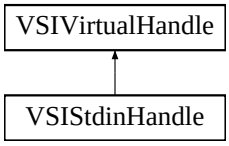


The documentation for this class was generated from the following file:

- cpl_vsil_stdin.cpp

11.123 VSIStdinHandle Class Reference

Inheritance diagram for VSIStdinHandle:

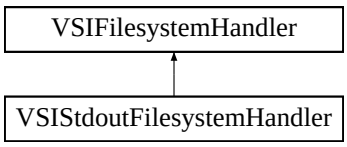


The documentation for this class was generated from the following file:

- cpl_vsil_stdin.cpp

11.124 VSIStdoutFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutFilesystemHandler:

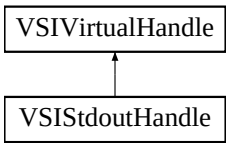


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

11.125 VSIStdoutHandle Class Reference

Inheritance diagram for VSIStdoutHandle:

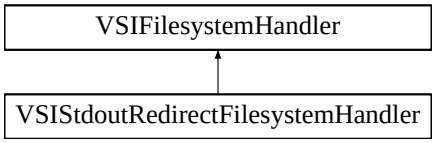


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

11.126 VSIStdoutRedirectFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutRedirectFilesystemHandler:

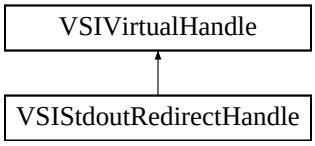


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

11.127 VSIStdoutRedirectHandle Class Reference

Inheritance diagram for VSIStdoutRedirectHandle:

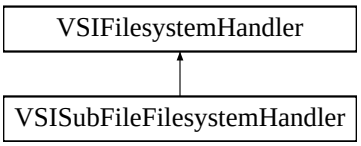


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

11.128 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

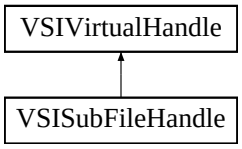


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

11.129 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:

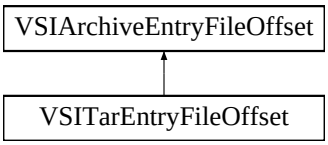


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

11.130 VSITarEntryFileOffset Class Reference

Inheritance diagram for VSITarEntryFileOffset:

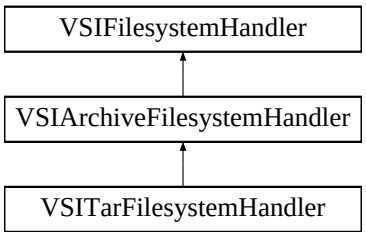


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

11.131 VSITarFilesystemHandler Class Reference

Inheritance diagram for VSITarFilesystemHandler:

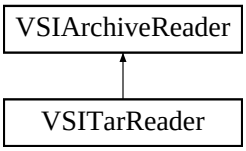


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

11.132 VSITarReader Class Reference

Inheritance diagram for VSITarReader:

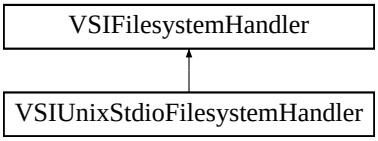


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

11.133 VSIUnixStdioFilesystemHandler Class Reference

Inheritance diagram for VSIUnixStdioFilesystemHandler:

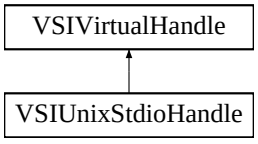


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

11.134 VSIUnixStdioHandle Class Reference

Inheritance diagram for VSIUnixStdioHandle:

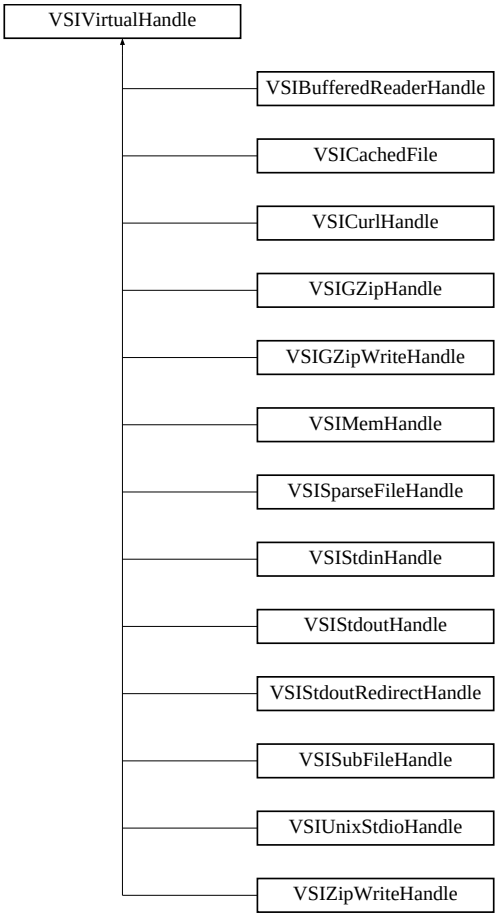


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

11.135 VSIVirtualHandle Class Reference

Inheritance diagram for VSIVirtualHandle:

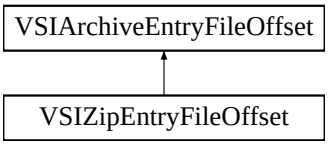


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil.cpp

11.136 VSIZipEntryFileOffset Class Reference

Inheritance diagram for VSIZipEntryFileOffset:

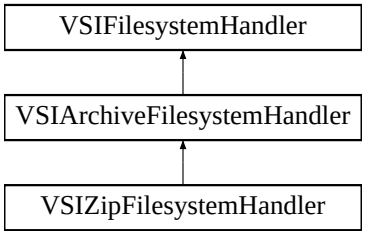


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.137 VSIZipFilesystemHandler Class Reference

Inheritance diagram for VSIZipFilesystemHandler:

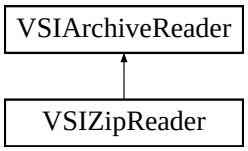


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.138 VSIZipReader Class Reference

Inheritance diagram for VSIZipReader:

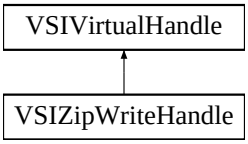


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.139 VSIZipWriteHandle Class Reference

Inheritance diagram for VSIZipWriteHandle:



The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

11.140 WriteFuncStruct Struct Reference

The documentation for this struct was generated from the following file:

- cpl_vsil_curl.cpp

11.141 yyalloc Union Reference

The documentation for this union was generated from the following file:

- swq_parser.cpp

11.142 zip_fileinfo Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.h

11.143 zip_internal Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_zip.cpp

11.144 zlib_filefunc_def_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl_minizip_ioapi.h

Chapter 12

File Documentation

12.1 cpl_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

Classes

- struct **CPLSharedFileInfo**
- class **CPLLocaleC**

Functions

- const char * **CPLGetConfigOption** (const char *, const char *)
- void **CPLSetConfigOption** (const char *, const char *)
- void **CPLSetThreadLocalConfigOption** (const char *pszKey, const char *pszValue)
- void * **CPLMalloc** (size_t)
- void * **CPLCalloc** (size_t, size_t)
- void * **CPLRealloc** (void *, size_t)
- char * **CPLStrdup** (const char *)
- char * **CPLStrlwr** (char *)
- char * **CPLFGets** (char *, int, FILE *)
- const char * **CPLReadLine** (FILE *)
- const char * **CPLReadLineL** (VSILFILE *)
- const char * **CPLReadLine2L** (VSILFILE *, int nMaxCols, char **papszOptions)
- double **CPLAtof** (const char *)
- double **CPLAtofDelim** (const char *, char)
- double **CPLStrtod** (const char *, char **)
- double **CPLStrtodDelim** (const char *, char **, char)
- float **CPLStrtof** (const char *, char **)
- float **CPLStrtofDelim** (const char *, char **, char)
- double **CPLAtofM** (const char *)
- char * **CPLScanString** (const char *, int, int, int)
- double **CPLScanDouble** (const char *, int)
- long **CPLScanLong** (const char *, int)
- unsigned long **CPLScanULong** (const char *, int)
- GUIntBig **CPLScanUIntBig** (const char *, int)

- void * **CPLScanPointer** (const char *, int)
- int **CPLPrintString** (char *, const char *, int)
- int **CPLPrintStringFill** (char *, const char *, int)
- int **CPLPrintInt32** (char *, GInt32, int)
- int **CPLPrintUIntBig** (char *, GUIntBig, int)
- int **CPLPrintDouble** (char *, const char *, double, const char *)
- int **CPLPrintTime** (char *, int, const char *, const struct tm *, const char *)
- int **CPLPrintPointer** (char *, void *, int)
- void * **CPLGetSymbol** (const char *, const char *)
- int **CPLGetExecPath** (char *pszPathBuf, int nMaxLength)
- const char * **CPLGetPath** (const char *)
- const char * **CPLGetDirname** (const char *)
- const char * **CPLGetFilename** (const char *)
- const char * **CPLGetBasename** (const char *)
- const char * **CPLGetExtension** (const char *)
- char * **CPLGetCurrentDir** (void)
- const char * **CPLFormFilename** (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char * **CPLFormCIFilename** (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char * **CPLResetExtension** (const char *, const char *)
- const char * **CPLProjectRelativeFilename** (const char *pszProjectDir, const char *pszSecondaryFilename)
- int **CPLIsFilenameRelative** (const char *pszFilename)
- const char * **CPLExtractRelativePath** (const char *, const char *, int *)
- const char * **CPLCleanTrailingSlash** (const char *)
- char ** **CPLCorrespondingPaths** (const char *pszOldFilename, const char *pszNewFilename, char **papszFileList)
- int **CPLCheckForFile** (char *pszFilename, char **papszSiblingList)
- const char * **CPLGenerateTempFilename** (const char *pszStem)
- FILE * **CPLOpenShared** (const char *, const char *, int)
- void **CPLCloseShared** (FILE *)
- **CPLSharedFileInfo** * **CPLGetSharedList** (int *)
- void **CPLDumpSharedList** (FILE *)
- double **CPLPackedDMSToDec** (double)
- double **CPLDecToPackedDMS** (double dfDec)
- int **CPLUnlinkTree** (const char *)

12.1.1 Detailed Description

Various convenience functions for CPL.

12.1.2 Function Documentation

12.1.2.1 double CPLAtof (const char * *nptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtod(nptr, (char **)NULL);

This function does the same as standard atof(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLAtofDelim()** (p. 313) function if you want to specify custom delimiter.

IMPORTANT NOTE. Existance of this function does not mean you should always use it. Sometimes you should use standard locale aware atof(3) and its family. When you need to process the user's input (for example, command line parameters) use atof(3), because user works in localized environment and her input will be done accordingly the locale set. In particular that means we should not make assumptions about character used as decimal delimiter, it can be either "." or ",". But when you are parsing some ASCII file in predefined format, you most likely need **CPLAtof()** (p. 312), because such files distributed across the systems with different locales and floating point representation shoudl be considered as a part of file format. If the format uses "." as a delimiter the same character must be used when parsing number regardless of actual locale setting.

Parameters

nptr	Pointer to string to convert.

Returns

Converted value, if any.

References CPLAtof(), and CPLStrtod().

Referenced by OGRSpatialReference::CloneGeogCS(), CPLAtof(), CPLScanDouble(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRSpatialReference::IsSameGeogCS(), OGRSpatialReference::IsSameVertCS(), OGRSpatialReference::SetGeogCS(), and OGRSpatialReference::SetStatePlane().

12.1.2.2 double CPLAtofDelim (const char * *nptr*, char *point*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtodDelim(nptr, (char **)NULL, point);

This function does the same as standard atof(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. 312) function.

Parameters

nptr	Pointer to string to convert.
point	Decimal delimiter.

Returns

Converted value, if any.

References CPLAtofDelim(), and CPLStrtodDelim().

Referenced by CPLAtofDelim().

12.1.2.3 double CPLAtofM (const char * *nptr*)

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard atof(), but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

Parameters

nptr	The string to convert.

Returns

Converted value, if any. Zero on failure.

References CPLAtofM(), and CPLStrtodDelim().

Referenced by CPLAtofM(), and OGRSpatialReference::importFromProj4().

12.1.2.4 void* CPLCalloc (size_t *nCount*, size_t *nSize*)

Safe version of calloc().

This function is like the C library calloc(), but raises a CE_Fatal error with **CPLError()** (p. 334) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSICalloc() to get the memory, so any hooking of VSICalloc() will apply to **CPLCalloc()** (p. 314) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLCalloc()** (p. 314).

Parameters

nCount	number of objects to allocate.
nSize	size (in bytes) of object to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize * nCount is NULL.

12.1.2.5 int CPLCheckForFile (char * *pszFilename*, char ** *papszSiblingFiles*)

Check for file existance.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, pszFilename is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a **VSIStatL()** (p. 384) is used to test for the files existance, and no case insensitive testing is done.

Parameters

pszFilename	name of file to check for - filename case updated in some cases.
papszSibling-Files	a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

Returns

TRUE if a match is found, or FALSE if not.

References CPLGetFilename(), and VSIStatL().

12.1.2.6 const char* CPLCleanTrailingSlash (const char * *pszPath*)

Remove trailing forward/backward slash from the path for unix/windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

Parameters

pszPath	the path to be cleaned up

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLCleanTrailingSlash().

Referenced by CPLCleanTrailingSlash().

12.1.2.7 void CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A **CPLError()** (p. 334) is issued if the file is not in the shared file list.

Parameters

fp	file handle from **CPLOpenShared()** (p. 323) to deaccess.

References VSIFCloseL().

12.1.2.8 char** CPLCorrespondingPaths (const char * *pszOldFilename,* const char * *pszNewFilename,* char ** *papszFileList*)

Identify corresponding paths.

Given a prototype old and new filename this function will attempt to determine corresponding names for a set of other old filenames that will rename them in a similar manner. This correspondance assumes there are two possibly kinds of renaming going on. A change of path, and a change of filename stem.

If a consistent renaming cannot be established for all the files this function will return indicating an error.

The returned file list becomes owned by the caller and should be destroyed with **CSLDestroy()** (p. 365).

Parameters

pszOldFilename	path to old prototype file.
pszNew-Filename	path to new prototype file.
papszFileList	list of other files associated with pszOldFilename to rename similarly.

Returns

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

References CPLCorrespondingPaths(), CPLFormFilename(), CPLGetBasename(), CPLGetFilename(), and CPLGetPath().

Referenced by CPLCorrespondingPaths().

12.1.2.9 double CPLDecToPackedDMS (double *dfDec*)

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

This function converts a value, specified in decimal degrees into packed DMS angle. The standard packed DMS format is:

degrees * 1000000 + minutes * 1000 + seconds

See also **CPLPackedDMSToDec()** (p. 323).

Parameters

dfDec	Angle in decimal degrees.

Returns

Angle in packed DMS format.

12.1.2.10 void CPLDumpSharedList (FILE * *fp*)

Report open shared files.

Dumps all open shared files to the indicated file handle. If the file handle is NULL information is sent via the **CPLDebug()** (p. 334) call.

Parameters

fp	File handle to write to.

12.1.2.11 const char* CPLExtractRelativePath (const char * *pszBaseDir,* const char * *pszTarget,* int * *pbGotRelative*)

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

Parameters

pszBaseDir	the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relitivizing.
pszTarget	the filename to be changed to be relative to pszBaseDir.
pbGotRelative	Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

Returns

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

References CPLExtractRelativePath(), and CPLIsFilenameRelative().

Referenced by CPLExtractRelativePath().

12.1.2.12 char* CPLFGets (char * *pszBuffer,* int *nBufferSize,* FILE * *fp*)

Reads in at most one less than nBufferSize characters from the fp stream and stores them into the buffer pointed to by pszBuffer. Reading stops after an EOF or a newline. If a newline is read, it is *not* stored into the buffer. A '\0' is stored after the last character in the buffer. All three types of newline terminators recognized by the **CPLFGets()** (p. 317): single '\r' and '\n' and '\r\n' combination.

Parameters

pszBuffer	pointer to the targeting character buffer.
nBufferSize	maximum size of the string to read (not including termonating '\0').
fp	file pointer to read from.

Returns

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

12.1.2.13 const char* CPLFormCIFilename (const char * *pszPath,* const char * *pszBasename,* const char * *pszExtension*)

Case insensitive file searching, returing full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivelent to **CPLFormFilename()** (p. 317) on case insensitive file systems (like Windows).

Parameters

pszPath	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
pszBasename	file basename. May optionally have path and/or extension. May not be NULL.
pszExtension	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormCIFilename(), CPLFormFilename(), VSIIsCaseSensitiveFS(), and VSIStatExL().

Referenced by CPLFormCIFilename().

12.1.2.14 const char* CPLFormFilename (const char * *pszPath,* const char * *pszBasename,* const char * *pszExtension*)

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz","def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL,"def", NULL ) == "def"
CPLFormFilename(NULL,"abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/","def.dat", NULL ) == "/abc/xyz/def.dat"
```

Parameters

pszPath	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
pszBasename	file basename. May optionally have path and/or extension. May not be NULL.
pszExtension	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormFilename().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCorrespondingPaths(), CPLFormCIFilename(), CPLFormFilename(), CPLGenerateTempFilename(), and CPLUnlinkTree().

12.1.2.15 const char* CPLGenerateTempFilename (const char * *pszStem*)

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL_TMPDIR configuration option.

Parameters

pszStem	if non-NULL this will be part of the filename.

Returns

a filename which is valid till the next CPL call in this thread.

References CPLFormFilename(), and CPLGenerateTempFilename().

Referenced by CPLGenerateTempFilename().

12.1.2.16 const char* CPLGetBasename (const char * *pszFullFilename*)

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

Parameters

pszFullFilename	the full filename potentially including a path.

Returns

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetBasename().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCorrespondingPaths(), and CPLGetBasename().

12.1.2.17 const char* CPLGetConfigOption (const char * *pszKey,* const char * *pszDefault*)

Get the value of a configuration option.

The value is the value of a (key, value) option set with **CPLSetConfigOption()** (p. 330). If the given option was no defined with **CPLSetConfigOption()** (p. 330), it tries to find it in environment variables.

Parameters

pszKey	the key of the option to retrieve
pszDefault	a default value if the key does not match existing defined options (may be NULL)

Returns

the value associated to the key, or the default value if not found

See Also

CPLSetConfigOption() (p. 330), `http://trac.osgeo.org/gdal/wiki/ConfigOptions`

12.1.2.18 char* CPLGetCurrentDir (void)

Get the current working directory name.

Returns

a pointer to buffer, containing current working directory path or NULL in case of error. User is responsible to free that buffer after usage with CPLFree() function. If HAVE_GETCWD macro is not defined, the function returns NULL.

References CPLGetCurrentDir().

Referenced by CPLGetCurrentDir().

12.1.2.19 const char* CPLGetDirname (const char * *pszFilename*)

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename the dot will be returned. It is the only difference from **CPLGetPath()** (p. 321).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters

pszFilename	the filename potentially including a path.

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetDirname().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetDirname().

12.1.2.20 int CPLGetExecPath (char * *pszPathBuf,* int *nMaxLength*)

Fetch path of executable.

The path to the executable currently running is returned. This path includes the name of the executable. Currently this only works on win32 and linux platforms. The returned path is UTF-8 encoded.

Parameters

pszPathBuf	the buffer into which the path is placed.
nMaxLength	the buffer size, MAX_PATH+1 is suggested.

Returns

FALSE on failure or TRUE on success.

References CPLGetExecPath().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetExecPath().

12.1.2.21 const char* CPLGetExtension (const char * *pszFullFilename*)

Extract filename extension from full filename.

Returns a string containing the extention portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```
CPLGetExtension( "abc/def.xyz" ) == "xyz"
CPLGetExtension( "abc/def" ) == ""
```

Parameters

pszFullFilename	the full filename potentially including a path.

Returns

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetExtension().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetExtension().

12.1.2.22 const char* CPLGetFilename (const char * *pszFullFilename*)

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

Parameters

pszFullFilename	the full filename potentially including a path.

Returns

just the non-directory portion of the path (points back into original string).

References CPLGetFilename().

Referenced by CPLCheckForFile(), CPLCorrespondingPaths(), and CPLGetFilename().

12.1.2.23 const char* CPLGetPath (const char * *pszFilename*)

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

Parameters

pszFilename	the filename potentially including a path.

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetPath().

Referenced by CPLCorrespondingPaths(), and CPLGetPath().

12.1.2.24 CPLSharedFileInfo* CPLGetSharedList (int * *pnCount*)

Fetch list of open shared files.

Parameters

pnCount	place to put the count of entries.

Returns

the pointer to the first in the array of shared file info structures.

12.1.2.25 void* CPLGetSymbol (const char * *pszLibrary*, const char * *pszSymbolName*)

Fetch a function pointer from a shared library / DLL.

This function is meant to abstract access to shared libraries and DLLs and performs functions similar to dlopen()/dlsym() on Unix and LoadLibrary() / GetProcAddress() on Windows.

If no support for loading entry points from a shared library is available this function will always return NULL. Rules on when this function issues a **CPLError()** (p. 334) or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. 321) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-prending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters

pszLibrary	the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.
pszSymbolName	the name of the function to fetch a pointer to.

Returns

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

References CPLGetSymbol().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetSymbol().

12.1.2.26 int CPLIsFilenameRelative (const char * *pszFilename*)

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

Parameters

pszFilename	the filename with path to test.

Returns

TRUE if the filename is relative or FALSE if it is absolute.

References CPLIsFilenameRelative().

Referenced by CPLExtractRelativePath(), CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

12.1.2.27 void* CPLMalloc (size_t *nSize*)

Safe version of malloc().

This function is like the C library malloc(), but raises a CE_Fatal error with **CPLError()** (p. 334) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIMalloc() to get the memory, so any hooking of VSIMalloc() will apply to **CPLMalloc()** (p. 322) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLMalloc()** (p. 322).

Parameters

nSize	size (in bytes) of memory block to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize is zero.

12.1.2.28 FILE* CPLOpenShared (const char * *pszFilename,* const char * *pszAccess,* int *bLarge*)

Open a shared file handle.

Some operating systems have limits on the number of file handles that can be open at one time. This function attempts to maintain a registry of already open file handles, and reuse existing ones if the same file is requested by another part of the application.

Note that access is only shared for access types "r", "rb", "r+" and "rb+". All others will just result in direct VSIOpen() calls. Keep in mind that a file is only reused if the file name is exactly the same. Different names referring to the same file will result in different handles.

The VSIFOpen() or **VSIFOpenL()** (p. 373) function is used to actually open the file, when an existing file handle can't be shared.

Parameters

pszFilename	the name of the file to open.
pszAccess	the normal fopen()/VSIFOpen() style access string.
bLarge	If TRUE **VSIFOpenL()** (p. 373) (for large files) will be used instead of VSIFOpen().

Returns

a file handle or NULL if opening fails.

References VSIFOpenL().

12.1.2.29 double CPLPackedDMSToDec (double *dfPacked*)

Convert a packed DMS value (DDDMMMSSS.SS) into decimal degrees.

This function converts a packed DMS angle to seconds. The standard packed DMS format is:

degrees * 1000000 + minutes * 1000 + seconds

Example: ang = 120025045.25 yields deg = 120 min = 25 sec = 45.25

The algorithm used for the conversion is as follows:

1. The absolute value of the angle is used.
1. The degrees are separated out: deg = ang/1000000 (fractional portion truncated)
1. The minutes are separated out: min = (ang - deg * 1000000) / 1000 (fractional portion truncated)
1. The seconds are then computed: sec = ang - deg * 1000000 - min * 1000
1. The total angle in seconds is computed: sec = deg * 3600.0 + min * 60.0 + sec
1. The sign of sec is set to that of the input angle.

Packed DMS values used by the USGS GCTP package and probably by other software.

NOTE: This code does not validate input value. If you give the wrong value, you will get the wrong result.

Parameters

dfPacked	Angle in packed DMS format.

Returns

Angle in decimal degrees.

12.1.2.30 int CPLPrintDouble (char * *pszBuffer,* const char * *pszFormat,* double *dfValue,* const char * *pszLocale*)

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
pszFormat	Format specifier (for example, "%16.9E").
dfValue	Numerical value to print.
pszLocale	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. With the pszLocale option we can control what exact locale will be used for printing a numeric value to the string (in most cases it should be C/POSIX).

Returns

Number of characters printed.

12.1.2.31 int CPLPrintInt32 (char * *pszBuffer,* GInt32 *iValue,* int *nMaxLen*)

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

12.1.2.32 int CPLPrintPointer (char * *pszBuffer,* void * *pValue,* int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
pValue	Pointer to ASCII encode.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

12.1.2.33 int CPLPrintString (char * *pszDest,* const char * *pszSrc,* int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest.

Parameters

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

12.1.2.34 int CPLPrintStringFill (char * *pszDest,* const char * *pszSrc,* int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

Parameters

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

12.1.2.35 int CPLPrintTime (char * *pszBuffer,* int *nMaxLen,* const char * *pszFormat,* const struct tm * *poBrokenTime,* const char * *pszLocale*)

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.
pszFormat	Controls the output format. Options are the same as for strftime(3) function.
poBrokenTime	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.

pszLocale	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be ajusted differently from the C/POSIX defaults. To solve these problems this option was introdiced.

Returns

Number of characters printed.

12.1.2.36 int CPLPrintUIntBig (char * *pszBuffer,* GUIntBig *iValue,* int *nMaxLen*)

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

12.1.2.37 const char* CPLProjectRelativeFilename (const char * *pszProjectDir,* const char * *pszSecondaryFilename*)

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def","tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def","/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def","../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\WIN","abc.gif") == "C:\WIN\abc.gif"
```

Parameters

pszProjectDir	the directory relative to which the secondary files path should be interpreted.
pszSecondary-Filename	the filename (potentially with path) that is to be interpreted relative to the project directory.

Returns

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

References CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

Referenced by CPLProjectRelativeFilename().

12.1.2.38 const char* CPLReadLine (FILE * *fp*)

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivelent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on it's value past the next call to **CPLReadLine()** (p. 327).

Note that **CPLReadLine()** (p. 327) uses VSIFGets(), so any hooking of VSI file services should apply to **CPLReadLine()** (p. 327) as well.

CPLReadLine() (p. 327) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **CPLReadLine()** (p. 327) may be called with a NULL FILE * at any time to free this working buffer.

Parameters

fp	file pointer opened with VSIFOpen().

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

12.1.2.39 const char* CPLReadLine2L (VSILFILE * *fp*, int *nMaxCars*, char ** *papszOptions*)

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. 327), but reading from a large file API handle.

Parameters

fp	file pointer opened with **VSIFOpenL()** (p. 373).
nMaxCars	maximum number of characters allowed, or -1 for no limit.
papszOptions	NULL-terminated array of options. Unused for now.

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered or the maximum number of characters allowed readched.

Since

GDAL 1.7.0

References VSIFReadL(), VSIFSeekL(), and VSIFTellL().

12.1.2.40 const char* CPLReadLineL (VSILFILE * *fp*)

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. 327), but reading from a large file API handle.

Parameters

fp	file pointer opened with **VSIFOpenL()** (p. 373).

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

12.1.2.41 void* CPLRealloc (void * *pData*, size_t *nNewSize*)

Safe version of realloc().

This function is like the C library realloc(), but raises a CE_Fatal error with **CPLError()** (p. 334) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIRealloc() to get the memory, so any hooking of VSIRealloc() will apply to **CPLRealloc()** (p. 328) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLRealloc()** (p. 328).

It is also safe to pass NULL in as the existing memory block for **CPLRealloc()** (p. 328), in which case it uses VSIMalloc() to allocate a new block.

Parameters

pData	existing memory block which should be copied to the new block.
nNewSize	new size (in bytes) of memory block to allocate.

Returns

pointer to allocated memory, only NULL if nNewSize is zero.

12.1.2.42 const char* CPLResetExtension (const char * *pszPath*, const char * *pszExt*)

Replace the extension with the provided one.

Parameters

pszPath	the input path, this string is not altered.
pszExt	the new extension to apply to the given path.

Returns

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLResetExtension().

Referenced by CPLResetExtension().

12.1.2.43 double CPLScanDouble (const char * *pszString*, int *nMaxLength*)

Extract double from string.

Scan up to a maximum number of characters from a string and convert the result to a double. This function uses **CPLAtof()** (p. 312) to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Double value, converted from its ASCII form.

References CPLAtof().

12.1.2.44 long CPLScanLong (const char * *pszString,* int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a long.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Long value, converted from its ASCII form.

12.1.2.45 void* CPLScanPointer (const char * *pszString,* int *nMaxLength*)

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

pointer value, converted from its ASCII form.

12.1.2.46 char* CPLScanString (const char * *pszString,* int *nMaxLength,* int *bTrimSpaces,* int *bNormalize*)

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to read. Less characters will be read if a null character is encountered.
bTrimSpaces	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
bNormalize	If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

Returns

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

12.1.2.47 GUIntBig CPLScanUIntBig (const char * *pszString,* int *nMaxLength*)

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

GUIntBig value, converted from its ASCII form.

12.1.2.48 unsigned long CPLScanULong (const char * *pszString,* int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Unsigned long value, converted from its ASCII form.

12.1.2.49 void CPLSetConfigOption (const char * *pszKey,* const char * *pszValue*)

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. 319) method.

This mechanism is similar to environment variables, but options set with **CPLSetConfigOption()** (p. 330) overrides, for **CPLGetConfigOption()** (p. 319) point of view, values defined in the environment.

If **CPLSetConfigOption()** (p. 330) is called several times with the same key, the value provided during the last call will be used.

Options can also be passed on the command line of most GDAL utilities with the with '–config KEY VALUE'. For example, ogrinfo –config CPL_DEBUG ON ~/data/test/point.shp

Parameters

pszKey	the key of the option
pszValue	the value of the option, or NULL to clear a setting.

See Also

`http://trac.osgeo.org/gdal/wiki/ConfigOptions`

12.1.2.50 void CPLSetThreadLocalConfigOption (const char * *pszKey,* const char * *pszValue*)

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. 319) method.

This function sets the configuration option that only applies in the current thread, as opposed to **CPLSetConfigOption()** (p. 330) which sets an option that applies on all threads.

Parameters

pszKey	the key of the option
pszValue	the value of the option, or NULL to clear a setting.

12.1.2.51 char* CPLStrdup (const char * *pszString*)

Safe version of strdup() function.

This function is similar to the C library strdup() function, but if the memory allocation fails it will issue a CE_Fatal error with **CPLError()** (p. 334) instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to **CPLStrdup()** (p. 331) as well. Memory allocated with **CPLStrdup()** (p. 331) can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into **CPLStrdup()** (p. 331). **CPLStrdup()** (p. 331) will allocate and return a zero length string (as opposed to a NULL string).

Parameters

pszString	input string to be duplicated. May be NULL.

Returns

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

12.1.2.52 char* CPLStrlwr (char * *pszString*)

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

Parameters

pszString	input string to be converted.

Returns

pointer to the same string, pszString.

12.1.2.53 double CPLStrtod (const char * *nptr,* char ** *endptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard strtod(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtodDelim()** (p. 332) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 312) function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns

Converted value, if any.

References CPLStrtod(), and CPLStrtodDelim().

Referenced by CPLAtof(), and CPLStrtod().

12.1.2.54 double CPLStrtodDelim (const char * *nptr*, char ** *endptr*, char *point*)

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard strtod(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. 312) function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
point	Decimal delimiter.

Returns

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), CPLStrtodDelim(), and CPLStrtofDelim().

12.1.2.55 float CPLStrtof (const char * *nptr*, char ** *endptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to single floating point representation. This function does the same as standard strtof(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtofDelim()** (p. 332) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 312) function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns

Converted value, if any.

References CPLStrtof(), and CPLStrtofDelim().

Referenced by CPLStrtof().

12.1.2.56 float CPLStrtofDelim (const char * *nptr*, char ** *endptr*, char *point*)

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by nptr to single floating point representation. This function does the same as standard strtof(3), but does not take locale in account. Instead of locale defined decimal

delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. 312) function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
point	Decimal delimiter.

Returns

Converted value, if any.

References CPLStrtodDelim(), and CPLStrtofDelim().

Referenced by CPLStrtof(), and CPLStrtofDelim().

12.1.2.57 int CPLUnlinkTree (const char * *pszPath*)

Returns

0 on successful completion, -1 if function fails.

References CPLFormFilename(), VSIRmdir(), and VSIUnlink().

12.2 cpl_error.h File Reference

```
#include "cpl_port.h"
```

Functions

- void **CPLError** (CPLErr eErrClass, int err_no, const char *fmt,...)
- void **CPLEmergencyError** (const char *)
- void **CPLErrorReset** (void)
- int **CPLGetLastErrorNo** (void)
- CPLErr **CPLGetLastErrorType** (void)
- const char * **CPLGetLastErrorMsg** (void)
- void * **CPLGetErrorHandlerUserData** (void)
- CPLErrorHandler **CPLSetErrorHandler** (CPLErrorHandler)
- CPLErrorHandler **CPLSetErrorHandlerEx** (CPLErrorHandler, void *)
- void **CPLPushErrorHandler** (CPLErrorHandler)
- void **CPLPushErrorHandlerEx** (CPLErrorHandler, void *)
- void **CPLPopErrorHandler** (void)
- void **CPLDebug** (const char *, const char *,...)
- void **_CPLAssert** (const char *, const char *, int)

12.2.1 Detailed Description

CPL error handling services.

12.2.2 Function Documentation

12.2.2.1 void _CPLAssert (const char * *pszExpression,* const char * *pszFile,* int *iLine*)

Report failure of a logical assertion.

Applications would normally use the CPLAssert() macro which expands into code calling **_CPLAssert()** (p. 334) only if the condition fails. **_CPLAssert()** (p. 334) will generate a CE_Fatal error call to **CPLError()** (p. 334), indicating the file name, and line number of the failed assertion, as well as containing the assertion itself.

There is no reason for application code to call **_CPLAssert()** (p. 334) directly.

12.2.2.2 void CPLDebug (const char * *pszCategory,* const char * *pszFormat,* ...)

Display a debugging message.

The category argument is used in conjunction with the CPL_DEBUG environment variable to establish if the message should be displayed. If the CPL_DEBUG environment variable is not set, no debug messages are emitted (use CPLError(CE_Warning,...) to ensure messages are displayed). If CPL_DEBUG is set, but is an empty string or the word "ON" then all debug messages are shown. Otherwise only messages whose category appears somewhere within the CPL_DEBUG value are displayed (as determinted by strstr()).

Categories are usually an identifier for the subsystem producing the error. For instance "GDAL" might be used for the GDAL core, and "TIFF" for messages from the TIFF translator.

Parameters

pszCategory	name of the debugging message category.
pszFormat	printf() style format string for message to display. Remaining arguments are assumed to be for format.

12.2.2.3 void CPLEmergencyError (const char * *pszMessage*)

Fatal error when things are bad.

This function should be called in an emergency situation where it is unlikely that a regular error report would work. This would include in the case of heap exhaustion for even small allocations, or any failure in the process of reporting an error (such as TLS allocations).

This function should never return. After the error message has been reported as best possible, the application will abort() similarly to how **CPLError()** (p. 334) aborts on CE_Fatal class errors.

Parameters

pszMessage	the error message to report.

12.2.2.4 void CPLError (CPLErr *eErrClass,* int *err_no,* const char * *fmt,* ...)

Report an error.

This function reports an error in a manner that can be hooked and reported appropriate by different applications.

The effect of this function can be altered by applications by installing a custom error handling using **CPLSetErrorHandler()** (p. 336).

The eErrClass argument can have the value CE_Warning indicating that the message is an informational warning, CE_Failure indicating that the action failed, but that normal recover mechanisms will be used or CE_Fatal meaning that a fatal error has occured, and that **CPLError()** (p. 334) should not return.

The default behaviour of **CPLError()** (p. 334) is to report errors to stderr, and to abort() after reporting a CE_Fatal

error. It is expected that some applications will want to supress error reporting, and will want to install a C++ exception, or longjmp() approach to no local fatal error recovery.

Regardless of how application error handlers or the default error handler choose to handle an error, the error number, and message will be stored for recovery with **CPLGetLastErrorNo()** (p. 335) and **CPLGetLastErrorMsg()** (p. 335).

Parameters

eErrClass	one of CE_Warning, CE_Failure or CE_Fatal.
err_no	the error number (CPLE_∗) from **cpl_error.h** (p. 333).
fmt	a printf() style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to printf().

12.2.2.5 void CPLErrorReset (void)

Erase any traces of previous errors.

This is normally used to ensure that an error which has been recovered from does not appear to be still in play with high level functions.

12.2.2.6 void∗ CPLGetErrorHandlerUserData (void)

Fetch the user data for the error context

Fetches the user data for the current error context. You can set the user data for the error context when you add your handler by issuing **CPLSetErrorHandlerEx()** (p. 337) and **CPLPushErrorHandlerEx()** (p. 336). Note that user data is primarily intended for providing context within error handlers themselves, but they could potentially be abused in other useful ways with the usual caveat emptor understanding.

Returns

the user data pointer for the error context

12.2.2.7 const char∗ CPLGetLastErrorMsg (void)

Get the last error message.

Fetches the last error message posted with **CPLError()** (p. 334), that hasn't been cleared by **CPLErrorReset()** (p. 335). The returned pointer is to an internal string that should not be altered or freed.

Returns

the last error message, or NULL if there is no posted error message.

12.2.2.8 int CPLGetLastErrorNo (void)

Fetch the last error number.

Fetches the last error number posted with **CPLError()** (p. 334), that hasn't been cleared by **CPLErrorReset()** (p. 335). This is the error number, not the error class.

Returns

the error number of the last error to occur, or CPLE_None (0) if there are no posted errors.

12.2.2.9 CPLErr CPLGetLastErrorType (void)

Fetch the last error type.

Fetches the last error type posted with **CPLError()** (p. 334), that hasn't been cleared by **CPLErrorReset()** (p. 335). This is the error class, not the error number.

Returns

the error type of the last error to occur, or CE_None (0) if there are no posted errors.

12.2.2.10 void CPLPopErrorHandler (void)

Pop error handler off stack.

Discards the current error handler on the error handler stack, and restores the one in use before the last **CPLPushErrorHandler()** (p. 336) call. This method has no effect if there are no error handlers on the current threads error handler stack.

12.2.2.11 void CPLPushErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Push a new CPLError handler.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with **CPLPopErrorHandler()** (p. 336).

The **CPLSetErrorHandler()** (p. 336) docs have further information on how CPLError handlers work.

Parameters

pfnErrorHandlerNew	new error handler function.

12.2.2.12 void CPLPushErrorHandlerEx (CPLErrorHandler *pfnErrorHandlerNew*, void * *pUserData*)

Push a new CPLError handler with user data on the error context.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with **CPLPopErrorHandler()** (p. 336). Obtain the user data back by using CPLGetErrorContext().

The **CPLSetErrorHandler()** (p. 336) docs have further information on how CPLError handlers work.

Parameters

pfnErrorHandlerNew	new error handler function.
pUserData	User data to put on the error context.

12.2.2.13 CPLErrorHandler CPLSetErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Install custom error handler.

Allow the library's user to specify his own error handler function. A valid error handler is a C function with the following prototype:

```
void MyErrorHandler(CPLErr eErrClass, int err_no, const char *msg)
```

Pass NULL to come back to the default behavior. The default behaviour (CPLDefaultErrorHandler()) is to write the

message to stderr.

The msg will be a partially formatted error message not containing the "ERROR %d:" portion emitted by the default handler. Message formatting is handled by **CPLError()** (p. 334) before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then **CPLError()** (p. 334) will call abort(). Applications wanting to interrupt this fatal behaviour will have to use longjmp(), or a C++ exception to indirectly exit the function.

Another standard error handler is CPLQuietErrorHandler() which doesn't make any attempt to report the passed error or warning messages but will process debug messages via CPLDefaultErrorHandler.

Note that error handlers set with **CPLSetErrorHandler()** (p. 336) apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPushErrorHandler (and not removed with CPLPopErrorHandler) take precidence over the global error handlers set with **CPLSetErrorHandler()** (p. 336). Generally speaking **CPLSetErrorHandler()** (p. 336) would be used to set a desired global error handler, while **CPLPushErrorHandler()** (p. 336) would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

Parameters

pfnErrorHandlerNew	new error handler function.

Returns

returns the previously installed error handler.

12.2.2.14 CPLErrorHandler CPLSetErrorHandlerEx (CPLErrorHandler *pfnErrorHandlerNew,* void * *pUserData*)

Install custom error handle with user's data. This method is essentially CPLSetErrorHandler with an added pointer to pUserData. The pUserData is not returned in the CPLErrorHandler, however, and must be fetched via CPLGetLastErrorUserData

Parameters

pfnErrorHandlerNew	new error handler function.
pUserData	User data to carry along with the error context.

Returns

returns the previously installed error handler.

12.3 cpl_hash_set.h File Reference

```
#include "cpl_port.h"
```

Functions

- **CPLHashSet** * **CPLHashSetNew** (CPLHashSetHashFunc fnHashFunc, CPLHashSetEqualFunc fnEqualFunc, CPLHashSetFreeEltFunc fnFreeEltFunc)
- void **CPLHashSetDestroy** (**CPLHashSet** *set)
- int **CPLHashSetSize** (const **CPLHashSet** *set)
- void **CPLHashSetForeach** (**CPLHashSet** *set, CPLHashSetIterEltFunc fnIterFunc, void *user_data)
- int **CPLHashSetInsert** (**CPLHashSet** *set, void *elt)
- void * **CPLHashSetLookup** (**CPLHashSet** *set, const void *elt)

- int **CPLHashSetRemove** (**CPLHashSet** ∗set, const void ∗elt)
- unsigned long **CPLHashSetHashPointer** (const void ∗elt)
- int **CPLHashSetEqualPointer** (const void ∗elt1, const void ∗elt2)
- unsigned long **CPLHashSetHashStr** (const void ∗pszStr)
- int **CPLHashSetEqualStr** (const void ∗pszStr1, const void ∗pszStr2)

12.3.1 Detailed Description

Hash set implementation.

An hash set is a data structure that holds elements that are unique according to a comparison function. Operations on the hash set, such as insertion, removal or lookup, are supposed to be fast if an efficient "hash" function is provided.

12.3.2 Function Documentation

12.3.2.1 void CPLHashSetDestroy (CPLHashSet ∗ *set*)

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters

set	the hash set

References _CPLList::pData, and _CPLList::psNext.

12.3.2.2 int CPLHashSetEqualPointer (const void ∗ *elt1*, const void ∗ *elt2*)

Equality function for arbitrary pointers

Parameters

elt1	the first arbitrary pointer to compare
elt2	the second arbitrary pointer to compare

Returns

TRUE if the pointers are equal

12.3.2.3 int CPLHashSetEqualStr (const void ∗ *elt1*, const void ∗ *elt2*)

Equality function for strings

Parameters

elt1	the first string to compare. May be NULL.
elt2	the second string to compare. May be NULL.

Returns

TRUE if the strings are equal

12.3.2.4 void CPLHashSetForeach (CPLHashSet * set, CPLHashSetIterEltFunc fnIterFunc, void * user_data)

Walk through the hash set and runs the provided function on all the elements

This function is provided the user_data argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

Parameters

set	the hash set.
fnIterFunc	the function called on each element.
user_data	the user data provided to the function.

References _CPLList::pData, and _CPLList::psNext.

12.3.2.5 unsigned long CPLHashSetHashPointer (const void * elt)

Hash function for an arbitrary pointer

Parameters

elt	the arbitrary pointer to hash

Returns

the hash value of the pointer

12.3.2.6 unsigned long CPLHashSetHashStr (const void * elt)

Hash function for a zero-terminated string

Parameters

elt	the string to hash. May be NULL.

Returns

the hash value of the string

12.3.2.7 int CPLHashSetInsert (CPLHashSet * set, void * elt)

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

Parameters

set	the hash set
elt	the new element to insert in the hash set

Returns

TRUE if the element was not already in the hash set

12.3.2.8 void* CPLHashSetLookup (CPLHashSet * *set*, const void * *elt*)

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

Parameters

set	the hash set
elt	the element to look up in the hash set

Returns

the element found in the hash set or NULL

12.3.2.9 CPLHashSet* CPLHashSetNew (CPLHashSetHashFunc *fnHashFunc*, CPLHashSetEqualFunc *fnEqualFunc*, CPLHashSetFreeEltFunc *fnFreeEltFunc*)

Creates a new hash set

The hash function must return a hash value for the elements to insert. If fnHashFunc is NULL, CPLHashSetHashPointer will be used.

The equal function must return if two elements are equal. If fnEqualFunc is NULL, CPLHashSetEqualPointer will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If fnFreeEltFunc is NULL, elements inserted into the hash set will not be freed.

Parameters

fnHashFunc	hash function. May be NULL.
fnEqualFunc	equal function. May be NULL.
fnFreeEltFunc	element free function. May be NULL.

Returns

a new hash set

12.3.2.10 int CPLHashSetRemove (CPLHashSet * *set*, const void * *elt*)

Removes an element from a hash set

Parameters

set	the hash set
elt	the new element to remove from the hash set

Returns

TRUE if the element was in the hash set

References _CPLList::pData, and _CPLList::psNext.

12.3.2.11 int CPLHashSetSize (const CPLHashSet * *set*)

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

Parameters

set	the hash set

Returns

the number of elements in the hash set

12.4 cpl_http.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "cpl_vsi.h"
```

Classes

- struct **CPLMimePart**
- struct **CPLHTTPResult**

Functions

- int **CPLHTTPEnabled** (void)

  Return if CPLHTTP services can be usefull.
- **CPLHTTPResult** ∗ **CPLHTTPFetch** (const char ∗pszURL, char ∗∗papszOptions)

  Fetch a document from an url and return in a string.
- void **CPLHTTPCleanup** (void)

  Cleanup function to call at application termination.
- void **CPLHTTPDestroyResult** (**CPLHTTPResult** ∗psResult)

  *Clean the memory associated with the return value of **CPLHTTPFetch()** (p. 342)*
- int **CPLHTTPParseMultipartMime** (**CPLHTTPResult** ∗psResult)

  Parses a a MIME multipart message.

12.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

12.4.2 Function Documentation

12.4.2.1 void CPLHTTPDestroyResult (CPLHTTPResult ∗ *psResult*)

Clean the memory associated with the return value of **CPLHTTPFetch()** (p. 342)

Parameters

psResult	pointer to the return value of **CPLHTTPFetch()** (p. 342)

References CPLHTTPResult::nMimePartCount, CPLHTTPResult::pabyData, CPLMimePart::papszHeaders, CPLHTTPResult::papszHeaders, CPLHTTPResult::pasMimePart, CPLHTTPResult::pszContentType, and CPLHTTPResult::pszErrBuf.

12.4.2.2 int CPLHTTPEnabled (void)

Return if CPLHTTP services can be usefull.

Those services depend on GDAL being build with libcurl support.

Returns

TRUE if libcurl support is enabled

12.4.2.3 CPLHTTPResult* CPLHTTPFetch (const char * *pszURL,* char ** *papszOptions*)

Fetch a document from an url and return in a string.

Parameters

pszURL	valid URL recognized by underlying download library (libcurl)
papszOptions	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : • TIMEOUT=val, where val is in seconds • HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwkt" • HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use. • USERPWD=userid:password to specify a user and password for authentication • POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request. • PROXY=val, to make requests go through a proxy server, where val is of the form proxy.server.com:port_number • PROXYUSERPWD=val, where val is of the form username:password • CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL >= 1.9.0)

Alternatively, if not defined in the papszOptions arguments, the PROXY and PROXYUSERPWD values are searched in the configuration options named GDAL_HTTP_PROXY and GDAL_HTTP_PROXYUSERPWD, as proxy configuration belongs to networking setup and makes more sense at the configuration option level than at the connection level.

Returns

a CPLHTTPResult* structure that must be freed by **CPLHTTPDestroyResult()** (p. 341), or NULL if libcurl support is disabled

References CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHTTPResult::papszHeaders, CPLHTTPResult::pszContentType, and CPLHTTPResult::pszErrBuf.

12.4.2.4 int CPLHTTPParseMultipartMime (CPLHTTPResult * *psResult*)

Parses a a MIME multipart message.

This function will iterate over each part and put it in a separate element of the pasMimePart array of the provided psResult structure.

Parameters

psResult	pointer to the return value of **CPLHTTPFetch()** (p. 342)

Returns

TRUE if the message contains MIME multipart message.

References CPLMimePart::nDataLen, CPLHTTPResult::nDataLen, CPLHTTPResult::nMimePartCount, CPLMimePart::pabyData, CPLHTTPResult::pabyData, CPLMimePart::papszHeaders, CPLHTTPResult::pasMimePart, and CPLHTTPResult::pszContentType.

12.5 cpl_list.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **_CPLList**

Typedefs

- typedef struct **_CPLList** **CPLList**

Functions

- **CPLList** * **CPLListAppend** (**CPLList** *psList, void *pData)
- **CPLList** * **CPLListInsert** (**CPLList** *psList, void *pData, int nPosition)
- **CPLList** * **CPLListGetLast** (**CPLList** *psList)
- **CPLList** * **CPLListGet** (**CPLList** *psList, int nPosition)
- int **CPLListCount** (**CPLList** *psList)
- **CPLList** * **CPLListRemove** (**CPLList** *psList, int nPosition)
- void **CPLListDestroy** (**CPLList** *psList)
- **CPLList** * **CPLListGetNext** (**CPLList** *psElement)
- void * **CPLListGetData** (**CPLList** *psElement)

12.5.1 Detailed Description

Simplest list implementation. List contains only pointers to stored objects, not objects itself. All operations regarding allocation and freeing memory for objects should be performed by the caller.

12.5.2 Typedef Documentation

12.5.2.1 typedef struct _CPLList CPLList

List element structure.

12.5.3 Function Documentation

12.5.3.1 CPLList* CPLListAppend (CPLList * *psList*, void * *pData*)

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

Parameters

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.

Returns

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

12.5.3.2 int CPLListCount (CPLList * *psList*)

Return the number of elements in a list.

Parameters

psList	pointer to list head.

Returns

number of elements in a list.

References _CPLList::psNext.

12.5.3.3 void CPLListDestroy (CPLList * *psList*)

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters

psList	pointer to list head.

References _CPLList::psNext.

12.5.3.4 CPLList* CPLListGet (CPLList * *psList*, int *nPosition*)

Return the pointer to the specified element in a list.

Parameters

psList	pointer to list head.
nPosition	the index of the element in the list, 0 being the first element

Returns

pointer to the specified element in a list.

References _CPLList::psNext.

12.5.3.5 void* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters

psElement	pointer to list element.

Returns

pointer to the data object contained in given list element.

References _CPLList::pData.

12.5.3.6 CPLList* CPLListGetLast (CPLList * *psList*)

Return the pointer to last element in a list.

Parameters

psList	pointer to list head.

Returns

pointer to last element in a list.

References _CPLList::psNext.

12.5.3.7 CPLList* CPLListGetNext (CPLList * *psElement*)

Return the pointer to next element in a list.

Parameters

psElement	pointer to list element.

Returns

pointer to the list element preceded by the given element.

References _CPLList::psNext.

12.5.3.8 CPLList* CPLListInsert (CPLList * *psList,* void * *pData,* int *nPosition*)

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

Parameters

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.
nPosition	position number to insert an object.

Returns

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

12.5.3.9 CPLList* CPLListRemove (CPLList * *psList,* int *nPosition*)

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

Parameters

psList	pointer to list head.
nPosition	position number to delet an element.

Returns

pointer to the head of modified list.

References _CPLList::psNext.

12.6 cpl_minixml.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLXMLNode**

Typedefs

- typedef struct **CPLXMLNode** **CPLXMLNode**

Enumerations

- enum **CPLXMLNodeType** {
  CXT_Element = 0, **CXT_Text** = 1, **CXT_Attribute** = 2, **CXT_Comment** = 3,
  CXT_Literal = 4 }

Functions

- **CPLXMLNode** * **CPLParseXMLString** (const char *)

  Parse an XML string into tree form.
- void **CPLDestroyXMLNode** (**CPLXMLNode** *)

  Destroy a tree.
- **CPLXMLNode** * **CPLGetXMLNode** (**CPLXMLNode** *poRoot, const char *pszPath)

  Find node by path.
- **CPLXMLNode** * **CPLSearchXMLNode** (**CPLXMLNode** *poRoot, const char *pszTarget)

  Search for a node in document.
- const char * **CPLGetXMLValue** (**CPLXMLNode** *poRoot, const char *pszPath, const char *pszDefault)

Fetch element/attribute value.

- **CPLXMLNode** * **CPLCreateXMLNode** (**CPLXMLNode** *poParent, **CPLXMLNodeType** eType, const char *pszText)

  Create an document tree item.
- char * **CPLSerializeXMLTree** (**CPLXMLNode** *psNode)

  Convert tree into string document.
- void **CPLAddXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)

  Add child node to parent.
- int **CPLRemoveXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)

  Remove child node from parent.
- void **CPLAddXMLSibling** (**CPLXMLNode** *psOlderSibling, **CPLXMLNode** *psNewSibling)

  Add new sibling.
- **CPLXMLNode** * **CPLCreateXMLElementAndValue** (**CPLXMLNode** *psParent, const char *pszName, const char *pszValue)

  Create an element and text value.
- **CPLXMLNode** * **CPLCloneXMLTree** (**CPLXMLNode** *psTree)

  Copy tree.
- int **CPLSetXMLValue** (**CPLXMLNode** *psRoot, const char *pszPath, const char *pszValue)

  Set element value by path.
- void **CPLStripXMLNamespace** (**CPLXMLNode** *psRoot, const char *pszNameSpace, int bRecurse)

  Strip indicated namespaces.
- void **CPLCleanXMLElementName** (char *)

  Make string into safe XML token.
- **CPLXMLNode** * **CPLParseXMLFile** (const char *pszFilename)

  Parse XML file into tree.
- int **CPLSerializeXMLTreeToFile** (**CPLXMLNode** *psTree, const char *pszFilename)

  Write document tree to a file.

12.6.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

12.6.2 Typedef Documentation

12.6.2.1 typedef struct CPLXMLNode CPLXMLNode

Document node structure.

This C structure is used to hold a single text fragment representing a component of the document when parsed. It should be allocated with the appropriate CPL function, and freed with **CPLDestroyXMLNode()** (p. 350). The structure contents should not normally be altered by application code, but may be freely examined by application code.

Using the psChild and psNext pointers, a heirarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. 72) structures.

12.6.3 Enumeration Type Documentation

12.6.3.1 enum CPLXMLNodeType

Enumerator:

CXT_Element Node is an element

CXT_Text Node is a raw text value

CXT_Attribute Node is attribute

CXT_Comment Node is an XML comment.

CXT_Literal Node is a special literal

12.6.4 Function Documentation

12.6.4.1 void CPLAddXMLChild (CPLXMLNode * *psParent,* CPLXMLNode * *psChild*)

Add child node to parent.

The passed child is added to the list of children of the indicated parent. Normally the child is added at the end of the parents child list, but attributes (CXT_Attribute) will be inserted after any other attributes but before any other element type. Ownership of the child node is effectively assumed by the parent node. If the child has siblings (it's psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

Parameters

psParent	the node to attach the child to. May not be NULL.
psChild	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References CXT_Attribute, CPLXMLNode::eType, CPLXMLNode::psChild, and CPLXMLNode::psNext.

12.6.4.2 void CPLAddXMLSibling (CPLXMLNode * *psOlderSibling,* CPLXMLNode * *psNewSibling*)

Add new sibling.

The passed psNewSibling is added to the end of siblings of the psOlderSibling node. That is, it is added to the end of the psNext chain. There is no special handling if psNewSibling is an attribute. If this is required, use **CPLAddXMLChild()** (p. 348).

Parameters

psOlderSibling	the node to attach the sibling after.
psNewSibling	the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

12.6.4.3 void CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifiies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of `http://www.w3.org/TR/xml11/` but it doesn't yet do that properly. We only do a rough approximation of that.

Parameters

pszTarget	the string to be adjusted. It is altered in place.

12.6.4.4 CPLXMLNode* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. 72) tree.

Parameters

psTree	the tree to duplicate.

Returns

a copy of the whole tree.

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.5 CPLXMLNode* CPLCreateXMLElementAndValue (CPLXMLNode * *psParent*, const char * *pszName*, const char * *pszValue*)

Create an element and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

psParent	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
pszName	the element name to create.
pszValue	the text to attach to the element. Must not be NULL.

Returns

the pointer to the new element node.

References CXT_Element, and CXT_Text.

12.6.4.6 CPLXMLNode* CPLCreateXMLNode (CPLXMLNode * *poParent*, CPLXMLNodeType *eType*, const char * *pszText*)

Create an document tree item.

Create a single **CPLXMLNode** (p. 72) object with the desired value and type, and attach it as a child of the indicated parent.

Parameters

poParent	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
eType	the type of the newly created node
pszText	the value of the newly created node

Returns

the newly created node, now owned by the caller (or parent node).

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.7 void CPLDestroyXMLNode (CPLXMLNode * psNode)

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. 72) and all its children nodes.

Parameters

psNode	the tree to free.

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.8 CPLXMLNode* CPLGetXMLNode (CPLXMLNode * psRoot, const char * pszPath)

Find node by path.

Searches the document or subdocument indicated by psRoot for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (psRoot). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Seaching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the pszPath is prefixed by "=" then the search will begin with the root node, and it's siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

Parameters

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	the list of element names in the path (dot separated).

Returns

the requested element node, or NULL if not found.

References CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.9 const char* CPLGetXMLValue (CPLXMLNode * psRoot, const char * pszPath, const char * pszDefault)

Fetch element/attribute value.

Searches the document for the element/attribute value associated with the path. The corresponding node is internally found with **CPLGetXMLNode()** (p. 350) (see there for details on path handling). Once found, the value is considered to be the first CXT_Text child of the node.

If the attribute/element search fails, or if the found node has not value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

Parameters

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
pszDefault	the value to return if a corresponding value is not found, may be NULL.

Returns

the requested value or pszDefault if not found.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.10 CPLXMLNode* CPLParseXMLFile (const char * *pszFilename*)

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with **CPLParseXMLString()** (p. 351). Errors in reading the file or parsing the XML will be reported by **CPLError()** (p. 334).

The "large file" API is used, so XML files can come from virtualized files.

Parameters

pszFilename	the file to open.

Returns

NULL on failure, or the document tree on success.

References VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

12.6.4.11 CPLXMLNode* CPLParseXMLString (const char * *pszString*)

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. 72) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via **CPLError()** (p. 334). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. 351) convenience function can be used to parse from a file.

The returned document tree is is owned by the caller and should be freed with **CPLDestroyXMLNode()** (p. 350) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT_Text node.

Parameters

pszString	the document to parse.

Returns

parsed tree or NULL on error.

References CXT_Attribute, CXT_Comment, CXT_Element, CXT_Literal, CXT_Text, and CPLXMLNode::pszValue.

12.6.4.12 int CPLRemoveXMLChild (CPLXMLNode * *psParent*, CPLXMLNode * *psChild*)

Remove child node from parent.

The passed child is removed from the child list of the passed parent, but the child is not destroyed. The child retains ownership of it's own children, but is cleanly removed from the child list of the parent.

Parameters

psParent	the node to the child is attached to.
psChild	the child to remove.

Returns

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

12.6.4.13 CPLXMLNode* CPLSearchXMLNode (CPLXMLNode * *psRoot,* const char * *pszElement*)

Search for a node in document.

Searches the children (and potentially siblings) of the documented passed in for the named element or attribute. To search following siblings as well as children, prefix the pszElement name with an equal sign. This function does an in-order traversal of the document tree. So it will first match against the current node, then it's first child, that childs first child, and so on.

Use **CPLGetXMLNode()** (p. 350) to find a specific child, or along a specific node path.

Parameters

psRoot	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
pszElement	the name of the element or attribute to search for.

Returns

The matching node or NULL on failure.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.6.4.14 char* CPLSerializeXMLTree (CPLXMLNode * *psNode*)

Convert tree into string document.

This function converts a **CPLXMLNode** (p. 72) tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with CPLFree() when no longer needed.

Parameters

psNode	

Returns

the document on success or NULL on failure.

References CPLXMLNode::psNext.

12.6.4.15 int CPLSerializeXMLTreeToFile (CPLXMLNode * *psTree,* const char * *pszFilename*)

Write document tree to a file.

The passed document tree is converted into one big string (with **CPLSerializeXMLTree()** (p. 352)) and then written to the named file. Errors writing the file will be reported by **CPLError()** (p. 334). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters

psTree	the document tree to write.
pszFilename	the name of the file to write to.

Returns

TRUE on success, FALSE otherwise.

References VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

12.6.4.16 int CPLSetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszValue*)

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue("Citation.Id.Description", "DOQ dataset"); CPLSetXMLValue("Citation.Id.-Description.#name", "doq");

Parameters

psRoot	the subdocument to be updated.
pszPath	the dot seperated path to the target element/attribute.
pszValue	the text value to assign.

Returns

TRUE on success.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXML-Node::psNext, and CPLXMLNode::pszValue.

12.6.4.17 void CPLStripXMLNamespace (CPLXMLNode * *psRoot*, const char * *pszNamespace*, int *bRecurse*)

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

Parameters

psRoot	the document to operate on.
pszNamespace	the name space prefix (not including colon), or NULL.
bRecurse	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

12.7 cpl_odbc.h File Reference

```
#include "cpl_port.h"
#include <sql.h>
#include <sqlext.h>
#include <odbcinst.h>
#include "cpl_string.h"
```

Classes

- class **CPLODBCDriverInstaller**
- class **CPLODBCSession**
- class **CPLODBCStatement**

12.7.1 Detailed Description

ODBC Abstraction Layer (C++).

12.8 cpl_port.h File Reference

```
#include "cpl_config.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <time.h>
#include <errno.h>
#include <locale.h>
```

Macros

- #define **CPL_LSBINT16PTR**(x) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))
- #define **CPL_LSBINT32PTR**(x)

12.8.1 Detailed Description

Core portability definitions for CPL.

12.8.2 Macro Definition Documentation

12.8.2.1 #define CPL_LSBINT16PTR(*x*) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))

Return a Int16 from the 2 bytes ordered in LSB order at address x

12.8.2.2 #define CPL_LSBINT32PTR(*x*)

Value:

```
((*(GByte*)(x)) | ((*(GByte*)((x)+1)) << 8) | \
                          ((*(GByte*)((x)+2)) << 16) | ((*(GByte*)((x)+3))
      << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x

12.9 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLRectObj**

Functions

- **CPLQuadTree** ∗ **CPLQuadTreeCreate** (const **CPLRectObj** ∗pGlobalBounds, CPLQuadTreeGetBounds-Func pfnGetBounds)
- void **CPLQuadTreeDestroy** (**CPLQuadTree** ∗hQuadtree)
- void **CPLQuadTreeSetBucketCapacity** (**CPLQuadTree** ∗hQuadtree, int nBucketCapacity)
- int **CPLQuadTreeGetAdvisedMaxDepth** (int nExpectedFeatures)
- void **CPLQuadTreeSetMaxDepth** (**CPLQuadTree** ∗hQuadtree, int nMaxDepth)
- void **CPLQuadTreeInsert** (**CPLQuadTree** ∗hQuadtree, void ∗hFeature)
- void ∗∗ **CPLQuadTreeSearch** (const **CPLQuadTree** ∗hQuadtree, const **CPLRectObj** ∗pAoi, int ∗pnFeature-Count)
- void **CPLQuadTreeForeach** (const **CPLQuadTree** ∗hQuadtree, CPLQuadTreeForeachFunc pfnForeach, void ∗pUserData)

12.9.1 Detailed Description

Quad tree implementation.

A quadtree is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions

12.9.2 Function Documentation

12.9.2.1 CPLQuadTree∗ CPLQuadTreeCreate (const CPLRectObj ∗ *pGlobalBounds,* CPLQuadTreeGetBoundsFunc *pfnGetBounds*)

Create a new quadtree

Parameters

pGlobalBounds	a pointer to the global extent of all the elements that will be inserted
pfnGetBounds	a user provided function to get the bounding box of the inserted elements

Returns

a newly allocated quadtree

12.9.2.2 void CPLQuadTreeDestroy (CPLQuadTree * *hQuadTree*)

Destroy a quadtree

Parameters

hQuadTree	the quad tree to destroy

12.9.2.3 void CPLQuadTreeForeach (const CPLQuadTree * *hQuadTree,* CPLQuadTreeForeachFunc *pfnForeach,* void * *pUserData*)

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters

hQuadTree	the quad tree
pfnForeach	the function called on each element.
pUserData	the user data provided to the function.

12.9.2.4 int CPLQuadTreeGetAdvisedMaxDepth (int *nExpectedFeatures*)

Returns the optimal depth of a quadtree to hold nExpectedFeatures

Parameters

nExpected-Features	the expected maximum number of elements to be inserted

Returns

the optimal depth of a quadtree to hold nExpectedFeatures

12.9.2.5 void CPLQuadTreeInsert (CPLQuadTree * *hQuadTree,* void * *hFeature*)

Insert a feature into a quadtree

Parameters

hQuadTree	the quad tree
hFeature	the feature to insert

12.9.2.6 void** CPLQuadTreeSearch (const CPLQuadTree * *hQuadTree,* const CPLRectObj * *pAoi,* int * *pnFeatureCount*)

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters

hQuadTree	the quad tree
pAoi	the pointer to the area of interest
pnFeatureCount	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

12.9.2.7 void CPLQuadTreeSetBucketCapacity (CPLQuadTree * *hQuadTree,* int *nBucketCapacity*)

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters

hQuadTree	the quad tree
nBucketCapacity	the maximum capactiy of a node of a quadtree

12.9.2.8 void CPLQuadTreeSetMaxDepth (CPLQuadTree * *hQuadTree,* int *nMaxDepth*)

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

Parameters

hQuadTree	the quad tree
nMaxDepth	the maximum depth allowed

12.10 cpl_string.h File Reference

```
#include "cpl_vsi.h"
#include "cpl_error.h"
#include "cpl_conv.h"
#include <string>
```

Classes

- class **CPLString**

  Convenient string class based on std::string.
- class **CPLStringList**

  *String list class designed around our use of C "char** " string lists.*

Functions

- int **CSLCount** (char **papszStrList)
- void **CSLDestroy** (char **papszStrList)
- char ** **CSLDuplicate** (char **papszStrList)
- char ** **CSLMerge** (char **papszOrig, char **papszOverride)

  Merge two lists.

- char ** **CSLTokenizeString2** (const char *pszString, const char *pszDelimeter, int nCSLTFlags)
- char ** **CSLLoad** (const char *pszFname)
- char ** **CSLLoad2** (const char *pszFname, int nMaxLines, int nMaxCols, char **papszOptions)
- int **CSLFindString** (char **, const char *)
- int **CSLPartialFindString** (char **papszHaystack, const char *pszNeedle)
- int **CSLFindName** (char **papszStrList, const char *pszName)
- int **CSLTestBoolean** (const char *pszValue)
- const char * **CPLParseNameValue** (const char *pszNameValue, char **ppszKey)
- char ** **CSLSetNameValue** (char **papszStrList, const char *pszName, const char *pszValue)
- void **CSLSetNameValueSeparator** (char **papszStrList, const char *pszSeparator)
- char * **CPLEscapeString** (const char *pszString, int nLength, int nScheme)
- char * **CPLUnescapeString** (const char *pszString, int *pnLength, int nScheme)
- char * **CPLBinaryToHex** (int nBytes, const GByte *pabyData)
- GByte * **CPLHexToBinary** (const char *pszHex, int *pnBytes)
- CPLValueType **CPLGetValueType** (const char *pszValue)
- size_t **CPLStrlcpy** (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t **CPLStrlcat** (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t **CPLStrnlen** (const char *pszStr, size_t nMaxLen)
- int **CPLEncodingCharSize** (const char *pszEncoding)
- char * **CPLRecode** (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- char * **CPLRecodeFromWChar** (const wchar_t *pwszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- wchar_t * **CPLRecodeToWChar** (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- int **CPLIsUTF8** (const char *pabyData, int nLen)
- char * **CPLForceToASCII** (const char *pabyData, int nLen, char chReplacementChar)
- **CPLString CPLURLGetValue** (const char *pszURL, const char *pszKey)
- **CPLString CPLURLAddKVP** (const char *pszURL, const char *pszKey, const char *pszValue)

12.10.1 Detailed Description

Various convenience functions for working with strings and string lists.

A StringList is just an array of strings with the last pointer being NULL. An empty StringList may be either a NULL pointer, or a pointer to a pointer memory location with a NULL value.

A common convention for StringLists is to use them to store name/value lists. In this case the contents are treated like a dictionary of name/value pairs. The actual data is formatted with each string having the format "<name>-:<value>" (though "=" is also an acceptable separator). A number of the functions in the file operate on name/value style string lists (such as **CSLSetNameValue()** (p. 368), and CSLFetchNameValue()).

To some extent the **CPLStringList** (p. 67) C++ class can be used to abstract managing string lists a bit but still be able to return them from C functions.

12.10.2 Function Documentation

12.10.2.1 char* CPLBinaryToHex (int *nBytes*, const GByte * *pabyData*)

Binary to hexadecimal translation.

Parameters

nBytes	number of bytes of binary data in pabyData.
pabyData	array of data bytes to translate.

Returns

hexadecimal translation, zero terminated. Free with CPLFree().

12.10.2.2 int CPLEncodingCharSize (const char * *pszEncoding*)

Return bytes per character for encoding.

This function returns the size in bytes of the smallest character in this encoding. For fixed width encodings (ASCII, UCS-2, UCS-4) this is straight forward. For encodings like UTF8 and UTF16 which represent some characters as a sequence of atomic character sizes the function still returns the atomic character size (1 for UTF8, 2 for UTF16).

This function will return the correct value for well known encodings with corresponding CPL_ENC_ values. It may not return the correct value for other encodings even if they are supported by the underlying iconv or windows transliteration services. Hopefully it will improve over time.

Parameters

pszEncoding	the name of the encoding.

Returns

the size of a minimal character in bytes or -1 if the size is unknown.

12.10.2.3 char* CPLEscapeString (const char * *pszInput,* int *nLength,* int *nScheme*)

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstitued to it's original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

CPLES_BackslashQuotable(0): This scheme turns a binary string into a form suitable to be placed within double quotes as a string constant. The backslash, quote, '\0' and newline characters are all escaped in the usual C style.

CPLES_XML(1): This scheme converts the '<', '>', '"' and '&' characters into their XML/HTML equivelent (<, >, " and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

CPLES_URL(2): Everything except alphanumerics and the underscore are converted to a percent followed by a two digit hex encoding of the character (leading zero supplied if needed). This is the mechanism used for encoding values to be passed in URLs.

CPLES_SQL(3): All single quotes are replaced with two single quotes. Suitable for use when constructing literal values for SQL commands where the literal will be enclosed in single quotes.

CPLES_CSV(4): If the values contains commas, semicolons, tabs, double quotes, or newlines it placed in double quotes, and double quotes in the value are doubled. Suitable for use when constructing field values for .csv files. Note that **CPLUnescapeString()** (p. 364) currently does not support this format, only **CPLEscapeString()** (p. 359). See cpl_csv.cpp for csv parsing support.

Parameters

pszInput	the string to escape.
nLength	The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.
nScheme	the encoding scheme to use.

Returns

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

12.10.2.4 char* CPLForceToASCII (const char * *pabyData,* int *nLen,* char *chReplacementChar*)

Return a new string that is made only of ASCII characters. If non-ASCII characters are found in the input string, they will be replaced by the provided replacement character.

Parameters

pabyData	input string to test
nLen	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
chReplacement-Char	character which will be used when the input stream contains a non ASCII character. Must be valid ASCII !

Returns

a new string that must be freed with CPLFree().

Since

GDAL 1.7.0

12.10.2.5 CPLValueType CPLGetValueType (const char * *pszValue*)

Detect the type of the value contained in a string, whether it is a real, an integer or a string Leading and trailing spaces are skipped in the analysis.

Note: in the context of this function, integer must be understood in a broad sense. It does not mean that the value can fit into a 32 bit integer for example. It might be larger.

Parameters

pszValue	the string to analyze

Returns

returns the type of the value contained in the string.

12.10.2.6 GByte* CPLHexToBinary (const char * *pszHex,* int * *pnBytes*)

Hexadecimal to binary translation

Parameters

pszHex	the input hex encoded string.
pnBytes	the returned count of decoded bytes placed here.

Returns

returns binary buffer of data - free with CPLFree().

12.10.2.7 int CPLIsUTF8 (const char * pabyData, int nLen)

Test if a string is encoded as UTF-8.

Parameters

pabyData	input string to test
nLen	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.

Returns

TRUE if the string is encoded as UTF-8. FALSE otherwise

Since

GDAL 1.7.0

12.10.2.8 const char* CPLParseNameValue (const char * pszNameValue, char ** ppszKey)

Parse NAME=VALUE string into name and value components.

Note that if ppszKey is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimeter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. 361).

Parameters

pszNameValue	string in "NAME=VALUE" format.
ppszKey	optional pointer though which to return the name portion.

Returns

the value portion (pointing into original string).

12.10.2.9 char* CPLRecode (const char * pszSource, const char * pszSrcEncoding, const char * pszDstEncoding)

Convert a string from a source encoding to a destination encoding.

The only guaranteed supported encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8859_1. Currently, the following conversions are supported :

- CPL_ENC_ASCII -> CPL_ENC_UTF8 or CPL_ENC_ISO8859_1 (no conversion in fact)
- CPL_ENC_ISO8859_1 -> CPL_ENC_UTF8
- CPL_ENC_UTF8 -> CPL_ENC_ISO8859_1

If an error occurs an error may, or may not be posted with **CPLError()** (p. 334).

Parameters

pszSource	a NULL terminated string.
pszSrcEncoding	the source encoding.
pszDstEncoding	the destination encoding.

Returns

a NULL terminated string which should be freed with CPLFree().

Since

GDAL 1.6.0

12.10.2.10 char* CPLRecodeFromWChar (const wchar_t * *pwszSource,* const char * *pszSrcEncoding,* const char * *pszDstEncoding*)

Convert wchar_t string to UTF-8.

Convert a wchar_t string into a multibyte utf-8 string. The only guaranteed supported source encoding is CPL_ENC_UCS2, and the only guaranteed supported destination encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8859_1. In some cases (ie. using iconv()) other encodings may also be supported.

Note that the wchar_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. 334).

Parameters

pwszSource	the source wchar_t string, terminated with a 0 wchar_t.
pszSrcEncoding	the source encoding, typically CPL_ENC_UCS2.
pszDstEncoding	the destination encoding, typically CPL_ENC_UTF8.

Returns

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

Since

GDAL 1.6.0

12.10.2.11 wchar_t* CPLRecodeToWChar (const char * *pszSource,* const char * *pszSrcEncoding,* const char * *pszDstEncoding*)

Convert UTF-8 string to a wchar_t string.

Convert a 8bit, multi-byte per character input string into a wide character (wchar_t) string. The only guaranteed supported source encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8869_1 (LATIN1). The only guaranteed supported destination encoding is CPL_ENC_UCS2. Other source and destination encodings may be supported depending on the underlying implementation.

Note that the wchar_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. 334).

Parameters

pszSource	input multi-byte character string.
pszSrcEncoding	source encoding, typically CPL_ENC_UTF8.
pszDstEncoding	destination encoding, typically CPL_ENC_UCS2.

Returns

the zero terminated wchar_t string (to be freed with CPLFree()) or NULL on error.

Since

GDAL 1.6.0

12.10.2.12 size_t CPLStrlcat (char * *pszDest*, const char * *pszSrc*, size_t *nDestSize*)

Appends a source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1 and that there is at least one byte free in pszDest, that is to say strlen(pszDest_before) $<$ nDestSize)

This function is designed to be a safer, more consistent, and less error prone replacement for strncat. Its contract is identical to libbsd's strlcat.

Truncation can be detected by testing if the return value of CPLStrlcat is greater or equal to nDestSize.

```
char szDest[5];
CPLStrlcpy(szDest, "ab", sizeof(szDest));
if (CPLStrlcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occured !\n");
```

Parameters

pszDest	destination buffer. Must be NUL terminated before running CPLStrlcat
pszSrc	source string. Must be NUL terminated
nDestSize	size of destination buffer (including space for the NUL terminator character)

Returns

the thoretical length of the destination string after concatenation (=strlen(pszDest_before) + strlen(pszSrc)). If strlen(pszDest_before) $>=$ nDestSize, then it returns nDestSize + strlen(pszSrc)

Since

GDAL 1.7.0

12.10.2.13 size_t CPLStrlcpy (char * *pszDest*, const char * *pszSrc*, size_t *nDestSize*)

Copy source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1).

This function is designed to be a safer, more consistent, and less error prone replacement for strncpy. Its contract is identical to libbsd's strlcpy.

Truncation can be detected by testing if the return value of CPLStrlcpy is greater or equal to nDestSize.

```
char szDest[5];
if (CPLStrlcpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occured !\n");
```

Parameters

pszDest	destination buffer
pszSrc	source string. Must be NUL terminated
nDestSize	size of destination buffer (including space for the NUL terminator character)

Returns

the length of the source string (=strlen(pszSrc))

Since

GDAL 1.7.0

12.10.2.14 size_t CPLStrnlen (const char * *pszStr,* size_t *nMaxLen*)

Returns the length of a NUL terminated string by reading at most the specified number of bytes.

The **CPLStrnlen()** (p. 364) function returns MIN(strlen(pszStr), nMaxLen). Only the first nMaxLen bytes of the string will be read. Usefull to test if a string contains at least nMaxLen characters without reading the full string up to the NUL terminating character.

Parameters

pszStr	a NUL terminated string
nMaxLen	maximum number of bytes to read in pszStr

Returns

strlen(pszStr) if the length is lesser than nMaxLen, otherwise nMaxLen if the NUL character has not been found in the first nMaxLen bytes.

Since

GDAL 1.7.0

12.10.2.15 char* CPLUnescapeString (const char * *pszInput,* int * *pnLength,* int *nScheme*)

Unescape a string.

This function does the opposite of **CPLEscapeString()** (p. 359). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to it's original form.

Parameters

pszInput	the input string. This is a zero terminated string.
pnLength	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
nScheme	the escaped scheme to undo (see **CPLEscapeString()** (p. 359) for a list).

Returns

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

12.10.2.16 CPLString CPLURLAddKVP (const char * *pszURL,* const char * *pszKey,* const char * *pszValue*)

Return a new URL with a new key=value pair.

Parameters

pszURL	the URL.
pszKey	the key to find.
pszValue	the value of the key (may be NULL to unset an existing KVP).

Returns

the modified URL.

Since

GDAL 1.9.0

References CPLURLAddKVP(), and CPLString::ifind().

Referenced by CPLURLAddKVP().

12.10.2.17 CPLString CPLURLGetValue (const char * *pszURL*, const char * *pszKey*)

Return the value matching a key from a key=value pair in a URL.

Parameters

pszURL	the URL.
pszKey	the key to find.

Returns

the value of empty string if not found.

Since

GDAL 1.9.0

References CPLURLGetValue(), and CPLString::ifind().

Referenced by CPLURLGetValue().

12.10.2.18 int CSLCount (char ** *papszStrList*)

Return number of items in a string list.

Returns the number of items in a string list, not counting the terminating NULL. Passing in NULL is safe, and will result in a count of zero.

Lists are counted by iterating through them so long lists will take more time than short lists. Care should be taken to avoid using **CSLCount()** (p. 365) as an end condition for loops as it will result in O(n^2) behavior.

Parameters

papszStrList	the string list to count.

Returns

the number of entries.

12.10.2.19 void CSLDestroy (char ** *papszStrList*)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters

papszStrList	the list to free.

12.10.2.20 char** CSLDuplicate (char ** papszStrList)

Clone a string list.

Efficiently allocates a copy of a string list. The returned list is owned by the caller and should be freed with **CSLDestroy()** (p. 365).

Parameters

papszStrList	the input string list.

Returns

newly allocated copy.

12.10.2.21 int CSLFindName (char ** papszStrList, const char * pszName)

Find StringList entry with given key name.

Parameters

papszStrList	the string list to search.
pszName	the key value to look for (case insensitive).

Returns

-1 on failure or the list index of the first occurance matching the given key.

12.10.2.22 int CSLFindString (char ** papszList, const char * pszTarget)

Find a string within a string list.

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

Parameters

papszList	the string list to be searched.
pszTarget	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

12.10.2.23 char** CSLLoad (const char * pszFname)

Load a text file into a string list.

The VSI*L API is used, so **VSIFOpenL()** (p. 373) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. 327)).

If reading the file fails a **CPLError()** (p. 334) will be issued and NULL returned.

Parameters

pszFname	the name of the file to read.

Returns

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. 365)

12.10.2.24 char** CSLLoad2 (const char * *pszFname*, int *nMaxLines*, int *nMaxCols*, char ** *papszOptions*)

Load a text file into a string list.

The VSI*L API is used, so **VSIFOpenL()** (p. 373) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. 327)).

If reading the file fails a **CPLError()** (p. 334) will be issued and NULL returned.

Parameters

pszFname	the name of the file to read.
nMaxLines	maximum number of lines to read before stopping, or -1 for no limit.
nMaxCols	maximum number of characters in a line before stopping, or -1 for no limit.
papszOptions	NULL-terminated array of options. Unused for now.

Returns

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. 365)

Since

GDAL 1.7.0

References VSIFCloseL(), VSIFEofL(), and VSIFOpenL().

12.10.2.25 char** CSLMerge (char ** *papszOrig*, char ** *papszOverride*)

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (papszOverride) list take precidence.

Parameters

papszOrig	the original list, being modified.
papszOverride	the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns

updated list.

12.10.2.26 int CSLPartialFindString (char ** *papszHaystack*, const char * *pszNeedle*)

Find a substring within a string list.

Returns the index of the entry in the string list that contains the target string as a substring. The search is case sensitive (unlike **CSLFindString()** (p. 366)).

Parameters

papszHaystack	the string list to be searched.
pszNeedle	the substring to be searched for.

Returns

the index of the string within the list or -1 on failure.

12.10.2.27 char** CSLSetNameValue (char ** papszList, const char * pszName, const char * pszValue)

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

Parameters

papszList	the original list, the modified version is returned.
pszName	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
pszValue	the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

Returns

modified stringlist.

12.10.2.28 void CSLSetNameValueSeparator (char ** papszList, const char * pszSeparator)

Replace the default separator (":" or "=") with the passed separator in the given name/value list.

Note that if a separator other than ":" or "=" is used, the resulting list will not be manipulatable by the CSL name/value functions any more.

The **CPLParseNameValue()** (p. 361) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desireable to include some white space in the new separator. eg. ": " or " = ".

Parameters

papszList	the list to update. Component strings may be freed but the list array will remain at the same location.
pszSeparator	the new separator string to insert.

12.10.2.29 int CSLTestBoolean (const char * pszValue)

Test what boolean value contained in the string.

If pszValue is "NO", "FALSE", "OFF" or "0" will be returned FALSE. Otherwise, TRUE will be returned.

Parameters

pszValue	the string should be tested.

Returns

TRUE or FALSE.

12.10.2.30 char** CSLTokenizeString2 (const char * *pszString,* const char * *pszDelimiters,* int *nCSLTFlags*)

Tokenize a string.

This function will split a string into tokens based on specified' delimeter(s) with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. 365) when no longer needed.

The available parsing options are:

- CSLT_ALLOWEMPTYTOKENS: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- CSLT_STRIPLEADSPACES: strip leading space characters from the token (as reported by isspace());
- CSLT_STRIPENDSPACES: strip ending space characters from the token (as reported by isspace());
- CSLT_HONOURSTRINGS: double quotes can be used to hold values that should not be broken into multiple tokens;
- CSLT_PRESERVEQUOTES: string quotes are carried into the tokens when this is set, otherwise they are removed;
- CSLT_PRESERVEESCAPES: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

Example:

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens;
int i;

papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
                        CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
    printf( "arg %d: '%s'", papszTokens[i] );
CSLDestroy( papszTokens );
```

Parameters

pszString	the string to be split into tokens.
pszDelimiters	one or more characters to be used as token delimeters.
nCSLTFlags	an ORing of one or more of the CSLT_ flag values.

Returns

a string list of tokens owned by the caller.

References CPLStringList::AddString(), CPLStringList::Assign(), CPLStringList::Count(), and CPLStringList::StealList().

12.11 cpl_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

Functions

- VSILFILE * **VSIFOpenL** (const char *, const char *)

  Open file.
- int **VSIFCloseL** (VSILFILE *)

  Close file.
- int **VSIFSeekL** (VSILFILE *, vsi_l_offset, int)

  Seek to requested offset.
- vsi_l_offset **VSIFTellL** (VSILFILE *)

  Tell current file offset.
- size_t **VSIFReadL** (void *, size_t, size_t, VSILFILE *)

  Read bytes from file.
- int **VSIFReadMultiRangeL** (int nRanges, void **ppData, const vsi_l_offset *panOffsets, const size_t *panSizes, VSILFILE *)

  Read several ranges of bytes from file.
- size_t **VSIFWriteL** (const void *, size_t, size_t, VSILFILE *)

  Write bytes to file.
- int **VSIFEofL** (VSILFILE *)

  Test for end of file.
- int **VSIFTruncateL** (VSILFILE *, vsi_l_offset)

  Truncate/expand the file to the specified size.
- int **VSIFFlushL** (VSILFILE *)

  Flush pending writes to disk.
- int **VSIFPrintfL** (VSILFILE *, const char *,...)

  Formatted write to file.
- int **VSIStatL** (const char *, VSIStatBufL *)

  Get filesystem object info.
- int **VSIStatExL** (const char *pszFilename, VSIStatBufL *psStatBuf, int nFlags)

  Get filesystem object info.
- int **VSIIsCaseSensitiveFS** (const char *pszFilename)

  Returns if the filenames of the filesystem are case sensitive.
- void * **VSIMalloc2** (size_t nSize1, size_t nSize2)
- void * **VSIMalloc3** (size_t nSize1, size_t nSize2, size_t nSize3)
- char ** **VSIReadDir** (const char *)

  Read names in a directory.
- int **VSIMkdir** (const char *pathname, long mode)

  Create a directory.
- int **VSIRmdir** (const char *pathname)

  Delete a directory.
- int **VSIUnlink** (const char *pathname)

  Delete a file.
- int **VSIRename** (const char *oldpath, const char *newpath)

  Rename a file.
- void **VSIInstallMemFileHandler** (void)

Install "memory" file system handler.

- void **VSIInstallSubFileHandler** (void)
- void **VSIInstallCurlFileHandler** (void)

  Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)
- void **VSIInstallGZipFileHandler** (void)

  Install GZip file system handler.
- void **VSIInstallZipFileHandler** (void)

  Install ZIP file system handler.
- void **VSIInstallStdinHandler** (void)

  Install /vsistdin/ file system handler.
- void **VSIInstallStdoutHandler** (void)

  Install /vsistdout/ file system handler.
- void **VSIInstallSparseFileHandler** (void)
- void **VSIInstallTarFileHandler** (void)

  Install /vsitar/ file system handler.
- VSILFILE * **VSIFileFromMemBuffer** (const char *pszFilename, GByte *pabyData, vsi_l_offset nDataLength, int bTakeOwnership)

  Create memory "file" from a buffer.
- GByte * **VSIGetMemFileBuffer** (const char *pszFilename, vsi_l_offset *pnDataLength, int bUnlinkAndSeize)

  Fetch buffer underlying memory file.

12.11.1 Detailed Description

Standard C Covers

The VSI functions are intended to be hookable aliases for Standard C I/O, memory allocation and other system functions. They are intended to allow virtualization of disk I/O so that non file data sources can be made to appear as files, and so that additional error trapping and reporting can be interested. The memory access API is aliased so that special application memory management services can be used.

Is is intended that each of these functions retains exactly the same calling pattern as the original Standard C functions they relate to. This means we don't have to provide custom documentation, and also means that the default implementation is very simple.

12.11.2 Function Documentation

12.11.2.1 int VSIFCloseL (VSILFILE * *fp*)

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

0 on success or -1 on failure.

References VSIFCloseL().

Referenced by CPLCloseShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIFCloseL().

12.11.2.2 int VSIFEofL (VSILFILE * *fp*)

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occured during the previous read operation. The end-of-file flag is cleared by a successfull **VSIFSeekL()** (p. 375) call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

TRUE if at EOF else FALSE.

References VSIFEofL().

Referenced by CSLLoad2(), and VSIFEofL().

12.11.2.3 int VSIFFlushL (VSILFILE * *fp*)

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

0 on success or -1 on error.

References VSIFFlushL().

Referenced by VSIFFlushL().

12.11.2.4 VSILFILE* VSIFileFromMemBuffer (const char * *pszFilename,* GByte * *pabyData,* vsi_l_offset *nDataLength,* int *bTakeOwnership*)

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If bTakeOwnership is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the pabyData contents.

Parameters

pszFilename	the filename to be created.
pabyData	the data buffer for the file.
nDataLength	the length of buffer in bytes.
bTakeOwnership	TRUE to transfer "ownership" of buffer or FALSE.

Returns

open file handle on created file (see **VSIFOpenL()** (p. 373)).

References VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer().

12.11.2.5 VSILFILE* VSIFOpenL (const char * *pszFilename,* const char * *pszAccess*)

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the pszAccess string.

Note that the "VSILFILE *" returned since GDAL 1.8.0 by this function is *NOT* a standard C library FILE *, and cannot be used with any functions other than the "VSI*L" family of functions. They aren't "real" FILE objects.

On windows it is possible to define the configuration option GDAL_FILE_IS_UTF8 to have pszFilename treated as being in the local encoding instead of UTF-8, retoring the pre-1.8.0 behavior of **VSIFOpenL()** (p. 373).

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

pszFilename	the file to open. UTF-8 encoded.
pszAccess	access requested (ie. "r", "r+", "w".

Returns

NULL on failure, or the file handle.

References VSIFOpenL().

Referenced by CPLOpenShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIFOpenL().

12.11.2.6 int VSIFPrintfL (VSILFILE * *fp,* const char * *pszFormat, ...*)

Formatted write to file.

Provides fprintf() style formatted output to a VSI*L file. This formats an internal buffer which is written using **VSIFWriteL()** (p. 376).

Analog of the POSIX fprintf() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).
pszFormat	the printf style format string.

Returns

the number of bytes written or -1 on an error.

References VSIFPrintfL(), and VSIFWriteL().

Referenced by VSIFPrintfL().

12.11.2.7 size_t VSIFReadL (void * *pBuffer,* size_t *nSize,* size_t *nCount,* VSILFILE * *fp*)

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount * nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

number of objects successfully read.

References VSIFReadL().

Referenced by CPLParseXMLFile(), CPLReadLine2L(), and VSIFReadL().

12.11.2.8 int VSIFReadMultiRangeL (int *nRanges,* void ** *ppData,* const vsi_l_offset * *panOffsets,* const size_t * *panSizes,* VSILFILE * *fp*)

Read several ranges of bytes from file.

Reads nRanges objects of panSizes[i] bytes from the indicated file at the offset panOffsets[i] into the buffer ppData[i].

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

Parameters

nRanges	number of ranges to read.
ppData	array of nRanges buffer into which the data should be read (ppData[i] must be at list panSizes[i] bytes).
panOffsets	array of nRanges offsets at which the data should be read.
panSizes	array of nRanges sizes of objects to read (in bytes).
fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

References VSIFReadMultiRangeL().

Referenced by VSIFReadMultiRangeL().

12.11.2.9 int VSIFSeekL (VSILFILE * *fp*, vsi_l_offset *nOffset*, int *nWhence*)

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).
nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

References VSIFSeekL().

Referenced by CPLParseXMLFile(), CPLReadLine2L(), and VSIFSeekL().

12.11.2.10 vsi_l_offset VSIFTellL (VSILFILE * *fp*)

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

file offset in bytes.

References VSIFTellL().

Referenced by CPLParseXMLFile(), CPLReadLine2L(), and VSIFTellL().

12.11.2.11 int VSIFTruncateL (VSILFILE * *fp*, vsi_l_offset *nNewSize*)

Truncate/expand the file to the specified size.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. 373).
nNewSize	new size in bytes.

Returns

0 on success

Since

GDAL 1.9.0

References VSIFTruncateL().

Referenced by VSIFTruncateL().

12.11.2.12 size_t VSIFWriteL (const void * *pBuffer*, size_t *nSize*, size_t *nCount*, VSILFILE * *fp*)

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters

pBuffer	the buffer from which the data should be written (at least nCount * nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with **VSIFOpenL()** (p. 373).

Returns

number of objects successfully written.

References VSIFWriteL().

Referenced by CPLSerializeXMLTreeToFile(), VSIFPrintfL(), and VSIFWriteL().

12.11.2.13 GByte* VSIGetMemFileBuffer (const char * *pszFilename*, vsi_l_offset * *pnDataLength*, int *bUnlinkAndSeize*)

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If bUnlinkAndSeize is TRUE the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existance.

Parameters

pszFilename	the name of the file to grab the buffer of.
pnDataLength	(file) length returned in this variable.
bUnlinkAndSeize	TRUE to remove the file, or FALSE to leave unaltered.

Returns

pointer to memory buffer or NULL on failure.

References VSIGetMemFileBuffer().

Referenced by VSIGetMemFileBuffer().

12.11.2.14 void VSIInstallCurlFileHandler (void)

Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)

A special file handler is installed that allows reading on-the-fly of files available through HTTP/FTP web protocols, without downloading the entire file.

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

Partial downloads (requires the HTTP server to support random reading) are done with a 16 KB granularity by default. If the driver detects sequential reading it will progressively increase the chunk size up to 2 MB to improve download performance.

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

VSIStatL() (p. 384) will return the size in st_size member and file nature- file or directory - in st_mode member (the later only reliable with FTP resources for now).

VSIReadDir() (p. 382) should be able to parse the HTML directory listing returned by the most popular web servers, such as Apache or Microsoft IIS.

This special file handler can be combined with other virtual filesystems handlers, such as /vsizip. For example, /vsizip//vsicurl/path/to/remote/file.zip/path/inside/zip

Since

GDAL 1.8.0

References VSIInstallCurlFileHandler().

Referenced by VSIInstallCurlFileHandler().

12.11.2.15 void VSIInstallGZipFileHandler (void)

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly and writing in GZip (.gz) files.

All portions of the file system underneath the base path "/vsigzip/" will be handled by this driver.

Additional documentation is to be found at `http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip`

Since

GDAL 1.6.0

References VSIInstallGZipFileHandler().

Referenced by VSIInstallGZipFileHandler().

12.11.2.16 void VSIInstallMemFileHandler (void)

Install "memory" file system handler.

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

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

At this time the memory handler does not properly handle directory semantics for the memory portion of the filesystem. The **VSIReadDir()** (p. 382) function is not supported though this will be corrected in the future.

Calling this function repeatedly should do no harm, though it is not necessary. It is already called the first time a virtualizable file access function (ie. **VSIFOpenL()** (p. 373), VSIMkDir(), etc) is called.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
                            vsi_l_offset *pnOutDataLength )
{
    // create memory file system object from buffer.
    VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                      nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                             NULL, NULL );

    // close source file, and "unlink" it.
    GDALClose( hDS );
    VSIUnlink( "/vsimem/work.dat" );

    // seize the buffer associated with the output file.

    return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}
```

References VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

12.11.2.17 void VSIInstallSparseFileHandler (void)

Install /vsisparse/ virtual file handler.

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

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

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

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

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

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

Hopefully the values and semantics are fairly obvious.

This driver is installed by default.

References VSIInstallSparseFileHandler().

Referenced by VSIInstallSparseFileHandler().

12.11.2.18 void VSIInstallStdinHandler (void)

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input steam.

The file operations available are of course limited to Read() and forward Seek() (full seek in the first MB of a file).

Since

GDAL 1.8.0

References VSIInstallStdinHandler().

Referenced by VSIInstallStdinHandler().

12.11.2.19 void VSIInstallStdoutHandler (void)

Install /vsistdout/ file system handler.

A special file handler is installed that allows writing to the standard output stream.

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

Since

GDAL 1.8.0

References VSIInstallStdoutHandler().

Referenced by VSIInstallStdoutHandler().

12.11.2.20 void VSIInstallSubFileHandler (void)

Install /vsisubfile/ virtual file handler.

This virtual file system handler allows access to subregions of files, treating them as a file on their own to the virtual file system functions (**VSIFOpenL()** (p. 373), etc).

A special form of the filename is used to indicate a subportion of another file:

/vsisubfile/<offset>[_<size>],<filename>

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

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

Unlike the /vsimem/ or conventional file system handlers, there is no meaningful support for filesystem operations for creating new files, traversing directories, and deleting files within the /vsisubfile/ area. Only the **VSIStatL()** (p. 384), **VSIFOpenL()** (p. 373) and operations based on the file handle returned by **VSIFOpenL()** (p. 373) operate properly.

References VSIInstallSubFileHandler().

Referenced by VSIInstallSubFileHandler().

12.11.2.21 void VSIInstallTarFileHandler (void)

Install /vsitar/ file system handler.

A special file handler is installed that allows reading on-the-fly in TAR (regular .tar, or compressed .tar.gz/.tgz) archives.

All portions of the file system underneath the base path "/vsitar/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsitar/path/to/the/file.tar/path/inside/the/tar/file were path/to/the/file.tar is relative or absolute and path/inside/the/tar/file is the relative path to the file inside the archive.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsitar//home/gdal/... For example gdalinfo /vsitar/myarchive.tar/subdir1/file1.tif

Syntaxic sugar : if the tar archive contains only one file located at its root, just mentionning "/vsitar/path/to/the/file.tar" will work

VSIStatL() (p. 384) will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through **VSIReadDir()** (p. 382).

Since

GDAL 1.8.0

References VSIInstallTarFileHandler().

Referenced by VSIInstallTarFileHandler().

12.11.2.22 void VSIInstallZipFileHandler (void)

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives.

All portions of the file system underneath the base path "/vsizip/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsizip/path/to/the/file.zip/path/inside/the/zip/file were path/to/the/file.zip is relative or absolute and path/inside/the/zip/file is the relative path to the file inside the archive.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsizip//home/gdal/... For example gdalinfo /vsizip/myarchive.zip/subdir1/file1.tif

Syntaxic sugar : if the .zip file contains only one file located at its root, just mentionning "/vsizip/path/to/the/file.zip" will work

VSIStatL() (p. 384) will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through **VSIReadDir()** (p. 382).

Since GDAL 1.8.0, write capabilities are available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file. Read and write operations cannot be interleaved : the new zip must be closed before being re-opened for read.

Additional documentation is to be found at `http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip`

Since

GDAL 1.6.0

References VSIInstallZipFileHandler().

Referenced by VSIInstallZipFileHandler().

12.11.2.23 int VSIIsCaseSensitiveFS (const char * *pszFilename*)

Returns if the filenames of the filesystem are case sensitive.

This method retrieves to which filesystem belongs the passed filename and return TRUE if the filenames of that filesystem are case sensitive.

Currently, this will return FALSE only for Windows real filenames. Other VSI virtual filesystems are case sensitive.

This methods avoid ugly #ifndef WIN32 / #endif code, that is wrong when dealing with virtual filenames.

Parameters

pszFilename	the path of the filesystem object to be tested. UTF-8 encoded.

Returns

TRUE if the filenames of the filesystem are case sensitive.

Since

GDAL 1.8.0

References VSIIsCaseSensitiveFS().

Referenced by CPLFormCIFilename(), and VSIIsCaseSensitiveFS().

12.11.2.24 void* VSIMalloc2 (size_t *nSize1*, size_t *nSize2*)

VSIMalloc2 allocates (nSize1 * nSize2) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with **CPLError()** (p. 334). If nSize1 == 0 || nSize2 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc2().

Referenced by OGRPolygon::importFromWkb(), and VSIMalloc2().

12.11.2.25 void* VSIMalloc3 (size_t *nSize1*, size_t *nSize2*, size_t *nSize3*)

VSIMalloc3 allocates (nSize1 * nSize2 * nSize3) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with **CPLError()** (p. 334). If nSize1 == 0 || nSize2 == 0 || nSize3 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc3().

Referenced by VSIMalloc3().

12.11.2.26 int VSIMkdir (const char * *pszPathname,* long *mode*)

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters

pszPathname	the path to the directory to create. UTF-8 encoded.
mode	the permissions mode.

Returns

0 on success or -1 on an error.

References VSIMkdir().

Referenced by VSIMkdir().

12.11.2.27 char ** VSIReadDir (const char * *pszPath*)

Read names in a directory.

This function abstracts access to directory contains. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. 365) when no longer needed.

Note that no error is issued via **CPLError()** (p. 334) if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

Parameters

pszPath	the relative, or absolute path of a directory to read. UTF-8 encoded.

Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

References VSIReadDir().

Referenced by VSIReadDir().

12.11.2.28 int VSIRename (const char * *oldpath,* const char * *newpath*)

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

Parameters

oldpath	the name of the file to be renamed. UTF-8 encoded.
newpath	the name the file should be given. UTF-8 encoded.

Returns

0 on success or -1 on an error.

References VSIRename().

Referenced by VSIRename().

12.11.2.29 int VSIRmdir (const char * *pszDirname*)

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

Parameters

pszDirname	the path of the directory to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

References VSIRmdir().

Referenced by CPLUnlinkTree(), and VSIRmdir().

12.11.2.30 int VSIStatExL (const char * *pszFilename,* VSIStatBufL * *psStatBuf,* int *nFlags*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function, with an extra parameter to specify which information is needed, which offers a potential for speed optimizations on specialized and potentially slow virtual filesystem objects (/vsigzip/, /vsicurl/)

Parameters

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.
nFlags	0 to get all information, or VSI_STAT_EXISTS_FLAG, VSI_STAT_NATURE_FLAG or VSI_STAT_SIZE_FLAG, or a combination of those to get partial info.

Returns

0 on success or -1 on an error.

Since

GDAL 1.8.0

References VSIStatExL().

Referenced by CPLFormCIFilename(), VSIStatExL(), and VSIStatL().

12.11.2.31 int VSIStatL (const char * *pszFilename,* VSIStatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

Parameters

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.

Returns

0 on success or -1 on an error.

References VSIStatExL(), and VSIStatL().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCheckForFile(), and VSIStatL().

12.11.2.32 int VSIUnlink (const char * *pszFilename*)

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters

pszFilename	the path of the file to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

References VSIUnlink().

Referenced by CPLUnlinkTree(), and VSIUnlink().

12.12 ogr_api.h File Reference

```
#include "ogr_core.h"
```

Functions

- OGRErr **OGR_G_CreateFromWkb** (unsigned char *, OGRSpatialReferenceH, OGRGeometryH *, int)

  Create a geometry object of the appropriate type from it's well known binary representation.
- OGRErr **OGR_G_CreateFromWkt** (char **, OGRSpatialReferenceH, OGRGeometryH *)

  Create a geometry object of the appropriate type from it's well known text representation.
- void **OGR_G_DestroyGeometry** (OGRGeometryH)

  Destroy geometry object.
- OGRGeometryH **OGR_G_CreateGeometry** (**OGRwkbGeometryType**)

  Create an empty geometry of desired type.
- OGRGeometryH **OGR_G_ApproximateArcAngles** (double dfCenterX, double dfCenterY, double dfZ, double dfPrimaryRadius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees)
- OGRGeometryH **OGR_G_ForceToPolygon** (OGRGeometryH)

  Convert to polygon.
- OGRGeometryH **OGR_G_ForceToMultiPolygon** (OGRGeometryH)

  Convert to multipolygon.
- OGRGeometryH **OGR_G_ForceToMultiPoint** (OGRGeometryH)

  Convert to multipoint.
- OGRGeometryH **OGR_G_ForceToMultiLineString** (OGRGeometryH)

  Convert to multilinestring.
- int **OGR_G_GetDimension** (OGRGeometryH)

  Get the dimension of this geometry.
- int **OGR_G_GetCoordinateDimension** (OGRGeometryH)

  Get the dimension of the coordinates in this geometry.
- void **OGR_G_SetCoordinateDimension** (OGRGeometryH, int)

  Set the coordinate dimension.
- OGRGeometryH **OGR_G_Clone** (OGRGeometryH)

  Make a copy of this object.
- void **OGR_G_GetEnvelope** (OGRGeometryH, **OGREnvelope** *)

  Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.
- void **OGR_G_GetEnvelope3D** (OGRGeometryH, **OGREnvelope3D** *)

  Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.
- OGRErr **OGR_G_ImportFromWkb** (OGRGeometryH, unsigned char *, int)

  Assign geometry from well known binary data.
- OGRErr **OGR_G_ExportToWkb** (OGRGeometryH, OGRwkbByteOrder, unsigned char *)

  Convert a geometry into well known binary format.
- int **OGR_G_WkbSize** (OGRGeometryH hGeom)

  Returns size of related binary representation.
- OGRErr **OGR_G_ImportFromWkt** (OGRGeometryH, char **)

  Assign geometry from well known text data.
- OGRErr **OGR_G_ExportToWkt** (OGRGeometryH, char **)

  Convert a geometry into well known text format.
- **OGRwkbGeometryType OGR_G_GetGeometryType** (OGRGeometryH)

  Fetch geometry type.
- const char * **OGR_G_GetGeometryName** (OGRGeometryH)

  Fetch WKT name for geometry type.
- void **OGR_G_DumpReadable** (OGRGeometryH, FILE *, const char *)

  Dump geometry in well known text format to indicated output file.
- void **OGR_G_FlattenTo2D** (OGRGeometryH)

  Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

- void **OGR_G_CloseRings** (OGRGeometryH)

  Force rings to be closed.
- OGRGeometryH **OGR_G_CreateFromGML** (const char *)

  Create geometry from GML.
- char * **OGR_G_ExportToGML** (OGRGeometryH)

  Convert a geometry into GML format.
- char * **OGR_G_ExportToGMLEx** (OGRGeometryH, char **papszOptions)

  Convert a geometry into GML format.
- char * **OGR_G_ExportToKML** (OGRGeometryH, const char *pszAltitudeMode)

  Convert a geometry into KML format.
- char * **OGR_G_ExportToJson** (OGRGeometryH)

  Convert a geometry into GeoJSON format.
- char * **OGR_G_ExportToJsonEx** (OGRGeometryH, char **papszOptions)

  Convert a geometry into GeoJSON format.
- void **OGR_G_AssignSpatialReference** (OGRGeometryH, OGRSpatialReferenceH)

  Assign spatial reference to this object.
- OGRSpatialReferenceH **OGR_G_GetSpatialReference** (OGRGeometryH)

  Returns spatial reference system for geometry.
- OGRErr **OGR_G_Transform** (OGRGeometryH, OGRCoordinateTransformationH)

  Apply arbitrary coordinate transformation to geometry.
- OGRErr **OGR_G_TransformTo** (OGRGeometryH, OGRSpatialReferenceH)

  Transform geometry to new spatial reference system.
- OGRGeometryH **OGR_G_Simplify** (OGRGeometryH hThis, double tolerance)

  Compute a simplified geometry.
- OGRGeometryH **OGR_G_SimplifyPreserveTopology** (OGRGeometryH hThis, double tolerance)

  Simplify the geometry while preserving topology.
- void **OGR_G_Segmentize** (OGRGeometryH hGeom, double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.
- int **OGR_G_Intersects** (OGRGeometryH, OGRGeometryH)

  Do these features intersect?
- int **OGR_G_Equals** (OGRGeometryH, OGRGeometryH)

  Returns TRUE if two geometries are equivalent.
- int **OGR_G_Disjoint** (OGRGeometryH, OGRGeometryH)

  Test for disjointness.
- int **OGR_G_Touches** (OGRGeometryH, OGRGeometryH)

  Test for touching.
- int **OGR_G_Crosses** (OGRGeometryH, OGRGeometryH)

  Test for crossing.
- int **OGR_G_Within** (OGRGeometryH, OGRGeometryH)

  Test for containment.
- int **OGR_G_Contains** (OGRGeometryH, OGRGeometryH)

  Test for containment.
- int **OGR_G_Overlaps** (OGRGeometryH, OGRGeometryH)

  Test for overlap.
- OGRGeometryH **OGR_G_Boundary** (OGRGeometryH)

  Compute boundary.
- OGRGeometryH **OGR_G_ConvexHull** (OGRGeometryH)

  Compute convex hull.
- OGRGeometryH **OGR_G_Buffer** (OGRGeometryH, double, int)

  Compute buffer of geometry.
- OGRGeometryH **OGR_G_Intersection** (OGRGeometryH, OGRGeometryH)

Compute intersection.

- OGRGeometryH **OGR_G_Union** (OGRGeometryH, OGRGeometryH)

  Compute union.
- OGRGeometryH **OGR_G_UnionCascaded** (OGRGeometryH)

  Compute union using cascading.
- OGRGeometryH **OGR_G_Difference** (OGRGeometryH, OGRGeometryH)

  Compute difference.
- OGRGeometryH **OGR_G_SymDifference** (OGRGeometryH, OGRGeometryH)

  Compute symmetric difference.
- double **OGR_G_Distance** (OGRGeometryH, OGRGeometryH)

  Compute distance between two geometries.
- double **OGR_G_Length** (OGRGeometryH)

  Compute length of a geometry.
- double **OGR_G_Area** (OGRGeometryH)

  Compute geometry area.
- int **OGR_G_Centroid** (OGRGeometryH, OGRGeometryH)

  Compute the geometry centroid.
- void **OGR_G_Empty** (OGRGeometryH)

  Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.
- int **OGR_G_IsEmpty** (OGRGeometryH)

  Test if the geometry is empty.
- int **OGR_G_IsValid** (OGRGeometryH)

  Test if the geometry is valid.
- int **OGR_G_IsSimple** (OGRGeometryH)

  Returns TRUE if the geometry is simple.
- int **OGR_G_IsRing** (OGRGeometryH)

  Test if the geometry is a ring.
- OGRGeometryH **OGR_G_Polygonize** (OGRGeometryH)

  Polygonizes a set of sparse edges.
- OGRGeometryH **OGR_G_SymmetricDifference** (OGRGeometryH, OGRGeometryH)

  Compute symmetric difference (deprecated)
- double **OGR_G_GetArea** (OGRGeometryH)

  Compute geometry area (deprecated)
- OGRGeometryH **OGR_G_GetBoundary** (OGRGeometryH)

  Compute boundary (deprecated)
- int **OGR_G_GetPointCount** (OGRGeometryH)

  Fetch number of points from a geometry.
- int **OGR_G_GetPoints** (OGRGeometryH hGeom, void ∗pabyX, int nXStride, void ∗pabyY, int nYStride, void ∗pabyZ, int nZStride)

  Returns all points of line string.
- double **OGR_G_GetX** (OGRGeometryH, int)

  Fetch the x coordinate of a point from a geometry.
- double **OGR_G_GetY** (OGRGeometryH, int)

  Fetch the x coordinate of a point from a geometry.
- double **OGR_G_GetZ** (OGRGeometryH, int)

  Fetch the z coordinate of a point from a geometry.
- void **OGR_G_GetPoint** (OGRGeometryH, int iPoint, double ∗, double ∗, double ∗)

  Fetch a point in line string or a point geometry.
- void **OGR_G_SetPoint** (OGRGeometryH, int iPoint, double, double, double)

  Set the location of a vertex in a point or linestring geometry.

- void **OGR_G_SetPoint_2D** (OGRGeometryH, int iPoint, double, double)

  Set the location of a vertex in a point or linestring geometry.
- void **OGR_G_AddPoint** (OGRGeometryH, double, double, double)

  Add a point to a geometry (line string or point).
- void **OGR_G_AddPoint_2D** (OGRGeometryH, double, double)

  Add a point to a geometry (line string or point).
- int **OGR_G_GetGeometryCount** (OGRGeometryH)

  Fetch the number of elements in a geometry or number of geometries in container.
- OGRGeometryH **OGR_G_GetGeometryRef** (OGRGeometryH, int)

  Fetch geometry from a geometry container.
- OGRErr **OGR_G_AddGeometry** (OGRGeometryH, OGRGeometryH)

  Add a geometry to a geometry container.
- OGRErr **OGR_G_AddGeometryDirectly** (OGRGeometryH, OGRGeometryH)

  Add a geometry directly to an existing geometry container.
- OGRErr **OGR_G_RemoveGeometry** (OGRGeometryH, int, int)

  Remove a geometry from an exiting geometry container.
- OGRGeometryH **OGRBuildPolygonFromEdges** (OGRGeometryH hLinesAsCollection, int bBestEffort, int bAutoClose, double dfTolerance, OGRErr ∗peErr)
- OGRErr **OGRSetGenerate_DB2_V72_BYTE_ORDER** (int bGenerate_DB2_V72_BYTE_ORDER)

  Special entry point to enable the hack for generating DB2 V7.2 style WKB.
- OGRFieldDefnH **OGR_Fld_Create** (const char ∗, **OGRFieldType**) CPL_WARN_UNUSED_RESULT

  Create a new field definition.
- void **OGR_Fld_Destroy** (OGRFieldDefnH)

  Destroy a field definition.
- void **OGR_Fld_SetName** (OGRFieldDefnH, const char ∗)

  Reset the name of this field.
- const char ∗ **OGR_Fld_GetNameRef** (OGRFieldDefnH)

  Fetch name of this field.
- **OGRFieldType OGR_Fld_GetType** (OGRFieldDefnH)

  Fetch type of this field.
- void **OGR_Fld_SetType** (OGRFieldDefnH, **OGRFieldType**)

  *Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 116) that is already part of an **OGRFeatureDefn** (p. 109).*
- **OGRJustification OGR_Fld_GetJustify** (OGRFieldDefnH)

  Get the justification for this field.
- void **OGR_Fld_SetJustify** (OGRFieldDefnH, **OGRJustification**)

  Set the justification for this field.
- int **OGR_Fld_GetWidth** (OGRFieldDefnH)

  Get the formatting width for this field.
- void **OGR_Fld_SetWidth** (OGRFieldDefnH, int)

  Set the formatting width for this field in characters.
- int **OGR_Fld_GetPrecision** (OGRFieldDefnH)

  Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.
- void **OGR_Fld_SetPrecision** (OGRFieldDefnH, int)

  Set the formatting precision for this field in characters.
- void **OGR_Fld_Set** (OGRFieldDefnH, const char ∗, **OGRFieldType**, int, int, **OGRJustification**)

  Set defining parameters for a field in one call.
- int **OGR_Fld_IsIgnored** (OGRFieldDefnH hDefn)

  Return whether this field should be omitted when fetching features.
- void **OGR_Fld_SetIgnored** (OGRFieldDefnH hDefn, int)

  Set whether this field should be omitted when fetching features.

- const char * **OGR_GetFieldTypeName** (**OGRFieldType**)

  Fetch human readable name for a field type.
- OGRFeatureDefnH **OGR_FD_Create** (const char *) CPL_WARN_UNUSED_RESULT

  Create a new feature definition object to hold the field definitions.
- void **OGR_FD_Destroy** (OGRFeatureDefnH)

  Destroy a feature definition object and release all memory associated with it.
- void **OGR_FD_Release** (OGRFeatureDefnH)

  Drop a reference, and destroy if unreferenced.
- const char * **OGR_FD_GetName** (OGRFeatureDefnH)

  *Get name of the **OGRFeatureDefn** (p. 109) passed as an argument.*
- int **OGR_FD_GetFieldCount** (OGRFeatureDefnH)

  Fetch number of fields on the passed feature definition.
- OGRFieldDefnH **OGR_FD_GetFieldDefn** (OGRFeatureDefnH, int)

  Fetch field definition of the passed feature definition.
- int **OGR_FD_GetFieldIndex** (OGRFeatureDefnH, const char *)

  Find field by name.
- void **OGR_FD_AddFieldDefn** (OGRFeatureDefnH, OGRFieldDefnH)

  Add a new field definition to the passed feature definition.
- OGRErr **OGR_FD_DeleteFieldDefn** (OGRFeatureDefnH hDefn, int iField)

  Delete an existing field definition.
- **OGRwkbGeometryType OGR_FD_GetGeomType** (OGRFeatureDefnH)

  Fetch the geometry base type of the passed feature definition.
- void **OGR_FD_SetGeomType** (OGRFeatureDefnH, **OGRwkbGeometryType**)

  Assign the base geometry type for the passed layer (the same as the feature definition).
- int **OGR_FD_IsGeometryIgnored** (OGRFeatureDefnH)

  Determine whether the geometry can be omitted when fetching features.
- void **OGR_FD_SetGeometryIgnored** (OGRFeatureDefnH, int)

  Set whether the geometry can be omitted when fetching features.
- int **OGR_FD_IsStyleIgnored** (OGRFeatureDefnH)

  Determine whether the style can be omitted when fetching features.
- void **OGR_FD_SetStyleIgnored** (OGRFeatureDefnH, int)

  Set whether the style can be omitted when fetching features.
- int **OGR_FD_Reference** (OGRFeatureDefnH)

  Increments the reference count by one.
- int **OGR_FD_Dereference** (OGRFeatureDefnH)

  Decrements the reference count by one.
- int **OGR_FD_GetReferenceCount** (OGRFeatureDefnH)

  Fetch current reference count.
- OGRFeatureH **OGR_F_Create** (OGRFeatureDefnH) CPL_WARN_UNUSED_RESULT

  Feature factory.
- void **OGR_F_Destroy** (OGRFeatureH)

  Destroy feature.
- OGRFeatureDefnH **OGR_F_GetDefnRef** (OGRFeatureH)

  Fetch feature definition.
- OGRErr **OGR_F_SetGeometryDirectly** (OGRFeatureH, OGRGeometryH)

  Set feature geometry.
- OGRErr **OGR_F_SetGeometry** (OGRFeatureH, OGRGeometryH)

  Set feature geometry.
- OGRGeometryH **OGR_F_GetGeometryRef** (OGRFeatureH)

  Fetch an handle to feature geometry.
- OGRGeometryH **OGR_F_StealGeometry** (OGRFeatureH)

Take away ownership of geometry.

- OGRFeatureH **OGR_F_Clone** (OGRFeatureH)

  Duplicate feature.

- int **OGR_F_Equal** (OGRFeatureH, OGRFeatureH)

  Test if two features are the same.

- int **OGR_F_GetFieldCount** (OGRFeatureH)

  *Fetch number of fields on this feature This will always be the same as the field count for the **OGRFeatureDefn** (p. 109).*

- OGRFieldDefnH **OGR_F_GetFieldDefnRef** (OGRFeatureH, int)

  Fetch definition for this field.

- int **OGR_F_GetFieldIndex** (OGRFeatureH, const char *)

  Fetch the field index given field name.

- int **OGR_F_IsFieldSet** (OGRFeatureH, int)

  Test if a field has ever been assigned a value or not.

- void **OGR_F_UnsetField** (OGRFeatureH, int)

  Clear a field, marking it as unset.

- **OGRField** * **OGR_F_GetRawFieldRef** (OGRFeatureH, int)

  Fetch an handle to the internal field value given the index.

- int **OGR_F_GetFieldAsInteger** (OGRFeatureH, int)

  Fetch field value as integer.

- double **OGR_F_GetFieldAsDouble** (OGRFeatureH, int)

  Fetch field value as a double.

- const char * **OGR_F_GetFieldAsString** (OGRFeatureH, int)

  Fetch field value as a string.

- const int * **OGR_F_GetFieldAsIntegerList** (OGRFeatureH, int, int *)

  Fetch field value as a list of integers.

- const double * **OGR_F_GetFieldAsDoubleList** (OGRFeatureH, int, int *)

  Fetch field value as a list of doubles.

- char ** **OGR_F_GetFieldAsStringList** (OGRFeatureH, int)

  Fetch field value as a list of strings.

- GByte * **OGR_F_GetFieldAsBinary** (OGRFeatureH, int, int *)

  Fetch field value as binary.

- int **OGR_F_GetFieldAsDateTime** (OGRFeatureH, int, int *, int *, int *, int *, int *, int *, int *)

  Fetch field value as date and time.

- void **OGR_F_SetFieldInteger** (OGRFeatureH, int, int)

  Set field to integer value.

- void **OGR_F_SetFieldDouble** (OGRFeatureH, int, double)

  Set field to double value.

- void **OGR_F_SetFieldString** (OGRFeatureH, int, const char *)

  Set field to string value.

- void **OGR_F_SetFieldIntegerList** (OGRFeatureH, int, int, int *)

  Set field to list of integers value.

- void **OGR_F_SetFieldDoubleList** (OGRFeatureH, int, int, double *)

  Set field to list of doubles value.

- void **OGR_F_SetFieldStringList** (OGRFeatureH, int, char **)

  Set field to list of strings value.

- void **OGR_F_SetFieldRaw** (OGRFeatureH, int, **OGRField** *)

  Set field.

- void **OGR_F_SetFieldBinary** (OGRFeatureH, int, int, GByte *)

  Set field to binary data.

- void **OGR_F_SetFieldDateTime** (OGRFeatureH, int, int, int, int, int, int, int, int)

Set field to datetime.

- long **OGR_F_GetFID** (OGRFeatureH)

  Get feature identifier.

- OGRErr **OGR_F_SetFID** (OGRFeatureH, long)

  Set the feature identifier.

- void **OGR_F_DumpReadable** (OGRFeatureH, FILE *)

  Dump this feature in a human readable form.

- OGRErr **OGR_F_SetFrom** (OGRFeatureH, OGRFeatureH, int)

  Set one feature from another.

- OGRErr **OGR_F_SetFromWithMap** (OGRFeatureH, OGRFeatureH, int, int *)

  Set one feature from another.

- const char * **OGR_F_GetStyleString** (OGRFeatureH)

  Fetch style string for this feature.

- void **OGR_F_SetStyleString** (OGRFeatureH, const char *)

  *Set feature style string. This method operate exactly as **OGR_F_SetStyleStringDirectly()** (p. 415) except that it does not assume ownership of the passed string, but instead makes a copy of it.*

- void **OGR_F_SetStyleStringDirectly** (OGRFeatureH, char *)

  *Set feature style string. This method operate exactly as **OGR_F_SetStyleString()** (p. 415) except that it assumes ownership of the passed string.*

- const char * **OGR_L_GetName** (OGRLayerH)

  Return the layer name.

- **OGRwkbGeometryType OGR_L_GetGeomType** (OGRLayerH)

  Return the layer geometry type.

- OGRGeometryH **OGR_L_GetSpatialFilter** (OGRLayerH)

  This function returns the current spatial filter for this layer.

- void **OGR_L_SetSpatialFilter** (OGRLayerH, OGRGeometryH)

  Set a new spatial filter.

- void **OGR_L_SetSpatialFilterRect** (OGRLayerH, double, double, double, double)

  Set a new rectangular spatial filter.

- OGRErr **OGR_L_SetAttributeFilter** (OGRLayerH, const char *)

  Set a new attribute query.

- void **OGR_L_ResetReading** (OGRLayerH)

  Reset feature reading to start on the first feature.

- OGRFeatureH **OGR_L_GetNextFeature** (OGRLayerH)

  Fetch the next available feature from this layer.

- OGRErr **OGR_L_SetNextByIndex** (OGRLayerH, long)

  Move read cursor to the nIndex'th feature in the current resultset.

- OGRFeatureH **OGR_L_GetFeature** (OGRLayerH, long)

  Fetch a feature by its identifier.

- OGRErr **OGR_L_SetFeature** (OGRLayerH, OGRFeatureH)

  Rewrite an existing feature.

- OGRErr **OGR_L_CreateFeature** (OGRLayerH, OGRFeatureH)

  Create and write a new feature within a layer.

- OGRErr **OGR_L_DeleteFeature** (OGRLayerH, long)

  Delete feature from layer.

- OGRFeatureDefnH **OGR_L_GetLayerDefn** (OGRLayerH)

  Fetch the schema information for this layer.

- OGRSpatialReferenceH **OGR_L_GetSpatialRef** (OGRLayerH)

  Fetch the spatial reference system for this layer.

- int **OGR_L_GetFeatureCount** (OGRLayerH, int)

  Fetch the feature count in this layer.

- OGRErr **OGR_L_GetExtent** (OGRLayerH, **OGREnvelope** *, int)

  Fetch the extent of this layer.
- int **OGR_L_TestCapability** (OGRLayerH, const char *)

  Test if this layer supported the named capability.
- OGRErr **OGR_L_CreateField** (OGRLayerH, OGRFieldDefnH, int)

  Create a new field on a layer.
- OGRErr **OGR_L_DeleteField** (OGRLayerH, int iField)

  Create a new field on a layer.
- OGRErr **OGR_L_ReorderFields** (OGRLayerH, int *panMap)

  Reorder all the fields of a layer.
- OGRErr **OGR_L_ReorderField** (OGRLayerH, int iOldFieldPos, int iNewFieldPos)

  Reorder an existing field on a layer.
- OGRErr **OGR_L_AlterFieldDefn** (OGRLayerH, int iField, OGRFieldDefnH hNewFieldDefn, int nFlags)

  Alter the definition of an existing field on a layer.
- OGRErr **OGR_L_StartTransaction** (OGRLayerH)

  For datasources which support transactions, StartTransaction creates a transaction.
- OGRErr **OGR_L_CommitTransaction** (OGRLayerH)

  For datasources which support transactions, CommitTransaction commits a transaction.
- OGRErr **OGR_L_RollbackTransaction** (OGRLayerH)

  For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction. If no transaction is active, or the rollback fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.
- OGRErr **OGR_L_SyncToDisk** (OGRLayerH)

  Flush pending changes to disk.
- const char * **OGR_L_GetFIDColumn** (OGRLayerH)

  This method returns the name of the underlying database column being used as the FID column, or "" if not supported.
- const char * **OGR_L_GetGeometryColumn** (OGRLayerH)

  This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.
- OGRErr **OGR_L_SetIgnoredFields** (OGRLayerH, const char **)

  Set which fields can be omitted when retrieving features from the layer.
- void **OGR_DS_Destroy** (OGRDataSourceH)

  Closes opened datasource and releases allocated resources.
- const char * **OGR_DS_GetName** (OGRDataSourceH)

  Returns the name of the data source.
- int **OGR_DS_GetLayerCount** (OGRDataSourceH)

  Get the number of layers in this data source.
- OGRLayerH **OGR_DS_GetLayer** (OGRDataSourceH, int)

  Fetch a layer by index.
- OGRLayerH **OGR_DS_GetLayerByName** (OGRDataSourceH, const char *)

  Fetch a layer by name.
- OGRErr **OGR_DS_DeleteLayer** (OGRDataSourceH, int)

  Delete the indicated layer from the datasource.
- OGRSFDriverH **OGR_DS_GetDriver** (OGRDataSourceH)

  Returns the driver that the dataset was opened with.
- OGRLayerH **OGR_DS_CreateLayer** (OGRDataSourceH, const char *, OGRSpatialReferenceH, **OGRwkbGeometryType**, char **)

  This function attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.
- OGRLayerH **OGR_DS_CopyLayer** (OGRDataSourceH, OGRLayerH, const char *, char **)

  Duplicate an existing layer.
- int **OGR_DS_TestCapability** (OGRDataSourceH, const char *)

Test if capability is available.

- OGRLayerH **OGR_DS_ExecuteSQL** (OGRDataSourceH, const char *, OGRGeometryH, const char *)

  Execute an SQL statement against the data store.

- void **OGR_DS_ReleaseResultSet** (OGRDataSourceH, OGRLayerH)

  *Release results of **OGR_DS_ExecuteSQL()** (p. 399).*

- OGRErr **OGR_DS_SyncToDisk** (OGRDataSourceH)

  Flush pending changes to disk.

- const char * **OGR_Dr_GetName** (OGRSFDriverH)

  Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".

- OGRDataSourceH **OGR_Dr_Open** (OGRSFDriverH, const char *, int) CPL_WARN_UNUSED_RESULT

  Attempt to open file with this driver.

- int **OGR_Dr_TestCapability** (OGRSFDriverH, const char *)

  Test if capability is available.

- OGRDataSourceH **OGR_Dr_CreateDataSource** (OGRSFDriverH, const char *, char **) CPL_WARN_UNUSED_RESULT

  This function attempts to create a new data source based on the passed driver.

- OGRDataSourceH **OGR_Dr_CopyDataSource** (OGRSFDriverH, OGRDataSourceH, const char *, char **) CPL_WARN_UNUSED_RESULT

  This function creates a new datasource by copying all the layers from the source datasource.

- OGRErr **OGR_Dr_DeleteDataSource** (OGRSFDriverH, const char *)

  Delete a datasource.

- OGRDataSourceH **OGROpen** (const char *, int, OGRSFDriverH *) CPL_WARN_UNUSED_RESULT

  Open a file / data source with one of the registered drivers.

- OGRErr **OGRReleaseDataSource** (OGRDataSourceH)

  Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.

- void **OGRRegisterDriver** (OGRSFDriverH)

  Add a driver to the list of registered drivers.

- void **OGRDeregisterDriver** (OGRSFDriverH)

  Remove the passed driver from the list of registered drivers.

- int **OGRGetDriverCount** (void)

  Fetch the number of registered drivers.

- OGRSFDriverH **OGRGetDriver** (int)

  Fetch the indicated driver.

- OGRSFDriverH **OGRGetDriverByName** (const char *)

  Fetch the indicated driver.

- int **OGRGetOpenDSCount** (void)

  Return the number of opened datasources.

- OGRDataSourceH **OGRGetOpenDS** (int iDS)

  Return the iDS th datasource opened.

- void **OGRRegisterAll** (void)

  Register all drivers.

- void **OGRCleanupAll** (void)

  Cleanup all OGR related resources.

- OGRStyleMgrH **OGR_SM_Create** (OGRStyleTableH hStyleTable) CPL_WARN_UNUSED_RESULT

  ***OGRStyleMgr** (p. 284) factory.*

- void **OGR_SM_Destroy** (OGRStyleMgrH hSM)

  Destroy Style Manager.

- const char * **OGR_SM_InitFromFeature** (OGRStyleMgrH hSM, OGRFeatureH hFeat)

  Initialize style manager from the style string of a feature.

- int **OGR_SM_InitStyleString** (OGRStyleMgrH hSM, const char *pszStyleString)

Initialize style manager from the style string.

- int **OGR_SM_GetPartCount** (OGRStyleMgrH hSM, const char *pszStyleString)

  Get the number of parts in a style.
- OGRStyleToolH **OGR_SM_GetPart** (OGRStyleMgrH hSM, int nPartId, const char *pszStyleString)

  Fetch a part (style tool) from the current style.
- int **OGR_SM_AddPart** (OGRStyleMgrH hSM, OGRStyleToolH hST)

  Add a part (style tool) to the current style.
- int **OGR_SM_AddStyle** (OGRStyleMgrH hSM, const char *pszStyleName, const char *pszStyleString)

  Add a style to the current style table.
- OGRStyleToolH **OGR_ST_Create** (**OGRSTClassId** eClassId) CPL_WARN_UNUSED_RESULT

  OGRStyleTool *(p. 292) factory.*
- void **OGR_ST_Destroy** (OGRStyleToolH hST)

  Destroy Style Tool.
- **OGRSTClassId OGR_ST_GetType** (OGRStyleToolH hST)

  Determine type of Style Tool.
- **OGRSTUnitId OGR_ST_GetUnit** (OGRStyleToolH hST)

  Get Style Tool units.
- void **OGR_ST_SetUnit** (OGRStyleToolH hST, **OGRSTUnitId** eUnit, double dfGroundPaperScale)

  Set Style Tool units.
- const char * **OGR_ST_GetParamStr** (OGRStyleToolH hST, int eParam, int *bValueIsNull)

  Get Style Tool parameter value as string.
- int **OGR_ST_GetParamNum** (OGRStyleToolH hST, int eParam, int *bValueIsNull)

  Get Style Tool parameter value as an integer.
- double **OGR_ST_GetParamDbl** (OGRStyleToolH hST, int eParam, int *bValueIsNull)

  Get Style Tool parameter value as a double.
- void **OGR_ST_SetParamStr** (OGRStyleToolH hST, int eParam, const char *pszValue)

  Set Style Tool parameter value from a string.
- void **OGR_ST_SetParamNum** (OGRStyleToolH hST, int eParam, int nValue)

  Set Style Tool parameter value from an integer.
- void **OGR_ST_SetParamDbl** (OGRStyleToolH hST, int eParam, double dfValue)

  Set Style Tool parameter value from a double.
- const char * **OGR_ST_GetStyleString** (OGRStyleToolH hST)

  Get the style string for this Style Tool.
- int **OGR_ST_GetRGBFromString** (OGRStyleToolH hST, const char *pszColor, int *pnRed, int *pnGreen, int *pnBlue, int *pnAlpha)

  Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.
- OGRStyleTableH **OGR_STBL_Create** (void) CPL_WARN_UNUSED_RESULT

  OGRStyleTable *(p. 289) factory.*
- void **OGR_STBL_Destroy** (OGRStyleTableH hSTBL)

  Destroy Style Table.
- int **OGR_STBL_SaveStyleTable** (OGRStyleTableH hStyleTable, const char *pszFilename)

  Save a style table to a file.
- int **OGR_STBL_LoadStyleTable** (OGRStyleTableH hStyleTable, const char *pszFilename)

  Load a style table from a file.
- const char * **OGR_STBL_Find** (OGRStyleTableH hStyleTable, const char *pszName)

  Get a style string by name.
- void **OGR_STBL_ResetStyleStringReading** (OGRStyleTableH hStyleTable)

  Reset the next style pointer to 0.
- const char * **OGR_STBL_GetNextStyle** (OGRStyleTableH hStyleTable)

  Get the next style string from the table.
- const char * **OGR_STBL_GetLastStyleName** (OGRStyleTableH hStyleTable)

12.12.1 Detailed Description

C API and defines for **OGRFeature** (p. 94), **OGRGeometry** (p. 126), and **OGRDataSource** (p. 85) related classes.

See also: **ogr_geometry.h** (p. 489), **ogr_feature.h** (p. 488), **ogrsf_frmts.h** (p. 518), **ogr_featurestyle.h** (p. 489)

12.12.2 Function Documentation

12.12.2.1 OGRDataSourceH OGR_Dr_CopyDataSource (OGRSFDriverH *hDriver,* OGRDataSourceH *hSrcDS,* const char * *pszNewName,* char ** *papszOptions*)

This function creates a new datasource by copying all the layers from the source datasource.

It is important to call **OGR_DS_Destroy()** (p. 399) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This function is the same as the C++ method **OGRSFDriver::CopyDataSource()** (p. 223).

Parameters

hDriver	handle to the driver on which data source creation is based.
hSrcDS	source datasource
pszNewName	the name for the new data source.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. 85) handle on success.

References OGRDataSource::GetDriver(), OGR_Dr_CopyDataSource(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_CopyDataSource().

12.12.2.2 OGRDataSourceH OGR_Dr_CreateDataSource (OGRSFDriverH *hDriver,* const char * *pszName,* char ** *papszOptions*)

This function attempts to create a new data source based on the passed driver.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

It is important to call **OGR_DS_Destroy()** (p. 399) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This function is the same as the C++ method **OGRSFDriver::CreateDataSource()** (p. 223).

Parameters

hDriver	handle to the driver on which data source creation is based.
pszName	the name for the new data source. UTF-8 encoded.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. 85) handle on success.

References OGRSFDriver::CreateDataSource(), OGRDataSource::GetDriver(), OGR_Dr_CreateDataSource(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_CreateDataSource().

12.12.2.3 OGRErr OGR_Dr_DeleteDataSource (OGRSFDriverH *hDriver,* const char * *pszDataSource*)

Delete a datasource.

Delete (from the disk, in the database, ...) the named datasource. Normally it would be safest if the datasource was not open at the time.

Whether this is a supported operation on this driver case be tested using TestCapability() on ODrCDeleteDataSource.

This method is the same as the C++ method **OGRSFDriver::DeleteDataSource()** (p. 224).

Parameters

hDriver	handle to the driver on which data source deletion is based.
pszDataSource	the name of the datasource to delete.

Returns

OGRERR_NONE on success, and OGRERR_UNSUPPORTED_OPERATION if this is not supported by this driver.

References OGR_Dr_DeleteDataSource().

Referenced by OGR_Dr_DeleteDataSource().

12.12.2.4 const char * OGR_Dr_GetName (OGRSFDriverH *hDriver*)

Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".

This function is the same as the C++ method **OGRSFDriver::GetName()** (p. 224).

Parameters

hDriver	handle to the the driver to get the name from.

Returns

driver name. This is an internal string and should not be modified or freed.

References OGR_Dr_GetName().

Referenced by OGR_Dr_GetName().

12.12.2.5 OGRDataSourceH OGR_Dr_Open (OGRSFDriverH *hDriver,* const char * *pszName,* int *bUpdate*)

Attempt to open file with this driver.

This function is the same as the C++ method **OGRSFDriver::Open()** (p. 225).

Parameters

hDriver	handle to the driver that is used to open file.
pszName	the name of the file, or data source to try and open.
bUpdate	TRUE if update access is required, otherwise FALSE (the default).

Returns

NULL on error or if the pass name is not supported by this driver, otherwise an handle to an **OGRDataSource** (p. 85). This **OGRDataSource** (p. 85) should be closed by deleting the object when it is no longer needed.

References OGRDataSource::GetDriver(), OGR_Dr_Open(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_Open().

12.12.2.6 int OGR_Dr_TestCapability (OGRSFDriverH *hDriver,* const char * *pszCap*)

Test if capability is available.

One of the following data source capability names can be passed into this function, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODrCCreateDataSource**: True if this driver can support creating data sources.
- **ODrCDeleteDataSource**: True if this driver supports deleting data sources.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid mispelling.

This function is the same as the C++ method **OGRSFDriver::TestCapability()** (p. 225).

Parameters

hDriver	handle to the driver to test the capability against.
pszCap	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

References OGR_Dr_TestCapability().

Referenced by OGR_Dr_TestCapability().

12.12.2.7 OGRLayerH OGR_DS_CopyLayer (OGRDataSourceH *hDS,* OGRLayerH *hSrcLayer,* const char * *pszNewName,* char ** *papszOptions*)

Duplicate an existing layer.

This function creates a new layer, duplicate the field definitions of the source layer and then duplicate each features of the source layer. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation. The source layer may come from another dataset.

This function is the same as the C++ method **OGRDataSource::CopyLayer** (p. 87)

Parameters

hDS	handle to the data source where to create the new layer
hSrcLayer	handle to the source layer.
pszNewName	the name of the layer to create.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

an handle to the layer, or NULL if an error occurs.

References OGR_DS_CopyLayer().

Referenced by OGR_DS_CopyLayer().

12.12.2.8 OGRLayerH OGR_DS_CreateLayer (OGRDataSourceH *hDS*, const char * *pszName*, OGRSpatialReferenceH *hSpatialRef*, OGRwkbGeometryType *eType*, char ** *papszOptions*)

This function attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

This function is the same as the C++ method **OGRDataSource::CreateLayer()** (p. 87).

Parameters

hDS	The dataset handle.
pszName	the name for the new layer. This should ideally not match any existing layer on the datasource.
hSpatialRef	handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available.
eType	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRLayer** (p. 162) handle on success.

Example:

```
#include "ogrsf_frmts.h"
#include "cpl_string.h"

...

        OGRLayerH *hLayer;
        char     *papszOptions;

        if( OGR_DS_TestCapability( hDS, ODsCCreateLayer ) )
        {
            ...
        }

        papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
        hLayer = OGR_DS_CreateLayer( hDS, "NewLayer", NULL, wkbUnknown,
                                     papszOptions );
        CSLDestroy( papszOptions );

        if( hLayer == NULL )
        {
            ...
        }
```

References OGR_DS_CreateLayer().

Referenced by OGR_DS_CreateLayer().

12.12.2.9 OGRErr OGR_DS_DeleteLayer (OGRDataSourceH *hDS*, int *iLayer*)

Delete the indicated layer from the datasource.

If this method is supported the ODsCDeleteLayer capability will test TRUE on the **OGRDataSource** (p. 85).

This method is the same as the C++ method **OGRDataSource::DeleteLayer()** (p. 88).

Parameters

hDS	handle to the datasource
iLayer	the index of the layer to delete.

Returns

OGRERR_NONE on success, or OGRERR_UNSUPPORTED_OPERATION if deleting layers is not supported for this datasource.

References OGR_DS_DeleteLayer().

Referenced by OGR_DS_DeleteLayer().

12.12.2.10 void OGR_DS_Destroy (OGRDataSourceH *hDataSource*)

Closes opened datasource and releases allocated resources.

This method is the same as the C++ method **OGRDataSource::DestroyDataSource()** (p. 88).

Parameters

hDataSource	handle to allocated datasource object.

References OGR_DS_Destroy().

Referenced by OGR_DS_Destroy().

12.12.2.11 OGRLayerH OGR_DS_ExecuteSQL (OGRDataSourceH *hDS,* const char * *pszSQLCommand,* OGRGeometryH *hSpatialFilter,* const char * *pszDialect*)

Execute an SQL statement against the data store.

The result of an SQL query is either NULL for statements that are in error, or that have no results set, or an **OGRLayer** (p. 162) handle representing a results set from the query. Note that this **OGRLayer** (p. 162) is in addition to the layers in the data store and must be destroyed with **OGR_DS_ReleaseResultSet()** (p. 401) before the data source is closed (destroyed).

For more information on the SQL dialect supported internally by OGR review the `OGR SQL` document. Some drivers (ie. Oracle and PostGIS) pass the SQL directly through to the underlying RDBMS.

This function is the same as the C++ method **OGRDataSource::ExecuteSQL()** (p. 89);

Parameters

hDS	handle to the data source on which the SQL query is executed.
pszSQL-Command	the SQL statement to execute.
hSpatialFilter	handle to a geometry which represents a spatial filter. Can be NULL.
pszDialect	allows control of the statement dialect. If set to NULL, the OGR SQL engine will be used, except for RDBMS drivers that will use their dedicated SQL engine, unless OGRSQL is explicitely passed as the dialect.

Returns

an handle to a **OGRLayer** (p. 162) containing the results of the query. Deallocate with **OGR_DS_ReleaseResultSet()** (p. 401).

References OGR_DS_ExecuteSQL().

Referenced by OGR_DS_ExecuteSQL().

12.12.2.12 OGRSFDriverH OGR_DS_GetDriver (OGRDataSourceH *hDS*)

Returns the driver that the dataset was opened with.

This method is the same as the C++ method **OGRDataSource::GetDriver()** (p. 89)

Parameters

hDS	handle to the datasource

Returns

NULL if driver info is not available, or pointer to a driver owned by the OGRSFDriverManager.

References OGR_DS_GetDriver().

Referenced by OGR_DS_GetDriver().

12.12.2.13 OGRLayerH OGR_DS_GetLayer (OGRDataSourceH *hDS*, int *iLayer*)

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. 85) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayer()** (p. 89).

Parameters

hDS	handle to the data source from which to get the layer.
iLayer	a layer number between 0 and **OGR_DS_GetLayerCount()** (p. 401)-1.

Returns

an handle to the layer, or NULL if iLayer is out of range or an error occurs.

References OGR_DS_GetLayer().

Referenced by OGR_DS_GetLayer().

12.12.2.14 OGRLayerH OGR_DS_GetLayerByName (OGRDataSourceH *hDS*, const char * *pszLayerName*)

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. 85) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayerByName()** (p. 90).

Parameters

hDS	handle to the data source from which to get the layer.
pszLayerName	Layer the layer name of the layer to fetch.

Returns

an handle to the layer, or NULL if the layer is not found or an error occurs.

References OGR_DS_GetLayerByName().

Referenced by OGR_DS_GetLayerByName().

12.12.2.15 int OGR_DS_GetLayerCount (OGRDataSourceH *hDS*)

Get the number of layers in this data source.

This function is the same as the C++ method **OGRDataSource::GetLayerCount()** (p. 90).

Parameters

hDS	handle to the data source from which to get the number of layers.

Returns

layer count.

References OGR_DS_GetLayerCount().

Referenced by OGR_DS_GetLayerCount().

12.12.2.16 const char * OGR_DS_GetName (OGRDataSourceH *hDS*)

Returns the name of the data source.

This string should be sufficient to open the data source if passed to the same **OGRSFDriver** (p. 222) that this data source was opened with, but it need not be exactly the same string that was used to open the data source. Normally this is a filename.

This function is the same as the C++ method **OGRDataSource::GetName()** (p. 90).

Parameters

hDS	handle to the data source to get the name from.

Returns

pointer to an internal name string which should not be modified or freed by the caller.

References OGR_DS_GetName().

Referenced by OGR_DS_GetName().

12.12.2.17 void OGR_DS_ReleaseResultSet (OGRDataSourceH *hDS*, OGRLayerH *hLayer*)

Release results of **OGR_DS_ExecuteSQL()** (p. 399).

This function should only be used to deallocate OGRLayers resulting from an **OGR_DS_ExecuteSQL()** (p. 399) call on the same **OGRDataSource** (p. 85). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 85) may cause errors.

This function is the same as the C++ method **OGRDataSource::ReleaseResultSet()** (p. 91).

Parameters

hDS	an handle to the data source on which was executed an SQL query.
hLayer	handle to the result of a previous **OGR_DS_ExecuteSQL()** (p. 399) call.

References OGR_DS_ReleaseResultSet().

Referenced by OGR_DS_ReleaseResultSet().

12.12.2.18 OGRErr OGR_DS_SyncToDisk (OGRDataSourceH *hDS*)

Flush pending changes to disk.

This call is intended to force the datasource to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some data sources do not implement this method, and will still return OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

The default implementation of this method just calls the SyncToDisk() method on each of the layers. Conceptionally, calling SyncToDisk() on a datasource should include any work that might be accomplished by calling SyncToDisk() on layers in that data source.

In any event, you should always close any opened datasource with **OGR_DS_Destroy()** (p. 399) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRDataSource::SyncToDisk()** (p. 92)

Parameters

hDS	handle to the data source

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

References OGR_DS_SyncToDisk().

Referenced by OGR_DS_SyncToDisk().

12.12.2.19 int OGR_DS_TestCapability (OGRDataSourceH *hDS,* const char * *pszCapability*)

Test if capability is available.

One of the following data source capability names can be passed into this function, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODsCCreateLayer**: True if this datasource can create new layers.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid mispelling.

This function is the same as the C++ method **OGRDataSource::TestCapability()** (p. 93).

Parameters

hDS	handle to the data source against which to test the capability.
pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

References OGR_DS_TestCapability().

Referenced by OGR_DS_TestCapability().

12.12.2.20 OGRFeatureH OGR_F_Clone (OGRFeatureH *hFeat*)

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. 109).

This function is the same as the C++ method **OGRFeature::Clone()** (p. 96).

Parameters

hFeat	handle to the feature to clone.

Returns

an handle to the new feature, exactly matching this feature.

References OGR_F_Clone().

Referenced by OGR_F_Clone().

12.12.2.21 OGRFeatureH OGR_F_Create (OGRFeatureDefnH *hDefn*)

Feature factory.

Note that the **OGRFeature** (p. 94) will increment the reference count of it's defining **OGRFeatureDefn** (p. 109). Destruction of the **OGRFeatureDefn** (p. 109) before destruction of all OGRFeatures that depend on it is likely to result in a crash.

This function is the same as the C++ method **OGRFeature::OGRFeature()** (p. 96).

Parameters

hDefn	handle to the feature class (layer) definition to which the feature will adhere.

Returns

an handle to the new feature object with null fields and no geometry.

References OGR_F_Create().

Referenced by OGR_F_Create().

12.12.2.22 void OGR_F_Destroy (OGRFeatureH *hFeat*)

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This function is the same as the C++ method **OGRFeature::DestroyFeature()** (p. 97).

Parameters

hFeat	handle to the feature to destroy.

References OGR_F_Destroy().

Referenced by OGR_F_Destroy().

12.12.2.23 void OGR_F_DumpReadable (OGRFeatureH *hFeat,* FILE * *fpOut*)

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

This function is the same as the C++ method **OGRFeature::DumpReadable()** (p. 97).

Parameters

hFeat	handle to the feature to dump.
fpOut	the stream to write to, such as strout.

References OGR_F_DumpReadable().

Referenced by OGR_F_DumpReadable().

12.12.2.24 int OGR_F_Equal (OGRFeatureH *hFeat,* OGRFeatureH *hOtherFeat*)

Test if two features are the same.

Two features are considered equal if the share them (handle equality) same **OGRFeatureDefn** (p. 109), have the same field values, and the same geometry (as tested by OGR_G_Equal()) as well as the same feature id.

This function is the same as the C++ method **OGRFeature::Equal()** (p. 98).

Parameters

hFeat	handle to one of the feature.
hOtherFeat	handle to the other feature to test this one against.

Returns

TRUE if they are equal, otherwise FALSE.

References OGR_F_Equal().

Referenced by OGR_F_Equal().

12.12.2.25 OGRFeatureDefnH OGR_F_GetDefnRef (OGRFeatureH *hFeat*)

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. 98).

Parameters

hFeat	handle to the feature to get the feature definition from.

Returns

an handle to the feature definition object on which feature depends.

References OGR_F_GetDefnRef().

Referenced by OGR_F_GetDefnRef().

12.12.2.26 long OGR_F_GetFID (OGRFeatureH *hFeat*)

Get feature identifier.

This function is the same as the C++ method **OGRFeature::GetFID()** (p. 98).

Parameters

hFeat	handle to the feature from which to get the feature identifier.

Returns

feature id or OGRNullFID if none has been assigned.

References OGR_F_GetFID().

Referenced by OGR_F_GetFID().

12.12.2.27 GByte* OGR_F_GetFieldAsBinary (OGRFeatureH *hFeat,* int *iField,* int * *pnBytes*)

Fetch field value as binary.

Currently this method only works for OFTBinary fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsBinary()** (p. 99).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnBytes	location to place count of bytes returned.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGR_F_GetFieldAsBinary().

Referenced by OGR_F_GetFieldAsBinary().

12.12.2.28 int OGR_F_GetFieldAsDateTime (OGRFeatureH *hFeat,* int *iField,* int * *pnYear,* int * *pnMonth,* int * *pnDay,* int * *pnHour,* int * *pnMinute,* int * *pnSecond,* int * *pnTZFlag*)

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDateTime()** (p. 99).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnYear	(including century)
pnMonth	(1-12)
pnDay	(1-31)
pnHour	(0-23)
pnMinute	(0-59)
pnSecond	(0-59)
pnTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns

TRUE on success or FALSE on failure.

References OGR_F_GetFieldAsDateTime().

Referenced by OGR_F_GetFieldAsDateTime().

12.12.2.29 double OGR_F_GetFieldAsDouble (OGRFeatureH *hFeat,* int *iField*)

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsDouble()** (p. 99).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

References OGR_F_GetFieldAsDouble().

Referenced by OGR_F_GetFieldAsDouble().

12.12.2.30 const double* OGR_F_GetFieldAsDoubleList (OGRFeatureH *hFeat,* int *iField,* int * *pnCount*)

Fetch field value as a list of doubles.

Currently this function only works for OFTRealList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDoubleList()** (p. 100).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnCount	an integer to put the list count (number of doubles) into.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGR_F_GetFieldAsDoubleList().

Referenced by OGR_F_GetFieldAsDoubleList().

12.12.2.31 int OGR_F_GetFieldAsInteger (OGRFeatureH *hFeat,* int *iField*)

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsInteger()** (p. 100).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

References OGR_F_GetFieldAsInteger().

Referenced by OGR_F_GetFieldAsInteger().

12.12.2.32 const int* OGR_F_GetFieldAsIntegerList (OGRFeatureH *hFeat*, int *iField*, int * *pnCount*)

Fetch field value as a list of integers.

Currently this function only works for OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsIntegerList()** (p. 100).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnCount	an integer to put the list count (number of integers) into.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGR_F_GetFieldAsIntegerList().

Referenced by OGR_F_GetFieldAsIntegerList().

12.12.2.33 const char* OGR_F_GetFieldAsString (OGRFeatureH *hFeat*, int *iField*)

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using sprintf(), but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsString()** (p. 101).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGR_F_GetFieldAsString().

Referenced by OGR_F_GetFieldAsString().

12.12.2.34 char** OGR_F_GetFieldAsStringList (OGRFeatureH *hFeat*, int *iField*)

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. 365).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. 101).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGR_F_GetFieldAsStringList().

Referenced by OGR_F_GetFieldAsStringList().

12.12.2.35 int OGR_F_GetFieldCount (OGRFeatureH *hFeat*)

Fetch number of fields on this feature This will always be the same as the field count for the **OGRFeatureDefn** (p. 109).

This function is the same as the C++ method **OGRFeature::GetFieldCount()** (p. 102).

Parameters

hFeat	handle to the feature to get the fields count from.

Returns

count of fields.

References OGR_F_GetFieldCount().

Referenced by OGR_F_GetFieldCount().

12.12.2.36 OGRFieldDefnH OGR_F_GetFieldDefnRef (OGRFeatureH *hFeat,* int *i*)

Fetch definition for this field.

This function is the same as the C++ method **OGRFeature::GetFieldDefnRef()** (p. 102).

Parameters

hFeat	handle to the feature on which the field is found.
i	the field to fetch, from 0 to GetFieldCount()-1.

Returns

an handle to the field definition (from the **OGRFeatureDefn** (p. 109)). This is an internal reference, and should not be deleted or modified.

References OGR_F_GetFieldDefnRef().

Referenced by OGR_F_GetFieldDefnRef().

12.12.2.37 int OGR_F_GetFieldIndex (OGRFeatureH *hFeat,* const char * *pszName*)

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. 112) method.

This function is the same as the C++ method **OGRFeature::GetFieldIndex()** (p. 102).

Parameters

hFeat	handle to the feature on which the field is found.
pszName	the name of the field to search for.

Returns

the field index, or -1 if no matching field is found.

References OGR_F_GetFieldIndex().

Referenced by OGR_F_GetFieldIndex().

12.12.2.38 OGRGeometryH OGR_F_GetGeometryRef (OGRFeatureH *hFeat*)

Fetch an handle to feature geometry.

This function is the same as the C++ method **OGRFeature::GetGeometryRef()** (p. 102).

Parameters

hFeat	handle to the feature to get geometry from.

Returns

an handle to internal feature geometry. This object should not be modified.

References OGR_F_GetGeometryRef().

Referenced by OGR_F_GetGeometryRef().

12.12.2.39 OGRField* OGR_F_GetRawFieldRef (OGRFeatureH *hFeat,* int *iField*)

Fetch an handle to the internal field value given the index.

This function is the same as the C++ method **OGRFeature::GetRawFieldRef()** (p. 103).

Parameters

hFeat	handle to the feature on which field is found.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the returned handle is to an internal data structure, and should not be freed, or modified.

References OGR_F_GetRawFieldRef().

Referenced by OGR_F_GetRawFieldRef().

12.12.2.40 const char* OGR_F_GetStyleString (OGRFeatureH *hFeat*)

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr_featurestyle.h** (p. 489) for services available to parse it.

This function is the same as the C++ method **OGRFeature::GetStyleString()** (p. 103).

Parameters

hFeat	handle to the feature to get the style from.

Returns

a reference to a representation in string format, or NULL if there isn't one.

References OGR_F_GetStyleString().

Referenced by OGR_F_GetStyleString().

12.12.2.41 int OGR_F_IsFieldSet (OGRFeatureH *hFeat,* int *iField*)

Test if a field has ever been assigned a value or not.

This function is the same as the C++ method **OGRFeature::IsFieldSet()** (p. 103).

Parameters

hFeat	handle to the feature on which the field is.
iField	the field to test.

Returns

TRUE if the field has been set, otherwise false.

References OGRFeature::GetFieldCount(), OGRFeature::IsFieldSet(), and OGR_F_IsFieldSet().

Referenced by OGR_F_IsFieldSet().

12.12.2.42 OGRErr OGR_F_SetFID (OGRFeatureH *hFeat,* long *nFID*)

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This function is the same as the C++ method **OGRFeature::SetFID()** (p. 104).

Parameters

hFeat	handle to the feature to set the feature id to.
nFID	the new feature identifier value to assign.

Returns

On success OGRERR_NONE, or on failure some other value.

References OGR_F_SetFID().

Referenced by OGR_F_SetFID().

12.12.2.43 void OGR_F_SetFieldBinary (OGRFeatureH *hFeat*, int *iField*, int *nBytes*, GByte * *pabyData*)

Set field to binary data.

This function currently on has an effect of OFTBinary fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nBytes	the number of bytes in pabyData array.
pabyData	the data to apply.

References OGR_F_SetFieldBinary().

Referenced by OGR_F_SetFieldBinary().

12.12.2.44 void OGR_F_SetFieldDateTime (OGRFeatureH *hFeat*, int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour*, int *nMinute*, int *nSecond*, int *nTZFlag*)

Set field to datetime.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nYear	(including century)
nMonth	(1-12)
nDay	(1-31)
nHour	(0-23)
nMinute	(0-59)
nSecond	(0-59)
nTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGR_F_SetFieldDateTime().

Referenced by OGR_F_SetFieldDateTime().

12.12.2.45 void OGR_F_SetFieldDouble (OGRFeatureH *hFeat*, int *iField*, double *dfValue*)

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
dfValue	the value to assign.

References OGR_F_SetFieldDouble().

Referenced by OGR_F_SetFieldDouble().

12.12.2.46 void OGR_F_SetFieldDoubleList (OGRFeatureH *hFeat*, int *iField*, int *nCount*, double * *padfValues*)

Set field to list of doubles value.

This function currently on has an effect of OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nCount	the number of values in the list being assigned.
padfValues	the values to assign.

References OGR_F_SetFieldDoubleList().

Referenced by OGR_F_SetFieldDoubleList().

12.12.2.47 void OGR_F_SetFieldInteger (OGRFeatureH *hFeat*, int *iField*, int *nValue*)

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
nValue	the value to assign.

References OGR_F_SetFieldInteger().

Referenced by OGR_F_SetFieldInteger().

12.12.2.48 void OGR_F_SetFieldIntegerList (OGRFeatureH *hFeat*, int *iField*, int *nCount*, int * *panValues*)

Set field to list of integers value.

This function currently on has an effect of OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nCount	the number of values in the list being assigned.
panValues	the values to assign.

References OGR_F_SetFieldIntegerList().

Referenced by OGR_F_SetFieldIntegerList().

12.12.2.49 void OGR_F_SetFieldRaw (OGRFeatureH *hFeat*, int *iField*, OGRField * *psValue*)

Set field.

The passed value **OGRField** (p. 116) must be of exactly the same type as the target field, or an application crash may occur. The passed value is copied, and will not be affected. It remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
psValue	handle on the value to assign.

References OGR_F_SetFieldRaw().

Referenced by OGR_F_SetFieldRaw().

12.12.2.50 void OGR_F_SetFieldString (OGRFeatureH *hFeat*, int *iField*, const char * *pszValue*)

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pszValue	the value to assign.

References OGR_F_SetFieldString().

Referenced by OGR_F_SetFieldString().

12.12.2.51 void OGR_F_SetFieldStringList (OGRFeatureH *hFeat*, int *iField*, char ** *papszValues*)

Set field to list of strings value.

This function currently on has an effect of OFTStringList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 104).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
papszValues	the values to assign.

References OGR_F_SetFieldStringList().

Referenced by OGR_F_SetFieldStringList().

12.12.2.52 OGRErr OGR_F_SetFrom (OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat*, int *bForgiving*)

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The hOtherFeature does not need to have the same **OGRFeatureDefn** (p. 109). Field values are copied by corresponding field names. Field

types do not have to exactly match. OGR_F_SetField*() function conversion rules will be applied as needed.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. 107).

Parameters

hFeat	handle to the feature to set to.
hOtherFeat	handle to the feature from which geometry, and field values will be copied.
bForgiving	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References OGR_F_SetFrom().

Referenced by OGR_F_SetFrom().

12.12.2.53 OGRErr OGR_F_SetFromWithMap (OGRFeatureH *hFeat,* OGRFeatureH *hOtherFeat,* int *bForgiving,* int * *panMap*)

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The hOtherFeature does not need to have the same **OGRFeatureDefn** (p. 109). Field values are copied according to the provided indices map. Field types do not have to exactly match. OGR_F_SetField*() function conversion rules will be applied as needed. This is more efficient than **OGR_F_SetFrom()** (p. 413) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. 107).

Parameters

hFeat	handle to the feature to set to.
hOtherFeat	handle to the feature from which geometry, and field values will be copied.
panMap	Array of the indices of the destination feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
bForgiving	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References OGR_F_SetFromWithMap().

Referenced by OGR_F_SetFromWithMap().

12.12.2.54 OGRErr OGR_F_SetGeometry (OGRFeatureH *hFeat,* OGRGeometryH *hGeom*)

Set feature geometry.

This function updates the features geometry, and operate exactly as SetGeometryDirectly(), except that this function does not assume ownership of the passed geometry, but instead makes a copy of it.

This function is the same as the C++ **OGRFeature::SetGeometry()** (p. 107).

Parameters

hFeat	handle to the feature on which new geometry is applied to.
hGeom	handle to the new geometry to apply to feature.

Returns

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 109) (checking not yet implemented).

References OGR_F_SetGeometry().

Referenced by OGR_F_SetGeometry().

12.12.2.55 OGRErr OGR_F_SetGeometryDirectly (OGRFeatureH *hFeat,* OGRGeometryH *hGeom*)

Set feature geometry.

This function updates the features geometry, and operate exactly as SetGeometry(), except that this function assumes ownership of the passed geometry.

This function is the same as the C++ method **OGRFeature::SetGeometryDirectly** (p. 108).

Parameters

hFeat	handle to the feature on which to apply the geometry.
hGeom	handle to the new geometry to apply to feature.

Returns

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 109) (checking not yet implemented).

References OGR_F_SetGeometryDirectly().

Referenced by OGR_F_SetGeometryDirectly().

12.12.2.56 void OGR_F_SetStyleString (OGRFeatureH *hFeat,* const char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleStringDirectly()** (p. 415) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This function is the same as the C++ method **OGRFeature::SetStyleString()** (p. 108).

Parameters

hFeat	handle to the feature to set style to.
pszStyle	the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleString().

Referenced by OGR_F_SetStyleString().

12.12.2.57 void OGR_F_SetStyleStringDirectly (OGRFeatureH *hFeat,* char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleString()** (p. 415) except that it assumes ownership of the passed string.

This function is the same as the C++ method **OGRFeature::SetStyleStringDirectly()** (p. 108).

Parameters

hFeat	handle to the feature to set style to.
pszStyle	the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleStringDirectly().

Referenced by OGR_F_SetStyleStringDirectly().

12.12.2.58 OGRGeometryH OGR_F_StealGeometry (OGRFeatureH *hFeat*)

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownship of the geometry from the feature without copying. Sort of an inverse to OGR_FSetGeometryDirectly().

After this call the **OGRFeature** (p. 94) will have a NULL geometry.

Returns

the pointer to the geometry.

References OGR_F_StealGeometry().

Referenced by OGR_F_StealGeometry().

12.12.2.59 void OGR_F_UnsetField (OGRFeatureH *hFeat,* int *iField*)

Clear a field, marking it as unset.

This function is the same as the C++ method **OGRFeature::UnsetField()** (p. 109).

Parameters

hFeat	handle to the feature on which the field is.
iField	the field to unset.

References OGR_F_UnsetField().

Referenced by OGR_F_UnsetField().

12.12.2.60 void OGR_FD_AddFieldDefn (OGRFeatureDefnH *hDefn,* OGRFieldDefnH *hNewField*)

Add a new field definition to the passed feature definition.

To add a new field definition to a layer definition, do not use this function directly, but use **OGR_L_CreateField()** (p. 458) instead.

This function should only be called while there are no **OGRFeature** (p. 94) objects in existance based on this **OGRFeatureDefn** (p. 109). The **OGRFieldDefn** (p. 116) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeatureDefn::AddFieldDefn()** (p. 111).

Parameters

hDefn	handle to the feature definition to add the field definition to.
hNewField	handle to the new field definition.

References OGR_FD_AddFieldDefn().

Referenced by OGR_FD_AddFieldDefn().

12.12.2.61 OGRFeatureDefnH OGR_FD_Create (const char * *pszName*)

Create a new feature definition object to hold the field definitions.

The **OGRFeatureDefn** (p. 109) maintains a reference count, but this starts at zero, and should normally be incremented by the owner.

This function is the same as the C++ method **OGRFeatureDefn::OGRFeatureDefn()** (p. 110).

Parameters

pszName	the name to be assigned to this layer/class. It does not need to be unique.

Returns

handle to the newly created feature definition.

References OGR_FD_Create().

Referenced by OGR_FD_Create().

12.12.2.62 OGRErr OGR_FD_DeleteFieldDefn (OGRFeatureDefnH *hDefn,* int *iField*)

Delete an existing field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use **OGR_L_DeleteField()** (p. 459) instead.

This method should only be called while there are no **OGRFeature** (p. 94) objects in existance based on this **OGRFeatureDefn** (p. 109).

This method is the same as the C++ method **OGRFeatureDefn::DeleteFieldDefn()** (p. 111).

Parameters

hDefn	handle to the feature definition.
iField	the index of the field defintion.

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

References OGR_FD_DeleteFieldDefn().

Referenced by OGR_FD_DeleteFieldDefn().

12.12.2.63 int OGR_FD_Dereference (OGRFeatureDefnH *hDefn*)

Decrements the reference count by one.

This function is the same as the C++ method **OGRFeatureDefn::Dereference()** (p. 112).

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. 94) are based on.

Returns

the updated reference count.

References OGR_FD_Dereference().

Referenced by OGR_FD_Dereference().

12.12.2.64 void OGR_FD_Destroy (OGRFeatureDefnH *hDefn*)

Destroy a feature definition object and release all memory associated with it.

This function is the same as the C++ method OGRFeatureDefn::~OGRFeatureDefn().

Parameters

hDefn	handle to the feature definition to be destroyed.

References OGR_FD_Destroy().

Referenced by OGR_FD_Destroy().

12.12.2.65 int OGR_FD_GetFieldCount (OGRFeatureDefnH *hDefn*)

Fetch number of fields on the passed feature definition.

This function is the same as the C++ **OGRFeatureDefn::GetFieldCount()** (p. 112).

Parameters

hDefn	handle to the feature definition to get the fields count from.

Returns

count of fields.

References OGR_FD_GetFieldCount().

Referenced by OGR_FD_GetFieldCount().

12.12.2.66 OGRFieldDefnH OGR_FD_GetFieldDefn (OGRFeatureDefnH *hDefn,* int *iField*)

Fetch field definition of the passed feature definition.

This function is the same as the C++ method **OGRFeatureDefn::GetFieldDefn()** (p. 112).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

Parameters

hDefn	handle to the feature definition to get the field definition from.
iField	the field to fetch, between 0 and GetFieldCount()-1.

Returns

an handle to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

References OGR_FD_GetFieldDefn().

Referenced by OGR_FD_GetFieldDefn().

12.12.2.67 int OGR_FD_GetFieldIndex (OGRFeatureDefnH *hDefn*, const char * *pszFieldName*)

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This function is the same as the C++ method **OGRFeatureDefn::GetFieldIndex** (p. 112).

Parameters

hDefn	handle to the feature definition to get field index from.
pszFieldName	the field name to search for.

Returns

the field index, or -1 if no match found.

References OGR_FD_GetFieldIndex().

Referenced by OGR_FD_GetFieldIndex().

12.12.2.68 OGRwkbGeometryType OGR_FD_GetGeomType (OGRFeatureDefnH *hDefn*)

Fetch the geometry base type of the passed feature definition.

This function is the same as the C++ method **OGRFeatureDefn::GetGeomType()** (p. 113).

Parameters

hDefn	handle to the feature definition to get the geometry type from.

Returns

the base type for all geometry related to this definition.

References OGR_FD_GetGeomType().

Referenced by OGR_FD_GetGeomType().

12.12.2.69 const char* OGR_FD_GetName (OGRFeatureDefnH *hDefn*)

Get name of the **OGRFeatureDefn** (p. 109) passed as an argument.

This function is the same as the C++ method **OGRFeatureDefn::GetName()** (p. 113).

Parameters

hDefn	handle to the feature definition to get the name from.

Returns

the name. This name is internal and should not be modified, or freed.

References OGR_FD_GetName().

Referenced by OGR_FD_GetName().

12.12.2.70 int OGR_FD_GetReferenceCount (OGRFeatureDefnH *hDefn*)

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. 113).

Parameters

hDefn	hanlde to the feature definition on witch **OGRFeature** (p. 94) are based on.

Returns

the current reference count.

References OGR_FD_GetReferenceCount().

Referenced by OGR_FD_GetReferenceCount().

12.12.2.71 int OGR_FD_IsGeometryIgnored (OGRFeatureDefnH *hDefn*)

Determine whether the geometry can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::IsGeometryIgnored()** (p. 113).

Parameters

hDefn	hanlde to the feature definition on witch **OGRFeature** (p. 94) are based on.

Returns

ignore state

References OGR_FD_IsGeometryIgnored().

Referenced by OGR_FD_IsGeometryIgnored().

12.12.2.72 int OGR_FD_IsStyleIgnored (OGRFeatureDefnH *hDefn*)

Determine whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::IsStyleIgnored()** (p. 114).

Parameters

hDefn	handle to the feature definition on which **OGRFeature** (p. 94) are based on.

Returns

ignore state

References OGR_FD_IsStyleIgnored().

Referenced by OGR_FD_IsStyleIgnored().

12.12.2.73 int OGR_FD_Reference (OGRFeatureDefnH *hDefn*)

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. 94) objects referencing this definition.

This function is the same as the C++ method **OGRFeatureDefn::Reference()** (p. 114).

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. 94) are based on.

Returns

the updated reference count.

References OGR_FD_Reference().

Referenced by OGR_FD_Reference().

12.12.2.74 void OGR_FD_Release (OGRFeatureDefnH *hDefn*)

Drop a reference, and destroy if unreferenced.

This function is the same as the C++ method **OGRFeatureDefn::Release()** (p. 110).

Parameters

hDefn	handle to the feature definition to be released.

References OGR_FD_Release().

Referenced by OGR_FD_Release().

12.12.2.75 void OGR_FD_SetGeometryIgnored (OGRFeatureDefnH *hDefn,* int *bIgnore*)

Set whether the geometry can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::SetGeometryIgnored()** (p. 114).

Parameters

hDefn	hanlde to the feature definition on witch **OGRFeature** (p. 94) are based on.
bIgnore	ignore state

References OGR_FD_SetGeometryIgnored().

Referenced by OGR_FD_SetGeometryIgnored().

12.12.2.76 void OGR_FD_SetGeomType (OGRFeatureDefnH *hDefn,* OGRwkbGeometryType *eType*)

Assign the base geometry type for the passed layer (the same as the feature definition).

All geometry objects using this type must be of the defined type or a derived type. The default upon creation is wkbUnknown which allows for any geometry type. The geometry type should generally not be changed after any OGRFeatures have been created against this definition.

This function is the same as the C++ method **OGRFeatureDefn::SetGeomType()** (p. 115).

Parameters

hDefn	handle to the layer or feature definition to set the geometry type to.
eType	the new type to assign.

References OGR_FD_SetGeomType().

Referenced by OGR_FD_SetGeomType().

12.12.2.77 void OGR_FD_SetStyleIgnored (OGRFeatureDefnH *hDefn*, int *bIgnore*)

Set whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::SetStyleIgnored()** (p. 115).

Parameters

hDefn	hanlde to the feature definition on witch **OGRFeature** (p. 94) are based on.
bIgnore	ignore state

References OGR_FD_SetStyleIgnored().

Referenced by OGR_FD_SetStyleIgnored().

12.12.2.78 OGRFieldDefnH OGR_Fld_Create (const char * *pszName*, OGRFieldType *eType*)

Create a new field definition.

This function is the same as the CPP method **OGRFieldDefn::OGRFieldDefn()** (p. 117).

Parameters

pszName	the name of the new field definition.
eType	the type of the new field definition.

Returns

handle to the new field definition.

References OGR_Fld_Create().

Referenced by OGR_Fld_Create().

12.12.2.79 void OGR_Fld_Destroy (OGRFieldDefnH *hDefn*)

Destroy a field definition.

Parameters

hDefn	handle to the field definition to destroy.

References OGR_Fld_Destroy().

Referenced by OGR_Fld_Destroy().

12.12.2.80 OGRJustification OGR_Fld_GetJustify (OGRFieldDefnH *hDefn*)

Get the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::GetJustify()** (p. 118).

Parameters

hDefn	handle to the field definition to get justification from.

Returns

the justification.

References OGR_Fld_GetJustify().

Referenced by OGR_Fld_GetJustify().

12.12.2.81 const char* OGR_Fld_GetNameRef (OGRFieldDefnH *hDefn*)

Fetch name of this field.

This function is the same as the CPP method **OGRFieldDefn::GetNameRef()** (p. 118).

Parameters

hDefn	handle to the field definition.

Returns

the name of the field definition.

References OGR_Fld_GetNameRef().

Referenced by OGR_Fld_GetNameRef().

12.12.2.82 int OGR_Fld_GetPrecision (OGRFieldDefnH *hDefn*)

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::GetPrecision()** (p. 118).

Parameters

hDefn	handle to the field definition to get precision from.

Returns

the precision.

References OGR_Fld_GetPrecision().

Referenced by OGR_Fld_GetPrecision().

12.12.2.83 OGRFieldType OGR_Fld_GetType (OGRFieldDefnH *hDefn*)

Fetch type of this field.

This function is the same as the CPP method **OGRFieldDefn::GetType()** (p. 118).

Parameters

hDefn	handle to the field definition to get type from.

Returns

field type.

References OGR_Fld_GetType().

Referenced by OGR_Fld_GetType().

12.12.2.84 int OGR_Fld_GetWidth (OGRFieldDefnH *hDefn*)

Get the formatting width for this field.

This function is the same as the CPP method **OGRFieldDefn::GetWidth()** (p. 119).

Parameters

hDefn	handle to the field definition to get width from.

Returns

the width, zero means no specified width.

References OGR_Fld_GetWidth().

Referenced by OGR_Fld_GetWidth().

12.12.2.85 int OGR_Fld_IsIgnored (OGRFieldDefnH *hDefn*)

Return whether this field should be omitted when fetching features.

This method is the same as the C++ method **OGRFieldDefn::IsIgnored()** (p. 119).

Parameters

hDefn	handle to the field definition

Returns

ignore state

References OGR_Fld_IsIgnored().

Referenced by OGR_Fld_IsIgnored().

12.12.2.86 void OGR_Fld_Set (OGRFieldDefnH *hDefn*, const char * *pszNameIn*, OGRFieldType *eTypeIn*, int *nWidthIn*, int *nPrecisionIn*, OGRJustification *eJustifyIn*)

Set defining parameters for a field in one call.

This function is the same as the CPP method **OGRFieldDefn::Set()** (p. 119).

Parameters

hDefn	handle to the field definition to set to.
pszNameIn	the new name to assign.
eTypeIn	the new type (one of the OFT values like OFTInteger).
nWidthIn	the preferred formatting width. Defaults to zero indicating undefined.
nPrecisionIn	number of decimals places for formatting, defaults to zero indicating undefined.
eJustifyIn	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References OGR_Fld_Set().

Referenced by OGR_Fld_Set().

12.12.2.87 void OGR_Fld_SetIgnored (OGRFieldDefnH *hDefn*, int *ignore*)

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGRFieldDefn::SetIgnored()** (p. 120).

Parameters

hDefn	handle to the field definition
ignore	ignore state

References OGR_Fld_SetIgnored().

Referenced by OGR_Fld_SetIgnored().

12.12.2.88 void OGR_Fld_SetJustify (OGRFieldDefnH *hDefn*, OGRJustification *eJustify*)

Set the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::SetJustify()** (p. 120).

Parameters

hDefn	handle to the field definition to set justification to.
eJustify	the new justification.

References OGR_Fld_SetJustify().

Referenced by OGR_Fld_SetJustify().

12.12.2.89 void OGR_Fld_SetName (OGRFieldDefnH *hDefn*, const char * *pszName*)

Reset the name of this field.

This function is the same as the CPP method **OGRFieldDefn::SetName()** (p. 120).

Parameters

hDefn	handle to the field definition to apply the new name to.
pszName	the new name to apply.

References OGR_Fld_SetName().

Referenced by OGR_Fld_SetName().

12.12.2.90 void OGR_Fld_SetPrecision (OGRFieldDefnH *hDefn*, int *nPrecision*)

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::SetPrecision()** (p. 120).

Parameters

hDefn	handle to the field definition to set precision to.
nPrecision	the new precision.

References OGR_Fld_SetPrecision().

Referenced by OGR_Fld_SetPrecision().

12.12.2.91 void OGR_Fld_SetType (OGRFieldDefnH *hDefn,* OGRFieldType *eType*)

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 116) that is already part of an **OGRFeatureDefn** (p. 109).

This function is the same as the CPP method **OGRFieldDefn::SetType()** (p. 120).

Parameters

hDefn	handle to the field definition to set type to.
eType	the new field type.

References OGR_Fld_SetType().

Referenced by OGR_Fld_SetType().

12.12.2.92 void OGR_Fld_SetWidth (OGRFieldDefnH *hDefn,* int *nNewWidth*)

Set the formatting width for this field in characters.

This function is the same as the CPP method **OGRFieldDefn::SetWidth()** (p. 121).

Parameters

hDefn	handle to the field definition to set width to.
nNewWidth	the new width.

References OGR_Fld_SetWidth().

Referenced by OGR_Fld_SetWidth().

12.12.2.93 OGRErr OGR_G_AddGeometry (OGRGeometryH *hGeom,* OGRGeometryH *hNewSubGeom*)

Add a geometry to a geometry container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This function is the same as the CPP method **OGRGeometryCollection::addGeometry** (p. 147).

For a polygon, hNewSubGeom must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

Parameters

hGeom	existing geometry container.
hNewSubGeom	geometry to add to the container.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of existing geometry.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

12.12.2.94 OGRErr OGR_G_AddGeometryDirectly (OGRGeometryH *hGeom,* OGRGeometryH *hNewSubGeom*)

Add a geometry directly to an existing geometry container.

Some subclasses of **OGRGeometryCollection** (p. 146) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as addGeometry() does.

This function is the same as the CPP method **OGRGeometryCollection::addGeometryDirectly** (p. 148).

There is no SFCOM analog to this method.

For a polygon, hNewSubGeom must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

Parameters

hGeom	existing geometry.
hNewSubGeom	geometry to add to the existing geometry.

Returns

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

12.12.2.95 void OGR_G_AddPoint (OGRGeometryH *hGeom,* double *dfX,* double *dfY,* double *dfZ*)

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

Parameters

hGeom	handle to the geometry to add a point to.
dfX	x coordinate of point to add.
dfY	y coordinate of point to add.
dfZ	z coordinate of point to add.

References wkbLineString, and wkbPoint.

12.12.2.96 void OGR_G_AddPoint_2D (OGRGeometryH *hGeom,* double *dfX,* double *dfY*)

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

Parameters

hGeom	handle to the geometry to add a point to.
dfX	x coordinate of point to add.
dfY	y coordinate of point to add.

References wkbLineString, and wkbPoint.

12.12.2.97 OGRGeometryH OGR_G_ApproximateArcAngles (double *dfCenterX,* double *dfCenterY,* double *dfZ,* double *dfPrimaryRadius,* double *dfSecondaryRadius,* double *dfRotation,* double *dfStartAngle,* double *dfEndAngle,* double *dfMaxAngleStepSizeDegrees*)

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in

degrees.

If the dfMaxAngleStepSizeDegrees is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR_ARC_STEPSIZE configuration variable.

See Also

CPLSetConfigOption() (p. 330)

Parameters

dfCenterX	center X
dfCenterY	center Y
dfZ	center Z
dfPrimaryRadius	X radius of ellipse.
dfSecondary-Radius	Y radius of ellipse.
dfRotation	rotation of the ellipse clockwise.
dfStartAngle	angle to first point on arc (clockwise of X-positive)
dfEndAngle	angle to last point on arc (clockwise of X-positive)
dfMaxAngle-StepSize-Degrees	the largest step in degrees along the arc, zero to use the default setting.

Returns

OGRLineString (p. 181) geometry representing an approximation of the arc.

Since

OGR 1.8.0

References OGRGeometryFactory::approximateArcAngles(), and OGR_G_ApproximateArcAngles().

Referenced by OGR_G_ApproximateArcAngles().

12.12.2.98 double OGR_G_Area (OGRGeometryH *hGeom*)

Compute geometry area.

Computes the area for an **OGRLinearRing** (p. 178), **OGRPolygon** (p. 211) or **OGRMultiPolygon** (p. 200). Undefined for all other geometry types (returns zero).

This function utilizes the C++ get_Area() methods such as **OGRPolygon::get_Area()** (p. 215).

Parameters

hGeom	the geometry to operate on.

Returns

the area or 0.0 for unsupported geometry types.

Since

OGR 1.8.0

References wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiPolygon, and wkbPolygon.

12.12.2.99 void OGR_G_AssignSpatialReference (OGRGeometryH *hGeom,* OGRSpatialReferenceH *hSRS*)

Assign spatial reference to this object.

Any existing spatial reference is replaced, but under no circumstances does this result in the object being reprojected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. 230), but does not copy it.

This is similar to the SFCOM IGeometry::put_SpatialReference() method.

This function is the same as the CPP method **OGRGeometry::assignSpatialReference** (p. 129).

Parameters

hGeom	handle on the geometry to apply the new spatial reference system.
hSRS	handle on the new spatial reference system to apply.

References OGR_G_AssignSpatialReference().

Referenced by OGR_G_AssignSpatialReference().

12.12.2.100 OGRGeometryH OGR_G_Boundary (OGRGeometryH *hTarget*)

Compute boundary.

A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.

This function is the same as the C++ method **OGR_G_Boundary()** (p. 429).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hTarget	The Geometry to calculate the boundary of.

Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

Since

OGR 1.8.0

References OGR_G_Boundary().

Referenced by OGR_G_Boundary().

12.12.2.101 OGRGeometryH OGR_G_Buffer (OGRGeometryH *hTarget,* double *dfDist,* int *nQuadSegs*)

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The nQuadSegs parameter can be used to control how many segements should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This function is the same as the C++ method **OGRGeometry::Buffer()** (p. 129).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hTarget	the geometry.
dfDist	the buffer distance to be applied.
nQuadSegs	the number of segments used to approximate a 90 degree (quadrant) of curvature.

Returns

the newly created geometry, or NULL if an error occurs.

References OGR_G_Buffer().

Referenced by OGR_G_Buffer().

12.12.2.102 int OGR_G_Centroid (OGRGeometryH *hGeom,* OGRGeometryH *hCentroidPoint*)

Compute the geometry centroid.

The centroid location is applied to the passed in **OGRPoint** (p. 203) object. The centroid is not necessarily within the geometry.

This method relates to the SFCOM ISurface::get_Centroid() method however the current implementation based on GEOS can operate on other geometry types such as multipoint, linestring, geometrycollection such as multipolygons. OGC SF SQL 1.1 defines the operation for surfaces (polygons). SQL/MM-Part 3 defines the operation for surfaces and multisurfaces (multipolygons).

This function is the same as the C++ method **OGRGeometry::Centroid()** (p. 130).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Returns

OGRERR_NONE on success or OGRERR_FAILURE on error.

References OGRGeometry::Centroid(), OGRPoint::getGeometryType(), OGR_G_Centroid(), and wkbPoint.

Referenced by OGR_G_Centroid().

12.12.2.103 OGRGeometryH OGR_G_Clone (OGRGeometryH *hGeom*)

Make a copy of this object.

This function relates to the SFCOM IGeometry::clone() method.

This function is the same as the CPP method **OGRGeometry::clone()** (p. 130).

Parameters

hGeom	handle on the geometry to clone from.

Returns

an handle on the copy of the geometry with the spatial reference system as the original.

References OGR_G_Clone().

Referenced by OGR_G_Clone().

12.12.2.104 void OGR_G_CloseRings (OGRGeometryH *hGeom*)

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Parameters

hGeom	handle to the geometry.

References OGR_G_CloseRings().

Referenced by OGR_G_CloseRings().

12.12.2.105 int OGR_G_Contains (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for containment.

Tests if this geometry contains the other geometry.

This function is the same as the C++ method **OGRGeometry::Contains()** (p. 131).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if hThis contains hOther geometry, otherwise FALSE.

References OGR_G_Contains().

Referenced by OGR_G_Contains().

12.12.2.106 OGRGeometryH OGR_G_ConvexHull (OGRGeometryH *hTarget*)

Compute convex hull.

A new geometry object is created and returned containing the convex hull of the geometry on which the method is invoked.

This function is the same as the C++ method **OGRGeometry::ConvexHull()** (p. 131).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hTarget	The Geometry to calculate the convex hull of.

Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

References OGR_G_ConvexHull().

Referenced by OGR_G_ConvexHull().

12.12.2.107 OGRGeometryH OGR_G_CreateFromGML (const char * *pszGML*)

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGRGeometry** (p. 126). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The following GML2 elements are parsed : Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, MultiGeometry.

(OGR >= 1.8.0) The following GML3 elements are parsed : Surface, MultiSurface, PolygonPatch, Triangle, Rectangle, Curve, MultiCurve, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the OGR_ARC_STEPSIZE configuration variable.

The C++ method **OGRGeometryFactory::createFromGML()** (p. 158) is the same as this function.

Parameters

pszGML	The GML fragment for the geometry.

Returns

a geometry on succes, or NULL on error.

12.12.2.108 OGRErr OGR_G_CreateFromWkb (unsigned char * *pabyData,* OGRSpatialReferenceH *hSRS,* OGRGeometryH * *phGeometry,* int *nBytes*)

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGR_G_WkbSize()** (p. 456) on the returned geometry to establish the number of bytes it required in WKB format.

The **OGRGeometryFactory::createFromWkb()** (p. 158) CPP method is the same as this function.

Parameters

pabyData	pointer to the input BLOB data.
hSRS	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
phGeometry	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL in case of failure.
nBytes	the number of bytes of data available in pabyData, or -1 if it is not known, but assumed to be sufficient.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometryFactory::createFromWkb(), and OGR_G_CreateFromWkb().

Referenced by OGR_G_CreateFromWkb().

12.12.2.109 OGRErr OGR_G_CreateFromWkt (char ** *ppszData,* OGRSpatialReferenceH *hSRS,* OGRGeometryH * *phGeometry*)

Create a geometry object of the appropriate type from it's well known text representation.

The **OGRGeometryFactory::createFromWkt** (p. 159) CPP method is the same as this function.

Parameters

ppszData	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
hSRS	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
phGeometry	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL if the method fails.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometryFactory::createFromWkt(), and OGR_G_CreateFromWkt().

Referenced by OGR_G_CreateFromWkt().

12.12.2.110 OGRGeometryH OGR_G_CreateGeometry (OGRwkbGeometryType *eGeometryType*)

Create an empty geometry of desired type.

This is equivelent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::createGeometry** (p. 159).

Parameters

eGeometryType	the type code of the geometry to be created.

Returns

handle to the newly create geometry or NULL on failure.

References OGRGeometryFactory::createGeometry(), and OGR_G_CreateGeometry().

Referenced by OGR_G_CreateGeometry().

12.12.2.111 int OGR_G_Crosses (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for crossing.

Tests if this geometry and the other geometry are crossing.

This function is the same as the C++ method **OGRGeometry::Crosses()** (p. 131).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are crossing, otherwise FALSE.

References OGR_G_Crosses().

Referenced by OGR_G_Crosses().

12.12.2.112 void OGR_G_DestroyGeometry (OGRGeometryH *hGeom*)

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::destroyGeometry** (p. 160).

Parameters

hGeom	handle to the geometry to delete.

References OGRGeometryFactory::destroyGeometry(), and OGR_G_DestroyGeometry().

Referenced by OGR_G_DestroyGeometry().

12.12.2.113 OGRGeometryH OGR_G_Difference (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the other geometry removed.

This function is the same as the C++ method **OGRGeometry::Difference()** (p. 131).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

References OGR_G_Difference().

Referenced by OGR_G_Difference().

12.12.2.114 int OGR_G_Disjoint (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for disjointness.

Tests if this geometry and the other geometry are disjoint.

This function is the same as the C++ method **OGRGeometry::Disjoint()** (p. 132).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are disjoint, otherwise FALSE.

References OGR_G_Disjoint().

Referenced by OGR_G_Disjoint().

12.12.2.115 double OGR_G_Distance (OGRGeometryH *hFirst,* OGRGeometryH *hOther*)

Compute distance between two geometries.

Returns the shortest distance between the two geometries.

This function is the same as the C++ method **OGRGeometry::Distance()** (p. 132).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hFirst	the first geometry to compare against.
hOther	the other geometry to compare against.

Returns

the distance between the geometries or -1 if an error occurs.

References OGR_G_Distance().

Referenced by OGR_G_Distance().

12.12.2.116 void OGR_G_DumpReadable (OGRGeometryH *hGeom,* FILE * *fp,* const char * *pszPrefix*)

Dump geometry in well known text format to indicated output file.

This method is the same as the CPP method **OGRGeometry::dumpReadable** (p. 132).

Parameters

hGeom	handle on the geometry to dump.
fp	the text file to write the geometry to.
pszPrefix	the prefix to put on each line of output.

References OGR_G_DumpReadable().

Referenced by OGR_G_DumpReadable().

12.12.2.117 void OGR_G_Empty (OGRGeometryH *hGeom*)

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This function relates to the SFCOM IGeometry::Empty() method.

This function is the same as the CPP method **OGRGeometry::empty()** (p. 133).

Parameters

hGeom	handle on the geometry to empty.

References OGR_G_Empty().

Referenced by OGR_G_Empty().

12.12.2.118 int OGR_G_Equals (OGRGeometryH *hGeom,* OGRGeometryH *hOther*)

Returns TRUE if two geometries are equivalent.

This function is the same as the CPP method **OGRGeometry::Equals()** (p. 133) method.

Parameters

hGeom	handle on the first geometry.
hOther	handle on the other geometry to test against.

Returns

TRUE if equivalent or FALSE otherwise.

References OGR_G_Equals().

Referenced by OGR_G_Equals().

12.12.2.119 char* OGR_G_ExportToGML (OGRGeometryH *hGeometry*)

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. 133).

Parameters

hGeometry	handle to the geometry.

Returns

A GML fragment or NULL in case of error.

12.12.2.120 char* OGR_G_ExportToGMLEx (OGRGeometryH *hGeometry,* char ** *papszOptions*)

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

The supported options in OGR 1.8.0 are :

- FORMAT=GML3. Otherwise it will default to GML 2.1.2 output.
- GML3_LINESTRING_ELEMENT=curve. (Only valid for FORMAT=GML3) To use gml:Curve element for linestrings. Otherwise gml:LineString will be used .
- GML3_LONGSRS=YES/NO. (Only valid for FORMAT=GML3) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.

- GMLID=astring. If specified, a gml:id attribute will be written in the top-level geometry element with the provided value. Required for GML 3.2 compatibility.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. 133).

Parameters

hGeometry	handle to the geometry.
papszOptions	NULL-terminated list of options.

Returns

A GML fragment or NULL in case of error.

Since

OGR 1.8.0

12.12.2.121 char* OGR_G_ExportToJson (OGRGeometryH *hGeometry*)

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. 134).

Parameters

hGeometry	handle to the geometry.

Returns

A GeoJSON fragment or NULL in case of error.

References OGR_G_ExportToJson(), and OGR_G_ExportToJsonEx().

Referenced by OGRGeometry::exportToJson(), and OGR_G_ExportToJson().

12.12.2.122 char* OGR_G_ExportToJsonEx (OGRGeometryH *hGeometry*, char ** *papszOptions*)

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. 134).

Parameters

hGeometry	handle to the geometry.
papszOptions	a null terminated list of options. For now, only COORDINATE_PRECISION=int_number where int_number is the maximum number of figures after decimal separator to write in coordinates.

Returns

A GeoJSON fragment or NULL in case of error.

Since

OGR 1.9.0

References OGR_G_ExportToJsonEx().

Referenced by OGR_G_ExportToJson(), and OGR_G_ExportToJsonEx().

12.12.2.123 char* OGR_G_ExportToKML (OGRGeometryH *hGeometry*, const char * *pszAltitudeMode*)

Convert a geometry into KML format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToKML()** (p. 134).

Parameters

hGeometry	handle to the geometry.
pszAltitudeMode	value to write in altitudeMode element, or NULL.

Returns

A KML fragment or NULL in case of error.

References OGR_G_ExportToKML().

Referenced by OGRGeometry::exportToKML(), and OGR_G_ExportToKML().

12.12.2.124 OGRErr OGR_G_ExportToWkb (OGRGeometryH *hGeom*, OGRwkbByteOrder *eOrder*, unsigned char * *pabyDstBuffer*)

Convert a geometry into well known binary format.

This function relates to the SFCOM IWks::ExportToWKB() method.

This function is the same as the CPP method **OGRGeometry::exportToWkb()** (p. 134).

Parameters

hGeom	handle on the geometry to convert to a well know binary data from.
eOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyDstBuffer	a buffer into which the binary representation is written. This buffer must be at least **OGR_G_WkbSize()** (p. 456) byte in size.

Returns

Currently OGRERR_NONE is always returned.

References OGR_G_ExportToWkb().

Referenced by OGR_G_ExportToWkb().

12.12.2.125 OGRErr OGR_G_ExportToWkt (OGRGeometryH *hGeom*, char ** *ppszSrcText*)

Convert a geometry into well known text format.

This function relates to the SFCOM IWks::ExportToWKT() method.

This function is the same as the CPP method **OGRGeometry::exportToWkt()** (p. 135).

Parameters

hGeom	handle on the geometry to convert to a text format from.
ppszSrcText	a text buffer is allocated by the program, and assigned to the passed pointer.

Returns

Currently OGRERR_NONE is always returned.

References OGR_G_ExportToWkt().

Referenced by OGR_G_ExportToWkt().

12.12.2.126 void OGR_G_FlattenTo2D (OGRGeometryH *hGeom*)

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This function is the same as the CPP method **OGRGeometry::flattenTo2D()** (p. 135).

Parameters

hGeom	handle on the geometry to convert.

References OGR_G_FlattenTo2D().

Referenced by OGR_G_FlattenTo2D().

12.12.2.127 OGRGeometryH OGR_G_ForceToMultiLineString (OGRGeometryH *hGeom*)

Convert to multilinestring.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiLineString()** (p. 160).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiLineString(), and OGR_G_ForceToMultiLineString().

Referenced by OGR_G_ForceToMultiLineString().

12.12.2.128 OGRGeometryH OGR_G_ForceToMultiPoint (OGRGeometryH *hGeom*)

Convert to multipoint.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPoint()** (p. 160).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPoint(), and OGR_G_ForceToMultiPoint().

Referenced by OGR_G_ForceToMultiPoint().

12.12.2.129 OGRGeometryH OGR_G_ForceToMultiPolygon (OGRGeometryH *hGeom*)

Convert to multipolygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPolygon()** (p. 161).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPolygon(), and OGR_G_ForceToMultiPolygon().

Referenced by OGR_G_ForceToMultiPolygon().

12.12.2.130 OGRGeometryH OGR_G_ForceToPolygon (OGRGeometryH *hGeom*)

Convert to polygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToPolygon()** (p. 161).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToPolygon(), and OGR_G_ForceToPolygon().

Referenced by OGR_G_ForceToPolygon().

12.12.2.131 double OGR_G_GetArea (OGRGeometryH *hGeom*)

Compute geometry area (deprecated)

Deprecated

See Also

OGR_G_Area() (p. 428)

12.12.2.132 OGRGeometryH OGR_G_GetBoundary (OGRGeometryH *hTarget*)

Compute boundary (deprecated)

Deprecated

See Also

OGR_G_Boundary() (p. 429)

References OGR_G_GetBoundary().

Referenced by OGR_G_GetBoundary().

12.12.2.133 int OGR_G_GetCoordinateDimension (OGRGeometryH *hGeom*)

Get the dimension of the coordinates in this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method.

This function is the same as the CPP method **OGRGeometry::getCoordinateDimension()** (p. 135).

Parameters

hGeom	handle on the geometry to get the dimension of the coordinates from.

Returns

in practice this will return 2 or 3. It can also return 0 in the case of an empty point.

References OGR_G_GetCoordinateDimension().

Referenced by OGR_G_GetCoordinateDimension().

12.12.2.134 int OGR_G_GetDimension (OGRGeometryH *hGeom*)

Get the dimension of this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the geometry, but does not indicate the dimension of the underlying space (as indicated by **OGR_G_GetCoordinate-Dimension()** (p. 441) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. 136).

Parameters

hGeom	handle on the geometry to get the dimension from.

Returns

0 for points, 1 for lines and 2 for surfaces.

References OGR_G_GetDimension().

Referenced by OGR_G_GetDimension().

12.12.2.135 void OGR_G_GetEnvelope (OGRGeometryH *hGeom*, **OGREnvelope** * *psEnvelope*)

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. 136).

Parameters

hGeom	handle of the geometry to get envelope from.
psEnvelope	the structure in which to place the results.

References OGR_G_GetEnvelope().

Referenced by OGR_G_GetEnvelope().

12.12.2.136 void OGR_G_GetEnvelope3D (OGRGeometryH *hGeom*, **OGREnvelope3D** * *psEnvelope*)

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. 136).

Parameters

hGeom	handle of the geometry to get envelope from.
psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

References OGR_G_GetEnvelope3D().

Referenced by OGR_G_GetEnvelope3D().

12.12.2.137 int OGR_G_GetGeometryCount (OGRGeometryH *hGeom*)

Fetch the number of elements in a geometry or number of geometries in container.

Only geometries of type wkbPolygon[25D], wkbMultiPoint[25D], wkbMultiLineString[25D], wkbMultiPolygon[25D] or wkbGeometryCollection[25D] may return a valid value. Other geometry types will silently return 0.

For a polygon, the returned number is the number of rings (exterior ring + interior rings).

Parameters

hGeom	single geometry or geometry container from which to get the number of elements.

Returns

the number of elements.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

12.12.2.138 const char* OGR_G_GetGeometryName (OGRGeometryH *hGeom*)

Fetch WKT name for geometry type.

There is no SFCOM analog to this function.

This function is the same as the CPP method **OGRGeometry::getGeometryName()** (p. 137).

Parameters

hGeom	handle on the geometry to get name from.

Returns

name used for this geometry type in well known text format.

References OGR_G_GetGeometryName().

Referenced by OGR_G_GetGeometryName().

12.12.2.139 OGRGeometryH OGR_G_GetGeometryRef (OGRGeometryH *hGeom,* int *iSubGeom*)

Fetch geometry from a geometry container.

This function returns an handle to a geometry within the container. The returned geometry remains owned by the container, and should not be modified. The handle is only valid untill the next change to the geometry container. Use **OGR_G_Clone()** (p. 430) to make a copy.

This function relates to the SFCOM IGeometryCollection::get_Geometry() method.

This function is the same as the CPP method **OGRGeometryCollection::getGeometryRef()** (p. 152).

For a polygon, OGR_G_GetGeometryRef(iSubGeom) returns the exterior ring if iSubGeom == 0, and the interior rings for iSubGeom $>$ 0.

Parameters

hGeom	handle to the geometry container from which to get a geometry from.
iSubGeom	the index of the geometry to fetch, between 0 and getNumGeometries() - 1.

Returns

handle to the requested geometry.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

12.12.2.140 OGRwkbGeometryType OGR_G_GetGeometryType (OGRGeometryH *hGeom*)

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This function is the same as the CPP method **OGRGeometry::getGeometryType()** (p. 137).

Parameters

hGeom	handle on the geometry to get type from.

Returns

the geometry type code.

References OGR_G_GetGeometryType(), and wkbUnknown.

Referenced by OGR_G_GetGeometryType().

12.12.2.141 void OGR_G_GetPoint (OGRGeometryH *hGeom*, int *i*, double * *pdfX*, double * *pdfY*, double * *pdfZ*)

Fetch a point in line string or a point geometry.

Parameters

hGeom	handle to the geometry from which to get the coordinates.
i	the vertex to fetch, from 0 to getNumPoints()-1, zero for a point.
pdfX	value of x coordinate.
pdfY	value of y coordinate.
pdfZ	value of z coordinate.

References OGRLineString::getNumPoints(), OGRLineString::getX(), OGRLineString::getY(), OGRLineString::getZ(), wkbLineString, and wkbPoint.

12.12.2.142 int OGR_G_GetPointCount (OGRGeometryH *hGeom*)

Fetch number of points from a geometry.

Only wkbPoint[25D] or wkbLineString[25D] may return a valid value. Other geometry types will silently return 0.

Parameters

hGeom	handle to the geometry from which to get the number of points.

Returns

the number of points.

References OGRLineString::getNumPoints(), wkbLineString, and wkbPoint.

12.12.2.143 int OGR_G_GetPoints (OGRGeometryH *hGeom*, void * *pabyX*, int *nXStride*, void * *pabyY*, int *nYStride*, void * *pabyZ*, int *nZStride*)

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutives elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

Parameters

hGeom	handle to the geometry from which to get the coordinates.
pabyX	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
nXStride	the number of bytes between 2 elements of pabyX.
pabyY	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
nYStride	the number of bytes between 2 elements of pabyY.
pabyZ	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
nZStride	the number of bytes between 2 elements of pabyZ.

Returns

the number of points

Since

OGR 1.9.0

References OGRLineString::getNumPoints(), OGRLineString::getPoints(), wkbLineString, and wkbPoint.

12.12.2.144 OGRSpatialReferenceH OGR_G_GetSpatialReference (OGRGeometryH *hGeom*)

Returns spatial reference system for geometry.

This function relates to the SFCOM IGeometry::get_SpatialReference() method.

This function is the same as the CPP method **OGRGeometry::getSpatialReference()** (p. 137).

Parameters

hGeom	handle on the geometry to get spatial reference from.

Returns

a reference to the spatial reference geometry.

References OGR_G_GetSpatialReference().

Referenced by OGR_G_GetSpatialReference().

12.12.2.145 double OGR_G_GetX (OGRGeometryH *hGeom*, int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the x coordinate.
i	point to get the x coordinate.

Returns

the X coordinate of this point.

References OGRLineString::getNumPoints(), OGRLineString::getX(), wkbLineString, and wkbPoint.

12.12.2.146 double OGR_G_GetY (OGRGeometryH *hGeom*, int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the y coordinate.
i	point to get the Y coordinate.

Returns

the Y coordinate of this point.

References OGRLineString::getNumPoints(), OGRLineString::getY(), wkbLineString, and wkbPoint.

12.12.2.147 double OGR_G_GetZ (OGRGeometryH *hGeom*, int *i*)

Fetch the z coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the Z coordinate.
i	point to get the Z coordinate.

Returns

the Z coordinate of this point.

References OGRLineString::getNumPoints(), OGRLineString::getZ(), wkbLineString, and wkbPoint.

12.12.2.148 OGRErr OGR_G_ImportFromWkb (OGRGeometryH *hGeom*, unsigned char * *pabyData*, int *nSize*)

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type.

This function relates to the SFCOM IWks::ImportFromWKB() method.

This function is the same as the CPP method **OGRGeometry::importFromWkb()** (p. 138).

Parameters

hGeom	handle on the geometry to assign the well know binary data to.
pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGR_G_ImportFromWkb().

Referenced by OGR_G_ImportFromWkb().

12.12.2.149 OGRErr OGR_G_ImportFromWkt (OGRGeometryH *hGeom*, char ** *ppszSrcText*)

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type.

This function relates to the SFCOM IWks::ImportFromWKT() method.

This function is the same as the CPP method **OGRGeometry::importFromWkt()** (p. 138).

Parameters

hGeom	handle on the geometry to assign well know text data to.
ppszSrcText	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGR_G_ImportFromWkt().

Referenced by OGR_G_ImportFromWkt().

12.12.2.150 OGRGeometryH OGR_G_Intersection (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **OGR_G_Intersects()** (p. 447) function can be used to test if two geometries intersect.

This function is the same as the C++ method **OGRGeometry::Intersection()** (p. 139).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

References OGR_G_Intersection().

Referenced by OGR_G_Intersection().

12.12.2.151 int OGR_G_Intersects (OGRGeometryH *hGeom,* OGRGeometryH *hOtherGeom*)

Do these features intersect?

Currently this is not implemented in a rigerous fashion, and generally just tests whether the envelopes of the two features intersect. Eventually this will be made rigerous.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. 139).

Parameters

hGeom	handle on the first geometry.
hOtherGeom	handle on the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

References OGR_G_Intersects().

Referenced by OGR_G_Intersects().

12.12.2.152 int OGR_G_IsEmpty (OGRGeometryH *hGeom*)

Test if the geometry is empty.

This method is the same as the CPP method **OGRGeometry::IsEmpty()** (p. 139).

Parameters

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsEmpty().

Referenced by OGR_G_IsEmpty().

12.12.2.153 int OGR_G_IsRing (OGRGeometryH *hGeom*)

Test if the geometry is a ring.

This function is the same as the C++ method **OGRGeometry::IsRing()** (p. 140).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

Parameters

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsRing().

Referenced by OGR_G_IsRing().

12.12.2.154 int OGR_G_IsSimple (OGRGeometryH *hGeom*)

Returns TRUE if the geometry is simple.

Returns TRUE if the geometry has no anomalous geometric points, such as self intersection or self tangency. The description of each instantiable geometric class will include the specific conditions that cause an instance of that class to be classified as not simple.

This function is the same as the c++ method **OGRGeometry::IsSimple()** (p. 140) method.

If OGR is built without the GEOS library, this function will always return FALSE.

Parameters

hGeom	The Geometry to test.

Returns

TRUE if object is simple, otherwise FALSE.

References OGR_G_IsSimple().

Referenced by OGR_G_IsSimple().

12.12.2.155 int OGR_G_IsValid (OGRGeometryH *hGeom*)

Test if the geometry is valid.

This function is the same as the C++ method **OGRGeometry::IsValid()** (p. 140).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

Parameters

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsValid().

Referenced by OGR_G_IsValid().

12.12.2.156 double OGR_G_Length (OGRGeometryH *hGeom*)

Compute length of a geometry.

Computes the area for **OGRCurve** (p. 83) or MultiCurve objects. Undefined for all other geometry types (returns zero).

This function utilizes the C++ get_Length() method.

Parameters

hGeom	the geometry to operate on.

Returns

the lenght or 0.0 for unsupported geometry types.

Since

OGR 1.8.0

References wkbGeometryCollection, wkbLinearRing, wkbLineString, and wkbMultiLineString.

12.12.2.157 int OGR_G_Overlaps (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for overlap.

Tests if this geometry and the other geometry overlap, that is their intersection has a non-zero area.

This function is the same as the C++ method **OGRGeometry::Overlaps()** (p. 140).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are overlapping, otherwise FALSE.

References OGR_G_Overlaps().

Referenced by OGR_G_Overlaps().

12.12.2.158 OGRGeometryH OGR_G_Polygonize (OGRGeometryH *hTarget*)

Polygonizes a set of sparse edges.

A new geometry object is created and returned containing a collection of reassembled Polygons: NULL will be returned if the input collection doesn't corresponds to a MultiLinestring, or when reassembling Edges into Polygons is impossible due to topogical inconsistencies.

This function is the same as the C++ method **OGRGeometry::Polygonize()** (p. 141).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hTarget	The Geometry to be polygonized.

Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

Since

OGR 1.9.0

References OGR_G_Polygonize().

Referenced by OGR_G_Polygonize().

12.12.2.159 OGRErr OGR_G_RemoveGeometry (OGRGeometryH *hGeom,* int *iGeom,* int *bDelete*)

Remove a geometry from an exiting geometry container.

Removing a geometry will cause the geometry count to drop by one, and all "higher" geometries will shuffle down one in index.

There is no SFCOM analog to this method.

This function is the same as the CPP method **OGRGeometryCollection::removeGeometry()** (p. 154).

Parameters

hGeom	the existing geometry to delete from.
iGeom	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
bDelete	if TRUE the geometry will be destroyed, otherwise it will not. The default is TRUE as the existing geometry is considered to own the geometries in it.

Returns

OGRERR_NONE if successful, or OGRERR_FAILURE if the index is out of range.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

12.12.2.160 void OGR_G_Segmentize (OGRGeometryH *hGeom*, double *dfMaxLength*)

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the CPP method **OGRGeometry::segmentize()** (p. 141).

Parameters

hGeom	handle on the geometry to segmentize
dfMaxLength	the maximum distance between 2 points after segmentization

References OGR_G_Segmentize().

Referenced by OGR_G_Segmentize().

12.12.2.161 void OGR_G_SetCoordinateDimension (OGRGeometryH *hGeom*, int *nNewDimension*)

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters

hGeom	handle on the geometry to set the dimension of the coordinates.
nNewDimension	New coordinate dimension value, either 2 or 3.

References OGR_G_SetCoordinateDimension().

Referenced by OGR_G_SetCoordinateDimension().

12.12.2.162 void OGR_G_SetPoint (OGRGeometryH *hGeom*, int *i*, double *dfX*, double *dfY*, double *dfZ*)

Set the location of a vertex in a point or linestring geometry.

If iPoint is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

Parameters

hGeom	handle to the geometry to add a vertex to.
i	the index of the vertex to assign (zero based) or zero for a point.
dfX	input X coordinate to assign.
dfY	input Y coordinate to assign.
dfZ	input Z coordinate to assign (defaults to zero).

References wkbLineString, and wkbPoint.

12.12.2.163 void OGR_G_SetPoint_2D (OGRGeometryH *hGeom*, int *i*, double *dfX*, double *dfY*)

Set the location of a vertex in a point or linestring geometry.

If iPoint is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

Parameters

hGeom	handle to the geometry to add a vertex to.
i	the index of the vertex to assign (zero based) or zero for a point.
dfX	input X coordinate to assign.
dfY	input Y coordinate to assign.

References wkbLineString, and wkbPoint.

12.12.2.164 OGRGeometryH OGR_G_Simplify (OGRGeometryH *hThis,* double *dTolerance*)

Compute a simplified geometry.

This function is the same as the C++ method **OGRGeometry::Simplify()** (p. 141).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.8.0

References OGR_G_Simplify().

Referenced by OGR_G_Simplify().

12.12.2.165 OGRGeometryH OGR_G_SimplifyPreserveTopology (OGRGeometryH *hThis,* double *dTolerance*)

Simplify the geometry while preserving topology.

This function is the same as the C++ method **OGRGeometry::SimplifyPreserveTopology()** (p. 142).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.9.0

References OGR_G_SimplifyPreserveTopology().

Referenced by OGR_G_SimplifyPreserveTopology().

12.12.2.166 OGRGeometryH OGR_G_SymDifference (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute symmetric difference.

Generates a new geometry which is the symmetric difference of this geometry and the other geometry.

This function is the same as the C++ method **OGRGeometry::SymmetricDifference()** (p. 143).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

Since

OGR 1.8.0

References OGR_G_SymDifference().

Referenced by OGR_G_SymDifference().

12.12.2.167 OGRGeometryH OGR_G_SymmetricDifference (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute symmetric difference (deprecated)

Deprecated

See Also

OGR_G_SymmetricDifference() (p. 453)

References OGR_G_SymmetricDifference().

Referenced by OGR_G_SymmetricDifference().

12.12.2.168 int OGR_G_Touches (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for touching.

Tests if this geometry and the other geometry are touching.

This function is the same as the C++ method **OGRGeometry::Touches()** (p. 143).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are touching, otherwise FALSE.

References OGR_G_Touches().

Referenced by OGR_G_Touches().

12.12.2.169 OGRErr OGR_G_Transform (OGRGeometryH *hGeom,* OGRCoordinateTransformationH *hTransform*)

Apply arbitrary coordinate transformation to geometry.

This function will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this function does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 81) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 230) of the **OGRCoordinateTransformation** (p. 81) will be assigned to the geometry.

This function is the same as the CPP method **OGRGeometry::transform** (p. 143).

Parameters

hGeom	handle on the geometry to apply the transform to.
hTransform	handle on the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

References OGR_G_Transform().

Referenced by OGR_G_Transform().

12.12.2.170 OGRErr OGR_G_TransformTo (OGRGeometryH *hGeom,* OGRSpatialReferenceH *hSRS*)

Transform geometry to new spatial reference system.

This function will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

This function will only work if the geometry already has an assigned spatial reference system, and if it is transformable to the target coordinate system.

Because this function requires internal creation and initialization of an **OGRCoordinateTransformation** (p. 81) object it is significantly more expensive to use this function to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 81) in advance, and call transform() with that transformation. This function exists primarily for convenience when only transforming a single geometry.

This function is the same as the CPP method **OGRGeometry::transformTo** (p. 144).

Parameters

hGeom	handle on the geometry to apply the transform to.
hSRS	handle on the spatial reference system to apply.

Returns

OGRERR_NONE on success, or an error code.

References OGR_G_TransformTo().

Referenced by OGR_G_TransformTo().

12.12.2.171 OGRGeometryH OGR_G_Union (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

This function is the same as the C++ method **OGRGeometry::Union()** (p. 144).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the union or NULL if an error occurs.

References OGR_G_Union().

Referenced by OGR_G_Union().

12.12.2.172 OGRGeometryH OGR_G_UnionCascaded (OGRGeometryH *hThis*)

Compute union using cascading.

This function is the same as the C++ method **OGRGeometry::UnionCascaded()** (p. 145).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry.

Returns

a new geometry representing the union or NULL if an error occurs.

References OGR_G_UnionCascaded().

Referenced by OGR_G_UnionCascaded().

12.12.2.173 int OGR_G_Within (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Test for containment.

Tests if this geometry is within the other geometry.

This function is the same as the C++ method **OGRGeometry::Within()** (p. 145).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if hThis is within hOther, otherwise FALSE.

References OGR_G_Within().

Referenced by OGR_G_Within().

12.12.2.174 int OGR_G_WkbSize (OGRGeometryH *hGeom*)

Returns size of related binary representation.

This function returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This function relates to the SFCOM IWks::WkbSize() method.

This function is the same as the CPP method **OGRGeometry::WkbSize()** (p. 145).

Parameters

hGeom	handle on the geometry to get the binary size from.

Returns

size of binary representation in bytes.

References OGR_G_WkbSize().

Referenced by OGR_G_WkbSize().

12.12.2.175 const char* OGR_GetFieldTypeName (OGRFieldType *eType*)

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldTypeName()** (p. 117).

Parameters

eType	the field type to get name for.

Returns

the name.

References OGRFieldDefn::GetFieldTypeName(), and OGR_GetFieldTypeName().

Referenced by OGR_GetFieldTypeName().

12.12.2.176 OGRErr OGR_L_AlterFieldDefn (OGRLayerH *hLayer,* int *iField,* OGRFieldDefnH *hNewFieldDefn,* int *nFlags*)

Alter the definition of an existing field on a layer.

You must use this to alter the definition of an existing field of a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This function should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the OLCAlterFieldDefn capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

This function is the same as the C++ method **OGRLayer::AlterFieldDefn()** (p. 164).

Parameters

hLayer	handle to the layer.
iField	index of the field whose definition must be altered.
hNewFieldDefn	new field definition
nFlags	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG and ALTER_WIDTH_PRECISION_FLAG to indicate which of the name and/or type and/or width and precision fields from the new field definition must be taken into account.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_L_AlterFieldDefn().

Referenced by OGR_L_AlterFieldDefn().

12.12.2.177 OGRErr OGR_L_CommitTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, CommitTransaction commits a transaction.

If no transaction is active, or the commit fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::CommitTransaction().

Parameters

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

References OGR_L_CommitTransaction().

Referenced by OGR_L_CommitTransaction().

12.12.2.178 OGRErr OGR_L_CreateFeature (OGRLayerH *hLayer,* OGRFeatureH *hFeat*)

Create and write a new feature within a layer.

The passed feature is written to the layer as a new feature, rather than overwriting an existing one. If the feature has a feature id other than OGRNullFID, then the native implementation may use that as the feature id of the new feature, but not necessarily. Upon successful return the passed feature will have been updated with the new feature id.

This function is the same as the C++ method **OGRLayer::CreateFeature()** (p. 165).

Parameters

hLayer	handle to the layer to write the feature to.
hFeat	the handle of the feature to write to disk.

Returns

OGRERR_NONE on success.

References OGR_L_CreateFeature().

Referenced by OGR_L_CreateFeature().

12.12.2.179 OGRErr OGR_L_CreateField (OGRLayerH *hLayer,* OGRFieldDefnH *hField,* int *bApproxOK*)

Create a new field on a layer.

You must use this to create new fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This function should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the OLCCreateField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::CreateField()** (p. 165).

Parameters

hLayer	handle to the layer to write the field definition.
hField	handle of the field definition to write to disk.
bApproxOK	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

Returns

OGRERR_NONE on success.

References OGR_L_CreateField().

Referenced by OGR_L_CreateField().

12.12.2.180 OGRErr OGR_L_DeleteFeature (OGRLayerH *hLayer,* long *nFID*)

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **OGR_L_TestCapability()** (p. 469) function may be called with OLCDeleteFeature to check if the driver supports feature deletion.

This method is the same as the C++ method **OGRLayer::DeleteFeature()** (p. 166).

Parameters

hLayer	handle to the layer
nFID	the feature id to be deleted from the layer

Returns

OGRERR_NONE on success.

References OGR_L_DeleteFeature().

Referenced by OGR_L_DeleteFeature().

12.12.2.181 OGRErr OGR_L_DeleteField (OGRLayerH *hLayer,* int *iField*)

Create a new field on a layer.

You must use this to delete existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the deleted field. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This function should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the OLCDeleteField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::DeleteField()** (p. 166).

Parameters

hLayer	handle to the layer.
iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_L_DeleteField().

Referenced by OGR_L_DeleteField().

12.12.2.182 OGRErr OGR_L_GetExtent (OGRLayerH *hLayer,* OGREnvelope * *psExtent,* int *bForce*)

Fetch the extent of this layer.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **OGR_L_GetExtent()** (p. 459) without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

This function is the same as the C++ method **OGRLayer::GetExtent()** (p. 167).

Parameters

hLayer	handle to the layer from which to get extent.
psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

References OGR_L_GetExtent().

Referenced by OGR_L_GetExtent().

12.12.2.183 OGRFeatureH OGR_L_GetFeature (OGRLayerH *hLayer,* long *nFeatureId*)

Fetch a feature by its identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters.

If this function returns a non-NULL feature, it is guaranteed that its feature id (**OGR_F_GetFID()** (p. 404)) will be the same as nFID.

Use OGR_L_TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **OGR_L_GetFeature()** (p. 460); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads are generally considered interrupted by a **OGR_L_GetFeature()** (p. 460) call.

The returned feature should be free with **OGR_F_Destroy()** (p. 403).

This function is the same as the C++ method **OGRLayer::GetFeature( )** (p. 167).

Parameters

hLayer	handle to the layer that owned the feature.
nFeatureId	the feature id of the feature to read.

Returns

an handle to a feature now owned by the caller, or NULL on failure.

References OGR_L_GetFeature().

Referenced by OGR_L_GetFeature().

12.12.2.184 int OGR_L_GetFeatureCount (OGRLayerH *hLayer,* int *bForce*)

Fetch the feature count in this layer.

Returns the number of features in the layer. For dynamic databases the count may not be exact. If bForce is FALSE, and it would be expensive to establish the feature count a value of -1 may be returned indicating that the count isn't know. If bForce is TRUE some implementations will actually scan the entire layer once to count objects.

The returned count takes the spatial filter into account.

Note that some implementations of this method may alter the read cursor of the layer.

This function is the same as the CPP **OGRLayer::GetFeatureCount()** (p. 168).

Parameters

hLayer	handle to the layer that owned the features.
bForce	Flag indicating whether the count should be computed even if it is expensive.

Returns

feature count, -1 if count not known.

References OGR_L_GetFeatureCount().

Referenced by OGR_L_GetFeatureCount().

12.12.2.185 const char * OGR_L_GetFIDColumn (OGRLayerH *hLayer*)

This method returns the name of the underlying database column being used as the FID column, or "" if not supported.

This method is the same as the C++ method **OGRLayer::GetFIDColumn()** (p. 168)

Parameters

hLayer	handle to the layer

Returns

fid column name.

References OGR_L_GetFIDColumn().

Referenced by OGR_L_GetFIDColumn().

12.12.2.186 const char * OGR_L_GetGeometryColumn (OGRLayerH *hLayer*)

This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.

This method is the same as the C++ method **OGRLayer::GetGeometryColumn()** (p. 168)

Parameters

hLayer	handle to the layer

Returns

geometry column name.

References OGR_L_GetGeometryColumn().

Referenced by OGR_L_GetGeometryColumn().

12.12.2.187 OGRwkbGeometryType OGR_L_GetGeomType (OGRLayerH *hLayer*)

Return the layer geometry type.

This returns the same result as OGR_FD_GetGeomType(OGR_L_GetLayerDefn(hLayer)), but for a few drivers, calling **OGR_L_GetGeomType()** (p. 461) directly can avoid lengthy layer definition initialization.

This function is the same as the C++ method **OGRLayer::GetGeomType()** (p. 168).

Parameters

hLayer	handle to the layer.

Returns

the geometry type

Since

OGR 1.8.0

References OGR_L_GetGeomType(), and wkbUnknown.

Referenced by OGR_L_GetGeomType().

12.12.2.188 OGRFeatureDefnH OGR_L_GetLayerDefn (OGRLayerH *hLayer*)

Fetch the schema information for this layer.

The returned handle to the **OGRFeatureDefn** (p. 109) is owned by the **OGRLayer** (p. 162), and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

This function is the same as the C++ method **OGRLayer::GetLayerDefn()** (p. 169).

Parameters

hLayer	handle to the layer to get the schema information.

Returns

an handle to the feature definition.

References OGR_L_GetLayerDefn().

Referenced by OGR_L_GetLayerDefn().

12.12.2.189 const char * OGR_L_GetName (OGRLayerH *hLayer*)

Return the layer name.

This returns the same content as OGR_FD_GetName(OGR_L_GetLayerDefn(hLayer)), but for a few drivers, calling **OGR_L_GetName()** (p. 462) directly can avoid lengthy layer definition initialization.

This function is the same as the C++ method **OGRLayer::GetName()** (p. 169).

Parameters

hLayer	handle to the layer.

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

References OGR_L_GetName().

Referenced by OGR_L_GetName().

12.12.2.190 OGRFeatureH OGR_L_GetNextFeature (OGRLayerH *hLayer*)

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity of the caller to delete with **OGR_F_Destroy()** (p. 403). It is critical that all features associated with an **OGRLayer** (p. 162) (more specifically an **OGRFeatureDefn** (p. 109)) be deleted before that layer/datasource is deleted.

Only features matching the current spatial filter (set with SetSpatialFilter()) will be returned.

This function implements sequential access to the features of a layer. The **OGR_L_ResetReading()** (p. 465) function can be used to start at the beginning again.

This function is the same as the C++ method **OGRLayer::GetNextFeature()** (p. 170).

Parameters

hLayer	handle to the layer from which feature are read.

Returns

an handle to a feature, or NULL if no more features are available.

References OGR_L_GetNextFeature().

Referenced by OGR_L_GetNextFeature().

12.12.2.191 OGRGeometryH OGR_L_GetSpatialFilter (OGRLayerH *hLayer*)

This function returns the current spatial filter for this layer.

The returned pointer is to an internally owned object, and should not be altered or deleted by the caller.

This function is the same as the C++ method **OGRLayer::GetSpatialFilter()** (p. 170).

Parameters

hLayer	handle to the layer to get the spatial filter from.

Returns

an handle to the spatial filter geometry.

References OGR_L_GetSpatialFilter().

Referenced by OGR_L_GetSpatialFilter().

12.12.2.192 OGRSpatialReferenceH OGR_L_GetSpatialRef (OGRLayerH *hLayer*)

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 162) and should not be modified or freed by the application.

This function is the same as the C++ method **OGRLayer::GetSpatialRef()** (p. 171).

Parameters

hLayer	handle to the layer to get the spatial reference from.

Returns

spatial reference, or NULL if there isn't one.

References OGR_L_GetSpatialRef().

Referenced by OGR_L_GetSpatialRef().

12.12.2.193 OGRErr OGR_L_ReorderField (OGRLayerH *hLayer,* int *iOldFieldPos,* int *iNewFieldPos*)

Reorder an existing field on a layer.

This function is a conveniency wrapper of **OGR_L_ReorderFields()** (p. 464) dedicated to move a single field.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This function should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

The field definition that was at initial position iOldFieldPos will be moved at position iNewFieldPos, and elements between will be shuffled accordingly.

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderField(1, 3) will reorder them as "0","2","3","1","4".

Not all drivers support this function. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::ReorderField()** (p. 171).

Parameters

hLayer	handle to the layer.
iOldFieldPos	previous position of the field to move. Must be in the range [0,GetFieldCount()-1].
iNewFieldPos	new position of the field to move. Must be in the range [0,GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_L_ReorderField().

Referenced by OGR_L_ReorderField().

12.12.2.194 OGRErr OGR_L_ReorderFields (OGRLayerH *hLayer,* int * *panMap*)

Reorder all the fields of a layer.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. 109) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. 109) used by a layer directly.

This function should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was panMap[i].

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderFields([0,2,3,1,4]) will reorder them as "0","2","3","1","4".

Not all drivers support this function. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::ReorderFields()** (p. 172).

Parameters

hLayer	handle to the layer.
panMap	an array of GetLayerDefn()->GetFieldCount() elements which is a permutation of [0, GetLayerDefn()->GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References OGR_L_ReorderFields().

Referenced by OGR_L_ReorderFields().

12.12.2.195 void OGR_L_ResetReading (OGRLayerH *hLayer*)

Reset feature reading to start on the first feature.

This affects GetNextFeature().

This function is the same as the C++ method **OGRLayer::ResetReading()** (p. 172).

Parameters

hLayer	handle to the layer on which features are read.

References OGR_L_ResetReading().

Referenced by OGR_L_ResetReading().

12.12.2.196 OGRErr OGR_L_RollbackTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction. If no transaction is active, or the rollback fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::RollbackTransaction().

Parameters

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

References OGR_L_RollbackTransaction().

Referenced by OGR_L_RollbackTransaction().

12.12.2.197 OGRErr OGR_L_SetAttributeFilter (OGRLayerH *hLayer,* const char * *pszQuery*)

Set a new attribute query.

This function sets the attribute query string to be used when fetching features via the **OGR_L_GetNextFeature()** (p. 462) function. Only features for which the query evaluates as true will be returned.

The query string should be in the format of an SQL WHERE clause. For instance "population > 1000000 and population < 5000000" where population is an attribute in the layer. The query format is a restricted form of SQL WHERE clause as defined "eq_format=restricted_where" about half way through this document:

`http://ogdi.sourceforge.net/prop/6.2.CapabilitiesMetadata.html`

Note that installing a query string will generally result in resetting the current reading position (ala **OGR_L_ResetReading()** (p. 465)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. 173).

Parameters

hLayer	handle to the layer on which attribute query will be executed.
pszQuery	query in restricted SQL WHERE format, or NULL to clear the current query.

Returns

OGRERR_NONE if successfully installed, or an error code if the query expression is in error, or some other failure occurs.

References OGR_L_SetAttributeFilter().

Referenced by OGR_L_SetAttributeFilter().

12.12.2.198 OGRErr OGR_L_SetFeature (OGRLayerH *hLayer,* OGRFeatureH *hFeat*)

Rewrite an existing feature.

This function will write a feature to the layer, based on the feature id within the **OGRFeature** (p. 94).

Use OGR_L_TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **OGR_L_SetFeature()** (p. 466).

This function is the same as the C++ method **OGRLayer::SetFeature()** (p. 173).

Parameters

hLayer	handle to the layer to write the feature.
hFeat	the feature to write.

Returns

OGRERR_NONE if the operation works, otherwise an appropriate error code.

References OGR_L_SetFeature().

Referenced by OGR_L_SetFeature().

12.12.2.199 OGRErr OGR_L_SetIgnoredFields (OGRLayerH , const char ** *papszFields*)

Set which fields can be omitted when retrieving features from the layer.

If the driver supports this functionality (testable using OLCIgnoreFields capability), it will not fetch the specified fields in subsequent calls to GetFeature() / GetNextFeature() and thus save some processing time and/or bandwidth.

Besides field names of the layers, the following special fields can be passed: "OGR_GEOMETRY" to ignore geometry and "OGR_STYLE" to ignore layer style.

By default, no fields are ignored.

This method is the same as the C++ method **OGRLayer::SetIgnoredFields()** (p. 173)

Parameters

papszFields	an array of field names terminated by NULL item. If NULL is passed, the ignored list is cleared.

Returns

OGRERR_NONE if all field names have been resolved (even if the driver does not support this method)

References OGR_L_SetIgnoredFields().

Referenced by OGR_L_SetIgnoredFields().

12.12.2.200 OGRErr OGR_L_SetNextByIndex (OGRLayerH *hLayer,* long *nIndex*)

Move read cursor to the nIndex'th feature in the current resultset.

This method allows positioning of a layer such that the GetNextFeature() call will read the requested feature, where nIndex is an absolute index into the current result set. So, setting it to 3 would mean the next feature read with GetNextFeature() would have been the 4th feature to have been read if sequential reading took place from the beginning of the layer, including accounting for spatial and attribute filters.

Only in rare circumstances is SetNextByIndex() efficiently implemented. In all other cases the default implementation which calls ResetReading() and then calls GetNextFeature() nIndex times is used. To determine if fast seeking is available on the current layer use the TestCapability() method with a value of OLCFastSetNextByIndex.

This method is the same as the C++ method **OGRLayer::SetNextByIndex()** (p. 174)

Parameters

hLayer	handle to the layer
nIndex	the index indicating how many steps into the result set to seek.

Returns

OGRERR_NONE on success or an error code.

References OGR_L_SetNextByIndex().

Referenced by OGR_L_SetNextByIndex().

12.12.2.201 void OGR_L_SetSpatialFilter (OGRLayerH *hLayer,* OGRGeometryH *hGeom*)

Set a new spatial filter.

This function set the geometry to be used as a spatial filter when fetching features via the **OGR_L_GetNextFeature()** (p. 462) function. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features who's envelope (as returned by **OGR_G_GetEnvelope()** (p. 442)) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

This function makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by **OGR_L_GetSpatialRef()** (p. 463)). In the future this may be generalized.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. 174).

Parameters

hLayer	handle to the layer on which to set the spatial filter.
hGeom	handle to the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

References OGR_L_SetSpatialFilter().

Referenced by OGR_L_SetSpatialFilter().

12.12.2.202 void OGR_L_SetSpatialFilterRect (OGRLayerH *hLayer,* double *dfMinX,* double *dfMinY,* double *dfMaxX,* double *dfMaxY*)

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **OGR_L_GetNextFeature()** (p. 462) method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as the layer as a whole (as returned by **OGRLayer::GetSpatialRef()** (p. 171)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. 174). It exists as a convenience.

The only way to clear a spatial filter set with this method is to call OGRLayer::SetSpatialFilter(NULL).

This method is the same as the C++ method **OGRLayer::SetSpatialFilterRect()** (p. 175).

Parameters

hLayer	handle to the layer on which to set the spatial filter.
dfMinX	the minimum X coordinate for the rectangular region.
dfMinY	the minimum Y coordinate for the rectangular region.
dfMaxX	the maximum X coordinate for the rectangular region.
dfMaxY	the maximum Y coordinate for the rectangular region.

References OGR_L_SetSpatialFilterRect().

Referenced by OGR_L_SetSpatialFilterRect().

12.12.2.203 OGRErr OGR_L_StartTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, StartTransaction creates a transaction.

If starting the transaction fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::StartTransaction().

Parameters

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

References OGR_L_StartTransaction().

Referenced by OGR_L_StartTransaction().

12.12.2.204 OGRErr OGR_L_SyncToDisk (OGRLayerH *hLayer*)

Flush pending changes to disk.

This call is intended to force the layer to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some layers do not implement this method, and will still return OGRERR_NONE. The default implementation just returns OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

In any event, you should always close any opened datasource with **OGR_DS_Destroy()** (p. 399) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRLayer::SyncToDisk()** (p. 176)

Parameters

hLayer	handle to the layer

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

References OGR_L_SyncToDisk().

Referenced by OGR_L_SyncToDisk().

12.12.2.205 int OGR_L_TestCapability (OGRLayerH *hLayer*, const char * *pszCap*)

Test if this layer supported the named capability.

The capability codes that can be tested are represented as strings, but #defined constants exists to ensure correct spelling. Specific layer types may implement class specific capabilities, but this can't generally be discovered by the caller.

- **OLCRandomRead** / "RandomRead": TRUE if the GetFeature() method is implemented in an optimized way for this layer, as opposed to the default implementation using ResetReading() and GetNextFeature() to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the CreateFeature() method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the SetFeature() method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 162) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCFastSpatialFilter** / "FastSpatialFilter": TRUE if this layer implements spatial filtering efficiently. Layers that effectively read all features, and test them with the **OGRFeature** (p. 94) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain its own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via **OGR_L_GetFeatureCount()** (p. 460)) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **OGR_L_GetExtent()** (p. 459)) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the SetNextByIndex() call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using CreateField(), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using DeleteField(), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using ReorderField() or ReorderFields(), otherwise FALSE.

- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using AlterFieldDefn(), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the DeleteFeature() method is supported on this layer, otherwise FALSE.
- **OLCStringsAsUTF8** / "StringsAsUTF8": TRUE if values of OFTString fields are assured to be in UTF-8 format. If FALSE the encoding of fields is uncertain, though it might still be UTF-8.
- **OLCTransactions** / "Transactions": TRUE if the StartTransaction(), CommitTransaction() and RollbackTransaction() methods work in a meaningful way, otherwise FALSE.

This function is the same as the C++ method **OGRLayer::TestCapability()** (p. 176).

Parameters

hLayer	handle to the layer to get the capability from.
pszCap	the name of the capability to test.

Returns

TRUE if the layer has the requested capability, or FALSE otherwise. OGRLayers will return FALSE for any unrecognised capabilities.

References OGR_L_TestCapability().

Referenced by OGR_L_TestCapability().

12.12.2.206 int OGR_SM_AddPart (OGRStyleMgrH *hSM,* OGRStyleToolH *hST*)

Add a part (style tool) to the current style.

This function is the same as the C++ method **OGRStyleMgr::AddPart()** (p. 285).

Parameters

hSM	handle to the style manager.
hST	the style tool defining the part to add.

Returns

TRUE on success, FALSE on errors.

References OGR_SM_AddPart().

Referenced by OGR_SM_AddPart().

12.12.2.207 int OGR_SM_AddStyle (OGRStyleMgrH *hSM,* const char * *pszStyleName,* const char * *pszStyleString*)

Add a style to the current style table.

This function is the same as the C++ method **OGRStyleMgr::AddStyle()** (p. 286).

Parameters

hSM	handle to the style manager.
pszStyleName	the name of the style to add.
pszStyleString	the style string to use, or NULL to use the style stored in the manager.

Returns

TRUE on success, FALSE on errors.

References OGR_SM_AddStyle().

Referenced by OGR_SM_AddStyle().

12.12.2.208 OGRStyleMgrH OGR_SM_Create (OGRStyleTableH *hStyleTable*)

OGRStyleMgr (p. 284) factory.

This function is the same as the C++ method **OGRStyleMgr::OGRStyleMgr()** (p. 285).

Parameters

hStyleTable	pointer to **OGRStyleTable** (p. 289) or NULL if not working with a style table.

Returns

an handle to the new style manager object.

References OGR_SM_Create().

Referenced by OGR_SM_Create().

12.12.2.209 void OGR_SM_Destroy (OGRStyleMgrH *hSM*)

Destroy Style Manager.

This function is the same as the C++ method **OGRStyleMgr::~OGRStyleMgr()** (p. 285).

Parameters

hSM	handle to the style manager to destroy.

References OGR_SM_Destroy().

Referenced by OGR_SM_Destroy().

12.12.2.210 OGRStyleToolH OGR_SM_GetPart (OGRStyleMgrH *hSM,* int *nPartId,* const char * *pszStyleString*)

Fetch a part (style tool) from the current style.

This function is the same as the C++ method **OGRStyleMgr::GetPart()** (p. 286).

This function instanciates a new object that should be freed with **OGR_ST_Destroy()** (p. 473).

Parameters

hSM	handle to the style manager.
nPartId	the part number (0-based index).
pszStyleString	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns

OGRStyleToolH of the requested part (style tools) or NULL on error.

References OGR_SM_GetPart().

Referenced by OGR_SM_GetPart().

12.12.2.211 int OGR_SM_GetPartCount (OGRStyleMgrH *hSM,* const char * *pszStyleString*)

Get the number of parts in a style.

This function is the same as the C++ method **OGRStyleMgr::GetPartCount()** (p. 286).

Parameters

hSM	handle to the style manager.
pszStyleString	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns

the number of parts (style tools) in the style.

References OGR_SM_GetPartCount().

Referenced by OGR_SM_GetPartCount().

12.12.2.212 const char* OGR_SM_InitFromFeature (OGRStyleMgrH *hSM,* OGRFeatureH *hFeat*)

Initialize style manager from the style string of a feature.

This function is the same as the C++ method **OGRStyleMgr::InitFromFeature()** (p. 287).

Parameters

hSM	handle to the style manager.
hFeat	handle to the new feature from which to read the style.

Returns

a reference to the style string read from the feature, or NULL in case of error.

References OGR_SM_InitFromFeature().

Referenced by OGR_SM_InitFromFeature().

12.12.2.213 int OGR_SM_InitStyleString (OGRStyleMgrH *hSM,* const char * *pszStyleString*)

Initialize style manager from the style string.

This function is the same as the C++ method **OGRStyleMgr::InitStyleString()** (p. 288).

Parameters

hSM	handle to the style manager.
pszStyleString	the style string to use (can be NULL).

Returns

TRUE on success, FALSE on errors.

References OGR_SM_InitStyleString().

Referenced by OGR_SM_InitStyleString().

12.12.2.214 OGRStyleToolH OGR_ST_Create (OGRSTClassId *eClassId*)

OGRStyleTool (p. 292) factory.

This function is a constructor for **OGRStyleTool** (p. 292) derived classes.

Parameters

eClassId	subclass of style tool to create. One of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4).

Returns

an handle to the new style tool object or NULL if the creation failed.

References OGR_ST_Create().

Referenced by OGR_ST_Create().

12.12.2.215 void OGR_ST_Destroy (OGRStyleToolH *hST*)

Destroy Style Tool.

Parameters

hST	handle to the style tool to destroy.

References OGR_ST_Destroy().

Referenced by OGR_ST_Destroy().

12.12.2.216 double OGR_ST_GetParamDbl (OGRStyleToolH *hST,* int *eParam,* int * *bValueIsNull*)

Get Style Tool parameter value as a double.

Maps to the **OGRStyleTool** (p. 292) subclasses' GetParamDbl() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
bValueIsNull	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns

the parameter value as double and sets bValueIsNull.

References OGR_ST_GetParamDbl().

Referenced by OGR_ST_GetParamDbl().

12.12.2.217 int OGR_ST_GetParamNum (OGRStyleToolH *hST,* int *eParam,* int * *bValueIsNull*)

Get Style Tool parameter value as an integer.

Maps to the **OGRStyleTool** (p. 292) subclasses' GetParamNum() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
bValueIsNull	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns

the parameter value as integer and sets bValueIsNull.

References OGR_ST_GetParamNum().

Referenced by OGR_ST_GetParamNum().

12.12.2.218 const char* OGR_ST_GetParamStr (OGRStyleToolH *hST,* int *eParam,* int * *bValueIsNull*)

Get Style Tool parameter value as string.

Maps to the **OGRStyleTool** (p. 292) subclasses' GetParamStr() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
bValueIsNull	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns

the parameter value as string and sets bValueIsNull.

References OGR_ST_GetParamStr().

Referenced by OGR_ST_GetParamStr().

12.12.2.219 int OGR_ST_GetRGBFromString (OGRStyleToolH *hST,* const char * *pszColor,* int * *pnRed,* int * *pnGreen,* int * *pnBlue,* int * *pnAlpha*)

Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.

Maps to OGRStyleTool::GetRGBFromString().

Parameters

hST	handle to the style tool.
pszColor	the color to parse
pnRed	pointer to an int in which the red value will be returned
pnGreen	pointer to an int in which the green value will be returned
pnBlue	pointer to an int in which the blue value will be returned
pnAlpha	pointer to an int in which the (optional) alpha value will be returned

Returns

TRUE if the color could be succesfully parsed, or FALSE in case of errors.

References OGR_ST_GetRGBFromString().

Referenced by OGR_ST_GetRGBFromString().

12.12.2.220 const char* OGR_ST_GetStyleString (OGRStyleToolH *hST*)

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. 292) subclasses' GetStyleString() methods.

Parameters

hST	handle to the style tool.

Returns

the style string for this style tool or "" if the hST is invalid.

References OGR_ST_GetStyleString().

Referenced by OGR_ST_GetStyleString().

12.12.2.221 OGRSTClassId OGR_ST_GetType (OGRStyleToolH *hST*)

Determine type of Style Tool.

Parameters

hST	handle to the style tool.

Returns

the style tool type, one of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4). Returns OGRSTCNone (0) if the OGRStyleToolH is invalid.

References OGR_ST_GetType().

Referenced by OGR_ST_GetType().

12.12.2.222 OGRSTUnitId OGR_ST_GetUnit (OGRStyleToolH *hST*)

Get Style Tool units.

Parameters

hST	handle to the style tool.

Returns

the style tool units.

References OGR_ST_GetUnit().

Referenced by OGR_ST_GetUnit().

12.12.2.223 void OGR_ST_SetParamDbl (OGRStyleToolH *hST,* int *eParam,* double *dfValue*)

Set Style Tool parameter value from a double.

Maps to the **OGRStyleTool** (p. 292) subclasses' SetParamDbl() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
dfValue	the new parameter value

References OGR_ST_SetParamDbl().

Referenced by OGR_ST_SetParamDbl().

12.12.2.224 void OGR_ST_SetParamNum (OGRStyleToolH *hST,* int *eParam,* int *nValue*)

Set Style Tool parameter value from an integer.

Maps to the **OGRStyleTool** (p. 292) subclasses' SetParamNum() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
nValue	the new parameter value

References OGR_ST_SetParamNum().

Referenced by OGR_ST_SetParamNum().

12.12.2.225 void OGR_ST_SetParamStr (OGRStyleToolH *hST,* int *eParam,* const char * *pszValue*)

Set Style Tool parameter value from a string.

Maps to the **OGRStyleTool** (p. 292) subclasses' SetParamStr() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
pszValue	the new parameter value

References OGR_ST_SetParamStr().

Referenced by OGR_ST_SetParamStr().

12.12.2.226 void OGR_ST_SetUnit (OGRStyleToolH *hST,* OGRSTUnitId *eUnit,* double *dfGroundPaperScale*)

Set Style Tool units.

This function is the same as OGRStyleTool::SetUnit()

Parameters

hST	handle to the style tool.
eUnit	the new unit.
dfGroundPaper-Scale	ground to paper scale factor.

References OGR_ST_SetUnit().

Referenced by OGR_ST_SetUnit().

12.12.2.227 OGRStyleTableH OGR_STBL_Create (void)

OGRStyleTable (p. 289) factory.

This function is the same as the C++ method OGRStyleTable::OGRStyleTable().

Returns

an handle to the new style table object.

References OGR_STBL_Create().

Referenced by OGR_STBL_Create().

12.12.2.228 void OGR_STBL_Destroy (OGRStyleTableH *hSTBL*)

Destroy Style Table.

Parameters

hSTBL	handle to the style table to destroy.

References OGR_STBL_Destroy().

Referenced by OGR_STBL_Destroy().

12.12.2.229 const char* OGR_STBL_Find (OGRStyleTableH *hStyleTable,* const char * *pszName*)

Get a style string by name.

This function is the same as the C++ method **OGRStyleTable::Find()** (p. 290).

Parameters

hStyleTable	handle to the style table.
pszName	the name of the style string to find.

Returns

the style string matching the name or NULL if not found or error.

References OGR_STBL_Find().

Referenced by OGR_STBL_Find().

12.12.2.230 const char* OGR_STBL_GetLastStyleName (OGRStyleTableH *hStyleTable*)

Get the style name of the last style string fetched with OGR_STBL_GetNextStyle.

This function is the same as the C++ method **OGRStyleTable::GetStyleName()** (p. 291).

Parameters

hStyleTable	handle to the style table.

Returns

the Name of the last style string or NULL on error.

References OGR_STBL_GetLastStyleName().

Referenced by OGR_STBL_GetLastStyleName().

12.12.2.231 const char* OGR_STBL_GetNextStyle (OGRStyleTableH *hStyleTable*)

Get the next style string from the table.

This function is the same as the C++ method OGRStyleTable::GetNextStyle().

Parameters

hStyleTable	handle to the style table.

Returns

the next style string or NULL on error.

References OGR_STBL_GetNextStyle().

Referenced by OGR_STBL_GetNextStyle().

12.12.2.232 int OGR_STBL_LoadStyleTable (OGRStyleTableH *hStyleTable,* const char * *pszFilename*)

Load a style table from a file.

This function is the same as the C++ method **OGRStyleTable::LoadStyleTable()** (p. 291).

Parameters

hStyleTable	handle to the style table.
pszFilename	the name of the file to load from.

Returns

TRUE on success, FALSE on error

References OGR_STBL_LoadStyleTable().

Referenced by OGR_STBL_LoadStyleTable().

12.12.2.233 void OGR_STBL_ResetStyleStringReading (OGRStyleTableH *hStyleTable*)

Reset the next style pointer to 0.

This function is the same as the C++ method OGRStyleTable::ResetStyleStringReading().

Parameters

hStyleTable	handle to the style table.

References OGR_STBL_ResetStyleStringReading().

Referenced by OGR_STBL_ResetStyleStringReading().

12.12.2.234 int OGR_STBL_SaveStyleTable (OGRStyleTableH *hStyleTable*, const char * *pszFilename*)

Save a style table to a file.

This function is the same as the C++ method **OGRStyleTable::SaveStyleTable()** (p. 292).

Parameters

hStyleTable	handle to the style table.
pszFilename	the name of the file to save to.

Returns

TRUE on success, FALSE on error

References OGR_STBL_SaveStyleTable().

Referenced by OGR_STBL_SaveStyleTable().

12.12.2.235 OGRGeometryH OGRBuildPolygonFromEdges (OGRGeometryH *hLines*, int *bBestEffort*, int *bAutoClose*, double *dfTolerance*, OGRErr * *peErr*)

Build a ring from a bunch of arcs.

Parameters

hLines	handle to an **OGRGeometryCollection** (p. 146) (or **OGRMultiLineString** (p. 195)) containing the line string geometries to be built into rings.
bBestEffort	not yet implemented???.
bAutoClose	indicates if the ring should be close when first and last points of the ring are the same.
dfTolerance	tolerance into which two arcs are considered close enough to be joined.
peErr	OGRERR_NONE on success, or OGRERR_FAILURE on failure.

Returns

an handle to the new geometry, a polygon.

References OGRLineString::addPoint(), OGRPolygon::addRingDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRLineString::getNumPoints(), OGRLineString::getX(), OGRLineString::getY(), OGRLineString::getZ(), OGRBuildPolygonFromEdges(), wkbGeometryCollection, wkbLineString, and wkbMultiLineString.

Referenced by OGRBuildPolygonFromEdges().

12.12.2.236 void OGRCleanupAll (void)

Cleanup all OGR related resources.

This function will destroy the **OGRSFDriverRegistrar** (p. 226) along with all registered drivers, and then cleanup long lived OSR (**OGRSpatialReference** (p. 230)) and CPL resources. This may be called in an application when

OGR services are no longer needed. It is not normally required, but by freeing all dynamically allocated memory it can make memory leak testing easier.

In addition to destroying the OGRDriverRegistrar, this function also calls:

- **OSRCleanup()** (p. 498)
- CPLFinderClean()
- VSICleanupFileManager()
- CPLFreeConfig()
- CPLCleanupTLS()

References OGRCleanupAll(), and OSRCleanup().

Referenced by OGRCleanupAll().

12.12.2.237 void OGRDeregisterDriver (OGRSFDriverH *hDriver*)

Remove the passed driver from the list of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::DeregisterDriver()** (p. 227).

Parameters

hDriver	handle to the driver to deregister.

Since

GDAL 1.8.0

References OGRSFDriverRegistrar::DeregisterDriver(), OGRSFDriverRegistrar::GetRegistrar(), and OGRDeregisterDriver().

Referenced by OGRDeregisterDriver().

12.12.2.238 OGRSFDriverH OGRGetDriver (int *iDriver*)

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriver()** (p. 227).

Parameters

iDriver	the driver index, from 0 to GetDriverCount()-1.

Returns

handle to the driver, or NULL if iDriver is out of range.

References OGRSFDriverRegistrar::GetDriver(), and OGRGetDriver().

Referenced by OGRGetDriver().

12.12.2.239 OGRSFDriverH OGRGetDriverByName (const char * *pszName*)

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverByName()** (p. 227)

Parameters

pszName	the driver name

Returns

the driver, or NULL if no driver with that name is found

References OGRSFDriverRegistrar::GetDriverByName(), OGRSFDriverRegistrar::GetRegistrar(), and OGRGetDriverByName().

Referenced by OGRGetDriverByName().

12.12.2.240 int OGRGetDriverCount (void)

Fetch the number of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverCount()** (p. 228).

Returns

the drivers count.

References OGRSFDriverRegistrar::GetDriverCount(), and OGRGetDriverCount().

Referenced by OGRGetDriverCount().

12.12.2.241 OGRDataSourceH OGRGetOpenDS (int *iDS*)

Return the iDS th datasource opened.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetOpenDS** (p. 228).

Parameters

iDS	the index of the dataset to return (between 0 and GetOpenDSCount() - 1)

References OGRSFDriverRegistrar::GetOpenDS(), OGRSFDriverRegistrar::GetRegistrar(), and OGRGetOpenDS().

Referenced by OGRGetOpenDS().

12.12.2.242 int OGRGetOpenDSCount (void)

Return the number of opened datasources.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetOpenDSCount()** (p. 228)

Returns

the number of opened datasources.

References OGRSFDriverRegistrar::GetOpenDSCount(), OGRSFDriverRegistrar::GetRegistrar(), and OGRGetOpenDSCount().

Referenced by OGRGetOpenDSCount().

12.12.2.243 OGRDataSourceH OGROpen (const char * *pszName,* int *bUpdate,* OGRSFDriverH * *pahDriverList*)

Open a file / data source with one of the registered drivers.

This function loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This function is static. Applications don't normally need to use any other **OGRSFDriverRegistrar** (p. 226) function, not do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 226) instance.

If this function fails, **CPLGetLastErrorMsg()** (p. 335) can be used to check if there is an error message explaining why.

For drivers supporting the VSI virtual file API, it is possible to open a file in a .zip archive (see **VSIInstallZipFileHandler()** (p. 380)), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()** (p. 380)) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()** (p. 377))

This function is the same as the C++ method **OGRSFDriverRegistrar::Open()** (p. 229).

Parameters

pszName	the name of the file, or data source to open.
bUpdate	FALSE for read-only access (the default) or TRUE for read-write access.
pahDriverList	if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

Returns

NULL on error or if the pass name is not supported by this driver, otherwise an handle to an **OGRDataSource** (p. 85). This **OGRDataSource** (p. 85) should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSourceH        hDS;
OGRSFDriverH        *pahDriver;

hDS = OGROpen( "polygon.shp", 0, pahDriver );
if( hDS == NULL )
{
    return;
}

... use the data source ...

OGRReleaseDataSource( hDS );
```

References OGROpen(), and OGRSFDriverRegistrar::Open().

Referenced by OGROpen().

12.12.2.244 void OGRRegisterDriver (OGRSFDriverH *hDriver*)

Add a driver to the list of registered drivers.

If the passed driver is already registered (based on handle comparison) then the driver isn't registered. New drivers are added at the end of the list of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::RegisterDriver()** (p. 229).

Parameters

hDriver	handle to the driver to add.

References OGRSFDriverRegistrar::GetRegistrar(), OGRRegisterDriver(), and OGRSFDriverRegistrar::RegisterDriver().

Referenced by OGRRegisterDriver().

12.12.2.245 OGRErr OGRReleaseDataSource (OGRDataSourceH *hDS*)

Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.

Internally this actually calls the OGRSFDriverRegistrar::ReleaseDataSource() method. This method is essentially a convenient alias.

This method is the same as the C++ method **OGRDataSource::Release()** (p. 91)

Parameters

hDS	handle to the data source to release

Returns

OGRERR_NONE on success or an error code.

References OGRSFDriverRegistrar::GetRegistrar(), and OGRReleaseDataSource().

Referenced by OGRReleaseDataSource().

12.12.2.246 OGRErr OGRSetGenerate_DB2_V72_BYTE_ORDER (int *bGenerate_DB2_V72_BYTE_ORDER*)

Special entry point to enable the hack for generating DB2 V7.2 style WKB.

DB2 seems to have placed (and require) an extra 0x30 or'ed with the byte order in WKB. This entry point is used to turn on or off the generation of such WKB.

References OGRSetGenerate_DB2_V72_BYTE_ORDER().

Referenced by OGRSetGenerate_DB2_V72_BYTE_ORDER().

12.13 ogr_core.h File Reference

```
#include "cpl_port.h"
#include "gdal_version.h"
```

Classes

- class **OGREnvelope**
- class **OGREnvelope3D**
- union **OGRField**

Macros

- #define **GDAL_CHECK_VERSION**(pszCallingComponentName) **GDALCheckVersion**(GDAL_VERSION_-MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Typedefs

- typedef enum **ogr_style_tool_class_id OGRSTClassId**
- typedef enum **ogr_style_tool_units_id OGRSTUnitId**
- typedef enum **ogr_style_tool_param_pen_id OGRSTPenParam**

- typedef enum
  ogr_style_tool_param_brush_id OGRSTBrushParam
- typedef enum
  ogr_style_tool_param_symbol_id OGRSTSymbolParam
- typedef enum
  ogr_style_tool_param_label_id OGRSTLabelParam

Enumerations

- enum **OGRwkbGeometryType** {
  wkbUnknown = 0, **wkbPoint** = 1, **wkbLineString** = 2, **wkbPolygon** = 3,
  wkbMultiPoint = 4, **wkbMultiLineString** = 5, **wkbMultiPolygon** = 6, **wkbGeometryCollection** = 7,
  wkbNone = 100, **wkbLinearRing** = 101, **wkbPoint25D** = 0x80000001, **wkbLineString25D** = 0x80000002,
  wkbPolygon25D = 0x80000003, **wkbMultiPoint25D** = 0x80000004, **wkbMultiLineString25D** = 0x80000005, **wkbMultiPolygon25D** = 0x80000006,
  wkbGeometryCollection25D = 0x80000007 }
- enum **OGRFieldType** {
  OFTInteger = 0, **OFTIntegerList** = 1, **OFTReal** = 2, **OFTRealList** = 3,
  OFTString = 4, **OFTStringList** = 5, **OFTWideString** = 6, **OFTWideStringList** = 7,
  OFTBinary = 8, **OFTDate** = 9, **OFTTime** = 10, **OFTDateTime** = 11 }
- enum **OGRJustification**
- enum **ogr_style_tool_class_id**
- enum **ogr_style_tool_units_id**
- enum **ogr_style_tool_param_pen_id**
- enum **ogr_style_tool_param_brush_id**
- enum **ogr_style_tool_param_symbol_id**
- enum **ogr_style_tool_param_label_id**

Functions

- const char * **OGRGeometryTypeToName** (**OGRwkbGeometryType** eType)

  Fetch a human readable name corresponding to an OGRwkBGeometryType value. The returned value should not be modified, or freed by the application.
- **OGRwkbGeometryType OGRMergeGeometryTypes** (**OGRwkbGeometryType** eMain, **OGRwkbGeometryType** eExtra)

  Find common geometry type.
- int **OGRParseDate** (const char *pszInput, **OGRField** *psOutput, int nOptions)
- int CPL_STDCALL **GDALCheckVersion** (int nVersionMajor, int nVersionMinor, const char *pszCallingComponentName)

12.13.1 Detailed Description

Core portability services for cross-platform OGR code.

12.13.2 Macro Definition Documentation

12.13.2.1 #define GDAL_CHECK_VERSION(*pszCallingComponentName*) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Helper macro for GDALCheckVersion

12.13.3 Typedef Documentation

12.13.3.1 typedef enum ogr_style_tool_param_brush_id OGRSTBrushParam

List of parameters for use with **OGRStyleBrush** (p. 283).

12.13.3.2 typedef enum ogr_style_tool_class_id OGRSTClassId

OGRStyleTool (p. 292) derived class types (returned by GetType()).

12.13.3.3 typedef enum ogr_style_tool_param_label_id OGRSTLabelParam

List of parameters for use with **OGRStyleLabel** (p. 284).

12.13.3.4 typedef enum ogr_style_tool_param_pen_id OGRSTPenParam

List of parameters for use with **OGRStylePen** (p. 288).

12.13.3.5 typedef enum ogr_style_tool_param_symbol_id OGRSTSymbolParam

List of parameters for use with **OGRStyleSymbol** (p. 289).

12.13.3.6 typedef enum ogr_style_tool_units_id OGRSTUnitId

List of units supported by OGRStyleTools.

12.13.4 Enumeration Type Documentation

12.13.4.1 enum ogr_style_tool_class_id

OGRStyleTool (p. 292) derived class types (returned by GetType()).

12.13.4.2 enum ogr_style_tool_param_brush_id

List of parameters for use with **OGRStyleBrush** (p. 283).

12.13.4.3 enum ogr_style_tool_param_label_id

List of parameters for use with **OGRStyleLabel** (p. 284).

12.13.4.4 enum ogr_style_tool_param_pen_id

List of parameters for use with **OGRStylePen** (p. 288).

12.13.4.5 enum ogr_style_tool_param_symbol_id

List of parameters for use with **OGRStyleSymbol** (p. 289).

12.13.4.6 enum ogr_style_tool_units_id

List of units supported by OGRStyleTools.

12.13.4.7 enum OGRFieldType

List of feature field types. This list is likely to be extended in the future ... avoid coding applications based on the assumption that all field types can be known.

Enumerator:

OFTInteger Simple 32bit integer
OFTIntegerList List of 32bit integers
OFTReal Double Precision floating point
OFTRealList List of doubles
OFTString String of ASCII chars
OFTStringList Array of strings
OFTWideString deprecated
OFTWideStringList deprecated
OFTBinary Raw Binary data
OFTDate Date
OFTTime Time
OFTDateTime Date and Time

12.13.4.8 enum OGRJustification

Display justification for field values.

12.13.4.9 enum OGRwkbGeometryType

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGRGeometry::getGeometryType()** (p. 137) to identify the type of a geometry object.

Enumerator:

wkbUnknown unknown type, non-standard
wkbPoint 0-dimensional geometric object, standard WKB
wkbLineString 1-dimensional geometric object with linear interpolation between Points, standard WKB
wkbPolygon planar 2-dimensional geometric object defined by 1 exterior boundary and 0 or more interior boundaries, standard WKB
wkbMultiPoint GeometryCollection of Points, standard WKB
wkbMultiLineString GeometryCollection of LineStrings, standard WKB
wkbMultiPolygon GeometryCollection of Polygons, standard WKB
wkbGeometryCollection geometric object that is a collection of 1 or more geometric objects, standard WKB
wkbNone non-standard, for pure attribute records
wkbLinearRing non-standard, just for createGeometry()
wkbPoint25D 2.5D extension as per 99-402
wkbLineString25D 2.5D extension as per 99-402
wkbPolygon25D 2.5D extension as per 99-402
wkbMultiPoint25D 2.5D extension as per 99-402
wkbMultiLineString25D 2.5D extension as per 99-402
wkbMultiPolygon25D 2.5D extension as per 99-402
wkbGeometryCollection25D 2.5D extension as per 99-402

12.13.5 Function Documentation

12.13.5.1 int CPL_STDCALL GDALCheckVersion (int *nVersionMajor,* int *nVersionMinor,* const char * *pszCallingComponentName*)

Return TRUE if GDAL library version at runtime matches nVersionMajor.nVersionMinor.

The purpose of this method is to ensure that calling code will run with the GDAL version it is compiled for. It is primarly intented for external plugins.

Parameters

nVersionMajor	Major version to be tested against
nVersionMinor	Minor version to be tested against
pszCalling-Component-Name	If not NULL, in case of version mismatch, the method will issue a failure mentionning the name of the calling component.

12.13.5.2 const char* OGRGeometryTypeToName (OGRwkbGeometryType *eType*)

Fetch a human readable name corresponding to an OGRwkBGeometryType value. The returned value should not be modified, or freed by the application.

This function is C callable.

Parameters

eType	the geometry type.

Returns

internal human readable string, or NULL on failure.

References OGRGeometryTypeToName(), wkbGeometryCollection, wkbGeometryCollection25D, wkbLineString, wkbLineString25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by OGRGeometryTypeToName().

12.13.5.3 OGRwkbGeometryType OGRMergeGeometryTypes (OGRwkbGeometryType *eMain,* OGRwkbGeometryType *eExtra*)

Find common geometry type.

Given two geometry types, find the most specific common type. Normally used repeatedly with the geometries in a layer to try and establish the most specific geometry type that can be reported for the layer.

NOTE: wkbUnknown is the "worst case" indicating a mixture of geometry types with nothing in common but the base geometry type. wkbNone should be used to indicate that no geometries have been encountered yet, and means the first geometry encounted will establish the preliminary type.

Parameters

eMain	the first input geometry type.
eExtra	the second input geometry type.

Returns

the merged geometry type.

References OGRMergeGeometryTypes(), wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbNone, and wkbUnknown.

Referenced by OGRMergeGeometryTypes().

12.13.5.4 int OGRParseDate (const char * *pszInput,* OGRField * *psField,* int *nOptions*)

Parse date string.

This function attempts to parse a date string in a variety of formats into the OGRField.Date format suitable for use with OGR. Generally speaking this function is expecting values like:

YYYY-MM-DD HH:MM:SS+nn

The seconds may also have a decimal portion (which is ignored). And just dates (YYYY-MM-DD) or just times (HH:MM:SS) are also supported. The date may also be in YYYY/MM/DD format. If the year is less than 100 and greater than 30 a "1900" century value will be set. If it is less than 30 and greater than -1 then a "2000" century value will be set. In the future this function may be generalized, and additional control provided through nOptions, but an nOptions value of "0" should always do a reasonable default form of processing.

The value of psField will be indeterminate if the function fails (returns FALSE).

Parameters

pszInput	the input date string.
psField	the **OGRField** (p. 116) that will be updated with the parsed result.
nOptions	parsing options, for now always 0.

Returns

TRUE if apparently successful or FALSE on failure.

References OGRParseDate().

Referenced by OGRParseDate(), and OGRFeature::SetField().

12.14 ogr_feature.h File Reference

```
#include "ogr_geometry.h"
#include "ogr_featurestyle.h"
#include "cpl_atomic_ops.h"
```

Classes

- class **OGRFieldDefn**
- class **OGRFeatureDefn**
- class **OGRFeature**
- class **OGRFeatureQuery**

12.14.1 Detailed Description

Simple feature classes.

12.15 ogr_featurestyle.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "ogr_core.h"
```

Classes

- struct **ogr_style_param**
- struct **ogr_style_value**
- class **OGRStyleTable**
- class **OGRStyleMgr**
- class **OGRStyleTool**
- class **OGRStylePen**
- class **OGRStyleBrush**
- class **OGRStyleSymbol**
- class **OGRStyleLabel**

12.15.1 Detailed Description

Simple feature style classes.

12.16 ogr_geometry.h File Reference

```
#include "ogr_core.h"
#include "ogr_spatialref.h"
```

Classes

- class **OGRRawPoint**
- class **OGRGeometry**
- class **OGRPoint**
- class **OGRCurve**
- class **OGRLineString**
- class **OGRLinearRing**
- class **OGRSurface**
- class **OGRPolygon**
- class **OGRGeometryCollection**
- class **OGRMultiPolygon**
- class **OGRMultiPoint**
- class **OGRMultiLineString**
- class **OGRGeometryFactory**

12.16.1 Detailed Description

Simple feature geometry classes.

12.17 ogr_spatialref.h File Reference

```
#include "ogr_srs_api.h"
```

Classes

- class **OGR_SRSNode**
- class **OGRSpatialReference**
- class **OGRCoordinateTransformation**

Functions

- **OGRCoordinateTransformation** * **OGRCreateCoordinateTransformation** (**OGRSpatialReference** *poSource, **OGRSpatialReference** *poTarget)

12.17.1 Detailed Description

Coordinate systems services.

12.17.2 Function Documentation

12.17.2.1 OGRCoordinateTransformation* OGRCreateCoordinateTransformation (OGRSpatialReference * *poSource,* OGRSpatialReference * *poTarget*)

Create transformation object.

This is the same as the C function **OCTNewCoordinateTransformation()** (p. 496).

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

The delete operator, or **OCTDestroyCoordinateTransformation()** (p. 496) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

Parameters

poSource	source spatial reference system.
poTarget	target spatial reference system.

Returns

NULL on failure or a ready to use transformation object.

References OGRCreateCoordinateTransformation().

Referenced by OCTNewCoordinateTransformation(), OGRCreateCoordinateTransformation(), and OGRGeometry::transformTo().

12.18 ogr_srs_api.h File Reference

```
#include "ogr_core.h"
```

Functions

- const char * **OSRAxisEnumToName** (OGRAxisOrientation eOrientation)

  Return the string representation for the OGRAxisOrientation enumeration.
- OGRSpatialReferenceH CPL_STDCALL **OSRNewSpatialReference** (const char *)

  Constructor.
- OGRSpatialReferenceH CPL_STDCALL **OSRCloneGeogCS** (OGRSpatialReferenceH)

  *Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. 230) object.*
- OGRSpatialReferenceH CPL_STDCALL **OSRClone** (OGRSpatialReferenceH)

  *Make a duplicate of this **OGRSpatialReference** (p. 230).*
- void CPL_STDCALL **OSRDestroySpatialReference** (OGRSpatialReferenceH)

  ***OGRSpatialReference** (p. 230) destructor.*
- int **OSRReference** (OGRSpatialReferenceH)

  Increments the reference count by one.
- int **OSRDereference** (OGRSpatialReferenceH)

  Decrements the reference count by one.
- void **OSRRelease** (OGRSpatialReferenceH)

  Decrements the reference count by one, and destroy if zero.
- OGRErr **OSRValidate** (OGRSpatialReferenceH)

  Validate SRS tokens.
- OGRErr **OSRFixupOrdering** (OGRSpatialReferenceH)

  Correct parameter ordering to match CT Specification.
- OGRErr **OSRFixup** (OGRSpatialReferenceH)

  Fixup as needed.
- OGRErr **OSRStripCTParms** (OGRSpatialReferenceH)

  Strip OGC CT Parameters.
- OGRErr CPL_STDCALL **OSRImportFromEPSG** (OGRSpatialReferenceH, int)

  Initialize SRS based on EPSG GCS or PCS code.
- OGRErr CPL_STDCALL **OSRImportFromEPSGA** (OGRSpatialReferenceH, int)

  Initialize SRS based on EPSG GCS or PCS code.
- OGRErr **OSRImportFromWkt** (OGRSpatialReferenceH, char **)

  Import from WKT string.
- OGRErr **OSRImportFromProj4** (OGRSpatialReferenceH, const char *)

  Import PROJ.4 coordinate string.
- OGRErr **OSRImportFromESRI** (OGRSpatialReferenceH, char **)

  Import coordinate system from ESRI .prj format(s).
- OGRErr **OSRImportFromPCI** (OGRSpatialReferenceH hSRS, const char *, const char *, double *)

  Import coordinate system from PCI projection definition.
- OGRErr **OSRImportFromUSGS** (OGRSpatialReferenceH, long, long, double *, long)

  Import coordinate system from USGS projection definition.
- OGRErr **OSRImportFromXML** (OGRSpatialReferenceH, const char *)

  Import coordinate system from XML format (GML only currently).
- OGRErr **OSRImportFromMICoordSys** (OGRSpatialReferenceH, const char *)

  Import Mapinfo style CoordSys definition.
- OGRErr **OSRImportFromERM** (OGRSpatialReferenceH, const char *, const char *, const char *)

  Create OGR WKT from ERMapper projection definitions.
- OGRErr **OSRImportFromUrl** (OGRSpatialReferenceH, const char *)

  Set spatial reference from a URL.
- OGRErr CPL_STDCALL **OSRExportToWkt** (OGRSpatialReferenceH, char **)

  Convert this SRS into WKT format.
- OGRErr CPL_STDCALL **OSRExportToPrettyWkt** (OGRSpatialReferenceH, char **, int)

Convert this SRS into a a nicely formatted WKT string for display to a person.

- OGRErr CPL_STDCALL **OSRExportToProj4** (OGRSpatialReferenceH, char **)

  Export coordinate system in PROJ.4 format.
- OGRErr **OSRExportToPCI** (OGRSpatialReferenceH, char **, char **, double **)

  Export coordinate system in PCI projection definition.
- OGRErr **OSRExportToUSGS** (OGRSpatialReferenceH, long *, long *, double **, long *)

  Export coordinate system in USGS GCTP projection definition.
- OGRErr **OSRExportToXML** (OGRSpatialReferenceH, char **, const char *)

  Export coordinate system in XML format.
- OGRErr **OSRExportToMICoordSys** (OGRSpatialReferenceH, char **)

  Export coordinate system in Mapinfo style CoordSys format.
- OGRErr **OSRExportToERM** (OGRSpatialReferenceH, char *, char *, char *)

  Convert coordinate system to ERMapper format.
- OGRErr **OSRMorphToESRI** (OGRSpatialReferenceH)

  Convert in place to ESRI WKT format.
- OGRErr **OSRMorphFromESRI** (OGRSpatialReferenceH)

  Convert in place from ESRI WKT format.
- OGRErr CPL_STDCALL **OSRSetAttrValue** (OGRSpatialReferenceH hSRS, const char *pszNodePath, const char *pszNewNodeValue)

  Set attribute value in spatial reference.
- const char *CPL_STDCALL **OSRGetAttrValue** (OGRSpatialReferenceH hSRS, const char *pszName, int iChild)

  Fetch indicated attribute of named node.
- OGRErr **OSRSetAngularUnits** (OGRSpatialReferenceH, const char *, double)

  Set the angular units for the geographic coordinate system.
- double **OSRGetAngularUnits** (OGRSpatialReferenceH, char **)

  Fetch angular geographic coordinate system units.
- OGRErr **OSRSetLinearUnits** (OGRSpatialReferenceH, const char *, double)

  Set the linear units for the projection.
- OGRErr **OSRSetTargetLinearUnits** (OGRSpatialReferenceH, const char *, const char *, double)

  Set the linear units for the target node.
- OGRErr **OSRSetLinearUnitsAndUpdateParameters** (OGRSpatialReferenceH, const char *, double)

  Set the linear units for the projection.
- double **OSRGetLinearUnits** (OGRSpatialReferenceH, char **)

  Fetch linear projection units.
- double **OSRGetTargetLinearUnits** (OGRSpatialReferenceH, const char *, char **)

  Fetch linear projection units.
- double **OSRGetPrimeMeridian** (OGRSpatialReferenceH, char **)

  Fetch prime meridian info.
- int **OSRIsGeographic** (OGRSpatialReferenceH)

  Check if geographic coordinate system.
- int **OSRIsLocal** (OGRSpatialReferenceH)

  Check if local coordinate system.
- int **OSRIsProjected** (OGRSpatialReferenceH)

  Check if projected coordinate system.
- int **OSRIsCompound** (OGRSpatialReferenceH)

  Check if the coordinate system is compound.
- int **OSRIsGeocentric** (OGRSpatialReferenceH)

  Check if geocentric coordinate system.
- int **OSRIsVertical** (OGRSpatialReferenceH)

  Check if vertical coordinate system.

- int **OSRIsSameGeogCS** (OGRSpatialReferenceH, OGRSpatialReferenceH)

  Do the GeogCS'es match?
- int **OSRIsSameVertCS** (OGRSpatialReferenceH, OGRSpatialReferenceH)

  Do the VertCS'es match?
- int **OSRIsSame** (OGRSpatialReferenceH, OGRSpatialReferenceH)

  Do these two spatial references describe the same system ?
- OGRErr **OSRSetLocalCS** (OGRSpatialReferenceH hSRS, const char *pszName)

  Set the user visible LOCAL_CS name.
- OGRErr **OSRSetProjCS** (OGRSpatialReferenceH hSRS, const char *pszName)

  Set the user visible PROJCS name.
- OGRErr **OSRSetGeocCS** (OGRSpatialReferenceH hSRS, const char *pszName)

  Set the user visible PROJCS name.
- OGRErr **OSRSetWellKnownGeogCS** (OGRSpatialReferenceH hSRS, const char *pszName)

  Set a GeogCS based on well known name.
- OGRErr CPL_STDCALL **OSRSetFromUserInput** (OGRSpatialReferenceH hSRS, const char *)

  Set spatial reference from various text formats.
- OGRErr **OSRCopyGeogCSFrom** (OGRSpatialReferenceH hSRS, OGRSpatialReferenceH hSrcSRS)

  *Copy GEOGCS from another **OGRSpatialReference** (p. 230).*
- OGRErr **OSRSetTOWGS84** (OGRSpatialReferenceH hSRS, double, double, double, double, double, double, double)

  Set the Bursa-Wolf conversion to WGS84.
- OGRErr **OSRGetTOWGS84** (OGRSpatialReferenceH hSRS, double *, int)

  Fetch TOWGS84 parameters, if available.
- OGRErr **OSRSetCompoundCS** (OGRSpatialReferenceH hSRS, const char *pszName, OGRSpatialReferenceH hHorizSRS, OGRSpatialReferenceH hVertSRS)

  Setup a compound coordinate system.
- OGRErr **OSRSetGeogCS** (OGRSpatialReferenceH hSRS, const char *pszGeogName, const char *pszDatumName, const char *pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char *pszPMName, double dfPMOffset, const char *pszUnits, double dfConvertToRadians)

  Set geographic coordinate system.
- OGRErr **OSRSetVertCS** (OGRSpatialReferenceH hSRS, const char *pszVertCSName, const char *pszVertDatumName, int nVertDatumType)

  Setup the vertical coordinate system.
- double **OSRGetSemiMajor** (OGRSpatialReferenceH, OGRErr *)

  Get spheroid semi major axis.
- double **OSRGetSemiMinor** (OGRSpatialReferenceH, OGRErr *)

  Get spheroid semi minor axis.
- double **OSRGetInvFlattening** (OGRSpatialReferenceH, OGRErr *)

  Get spheroid inverse flattening.
- OGRErr **OSRSetAuthority** (OGRSpatialReferenceH hSRS, const char *pszTargetKey, const char *pszAuthority, int nCode)

  Set the authority for a node.
- const char * **OSRGetAuthorityCode** (OGRSpatialReferenceH hSRS, const char *pszTargetKey)

  Get the authority code for a node.
- const char * **OSRGetAuthorityName** (OGRSpatialReferenceH hSRS, const char *pszTargetKey)

  Get the authority name for a node.
- OGRErr **OSRSetProjection** (OGRSpatialReferenceH, const char *)

  Set a projection name.
- OGRErr **OSRSetProjParm** (OGRSpatialReferenceH, const char *, double)

  Set a projection parameter value.
- double **OSRGetProjParm** (OGRSpatialReferenceH hSRS, const char *pszParmName, double dfDefault, OGRErr *)

Fetch a projection parameter value.

- OGRErr **OSRSetNormProjParm** (OGRSpatialReferenceH, const char *, double)

  Set a projection parameter with a normalized value.

- double **OSRGetNormProjParm** (OGRSpatialReferenceH hSRS, const char *pszParmName, double dfDefault, OGRErr *)

  *This function is the same as **OGRSpatialReference** (p. 230)::*

- OGRErr **OSRSetUTM** (OGRSpatialReferenceH hSRS, int nZone, int bNorth)

  Set UTM projection definition.

- int **OSRGetUTMZone** (OGRSpatialReferenceH hSRS, int *pbNorth)

  Get utm zone information.

- OGRErr **OSRSetStatePlane** (OGRSpatialReferenceH hSRS, int nZone, int bNAD83)

  Set State Plane projection definition.

- OGRErr **OSRSetStatePlaneWithUnits** (OGRSpatialReferenceH hSRS, int nZone, int bNAD83, const char *pszOverrideUnitName, double dfOverrideUnit)

  Set State Plane projection definition.

- OGRErr **OSRAutoIdentifyEPSG** (OGRSpatialReferenceH hSRS)

  Set EPSG authority info if possible.

- int **OSREPSGTreatsAsLatLong** (OGRSpatialReferenceH hSRS)

  This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

- const char * **OSRGetAxis** (OGRSpatialReferenceH hSRS, const char *pszTargetKey, int iAxis, OGRAxisOrientation *peOrientation)

  Fetch the orientation of one axis.

- OGRErr **OSRSetACEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetAE** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetBonne** (OGRSpatialReferenceH hSRS, double dfStandardParallel, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetCEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetCS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckert** (OGRSpatialReferenceH hSRS, int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckertIV** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckertVI** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquirectangular** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquirectangular2** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGH** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetIGH** (OGRSpatialReferenceH hSRS)
- OGRErr **OSRSetGEOS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGaussSchreiberTMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **OSRSetGnomonic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetHOM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)

  Set a Hotine Oblique Mercator projection using azimuth angle.
- OGRErr **OSRSetHOM2PNO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)

  Set a Hotine Oblique Mercator projection using two points on projection centerline.
- OGRErr **OSRSetIWMPolyconic** (OGRSpatialReferenceH hSRS, double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetKrovak** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLAEA** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC1SP** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCCB** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMC** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMollweide** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetNZMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOS** (OGRSpatialReferenceH hSRS, double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOrthographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPolyconic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetRobinson** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetSinusoidal** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetStereographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetSOC** (OGRSpatialReferenceH hSRS, double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMVariant** (OGRSpatialReferenceH hSRS, const char *pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMSO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetVDG** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetWagner** (OGRSpatialReferenceH hSRS, int nVariation, double dfFalseEasting, double dfFalseNorthing)

- void **OSRCleanup** (void)

  Cleanup cached SRS related memory.
- OGRCoordinateTransformationH CPL_STDCALL **OCTNewCoordinateTransformation** (OGRSpatialReferenceH hSourceSRS, OGRSpatialReferenceH hTargetSRS)
- void CPL_STDCALL **OCTDestroyCoordinateTransformation** (OGRCoordinateTransformationH)

  OGRCoordinateTransformation *(p. 81) destructor.*
- char ** **OPTGetProjectionMethods** ()
- char ** **OPTGetParameterList** (const char *pszProjectionMethod, char **ppszUserName)
- int **OPTGetParameterInfo** (const char *pszProjectionMethod, const char *pszParameterName, char **ppszUserName, char **ppszType, double *pdfDefaultValue)

12.18.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr_spatialref.h** (p. 490)

12.18.2 Function Documentation

12.18.2.1 void CPL_STDCALL OCTDestroyCoordinateTransformation (OGRCoordinateTransformationH *hCT*)

OGRCoordinateTransformation (p. 81) destructor.

This function is the same as **OGRCoordinateTransformation::DestroyCT()** (p. 82)

Parameters

hCT	the object to delete

References OCTDestroyCoordinateTransformation().

Referenced by OCTDestroyCoordinateTransformation().

12.18.2.2 OGRCoordinateTransformationH CPL_STDCALL OCTNewCoordinateTransformation (OGRSpatialReferenceH *hSourceSRS,* OGRSpatialReferenceH *hTargetSRS*)

Create transformation object.

This is the same as the C++ function **OGRCreateCoordinateTransformation()** (p. 490).

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

OCTDestroyCoordinateTransformation() (p. 496) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

Parameters

hSourceSRS	source spatial reference system.
hTargetSRS	target spatial reference system.

Returns

NULL on failure or a ready to use transformation object.

References OCTNewCoordinateTransformation(), and OGRCreateCoordinateTransformation().

Referenced by OCTNewCoordinateTransformation().

12.18.2.3 int OPTGetParameterInfo (const char * *pszProjectionMethod,* const char * *pszParameterName,* char ** *ppszUserName,* char ** *ppszType,* double * *pdfDefaultValue*)

Fetch information about a single parameter of a projection method.

Parameters

pszProjection-Method	name of projection method for which the parameter applies. Not currently used, but in the future this could affect defaults. This is the internal projection method name, such as "Tranverse-_Mercator".
pszParameter-Name	name of the parameter to fetch information about. This is the internal name such as "central-_meridian" (SRS_PP_CENTRAL_MERIDIAN).
ppszUserName	location at which to return the user visible name for the parameter. This pointer may be NULL to skip the user name. The returned name should not be modified or freed.
ppszType	location at which to return the parameter type for the parameter. This pointer may be NULL to skip. The returned type should not be modified or freed. The type values are described above.
pdfDefaultValue	location at which to put the default value for this parameter. The pointer may be NULL.

Returns

TRUE if parameter found, or FALSE otherwise.

12.18.2.4 char** OPTGetParameterList (const char * *pszProjectionMethod,* char ** *ppszUserName*)

Fetch the parameters for a given projection method.

Parameters

pszProjection-Method	internal name of projection methods to fetch the parameters for, such as "Transverse_-Mercator" (SRS_PT_TRANSVERSE_MERCATOR).
ppszUserName	pointer in which to return a user visible name for the projection name. The returned string should not be modified or freed by the caller. Legal to pass in NULL if user name not required.

Returns

returns a NULL terminated list of internal parameter names that should be freed by the caller when no longer needed. Returns NULL if projection method is unknown.

12.18.2.5 char** OPTGetProjectionMethods ( )

Fetch list of possible projection methods.

Returns

Returns NULL terminated list of projection methods. This should be freed with **CSLDestroy()** (p. 365) when no longer needed.

12.18.2.6 OGRErr OSRAutoIdentifyEPSG (OGRSpatialReferenceH *hSRS*)

Set EPSG authority info if possible.

This function is the same as **OGRSpatialReference::AutoIdentifyEPSG()** (p. 236).

12.18.2.7 const char* OSRAxisEnumToName (OGRAxisOrientation *eOrientation*)

Return the string representation for the OGRAxisOrientation enumeration.

For example "NORTH" for OAO_North.

Returns

an internal string

References OSRAxisEnumToName().

Referenced by OSRAxisEnumToName(), and OGRSpatialReference::SetAxes().

12.18.2.8 void OSRCleanup (void)

Cleanup cached SRS related memory.

This function will attempt to cleanup any cache spatial reference related information, such as cached tables of coordinate systems.

References OSRCleanup().

Referenced by OGRCleanupAll(), and OSRCleanup().

12.18.2.9 OGRSpatialReferenceH CPL_STDCALL OSRClone (OGRSpatialReferenceH *hSRS*)

Make a duplicate of this **OGRSpatialReference** (p. 230).

This function is the same as **OGRSpatialReference::Clone()** (p. 236)

References OSRClone().

Referenced by OSRClone().

12.18.2.10 OGRSpatialReferenceH CPL_STDCALL OSRCloneGeogCS (OGRSpatialReferenceH *hSource*)

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. 230) object.

This function is the same as **OGRSpatialReference::CloneGeogCS()** (p. 237).

References OSRCloneGeogCS().

Referenced by OSRCloneGeogCS().

12.18.2.11 OGRErr OSRCopyGeogCSFrom (OGRSpatialReferenceH *hSRS,* OGRSpatialReferenceH *hSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. 230).

This function is the same as **OGRSpatialReference::CopyGeogCSFrom()** (p. 237)

References OSRCopyGeogCSFrom().

Referenced by OSRCopyGeogCSFrom().

12.18.2.12 int OSRDereference (OGRSpatialReferenceH *hSRS*)

Decrements the reference count by one.

This function is the same as **OGRSpatialReference::Dereference()** (p. 237)

References OSRDereference().

Referenced by OSRDereference().

12.18.2.13 void CPL_STDCALL OSRDestroySpatialReference (OGRSpatialReferenceH *hSRS*)

OGRSpatialReference (p. 230) destructor.

This function is the same as **OGRSpatialReference::~OGRSpatialReference()** (p. 236) and **OGRSpatialReference::DestroySpatialReference()** (p. 237)

Parameters

hSRS	the object to delete

References OSRDestroySpatialReference().

Referenced by OSRDestroySpatialReference().

12.18.2.14 int OSREPSGTreatsAsLatLong (OGRSpatialReferenceH *hSRS*)

This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

This function is the same as OGRSpatialReference::OSREPSGTreatsAsLatLong().

12.18.2.15 OGRErr OSRExportToERM (OGRSpatialReferenceH *hSRS,* char * *pszProj,* char * *pszDatum,* char * *pszUnits*)

Convert coordinate system to ERMapper format.

This function is the same as **OGRSpatialReference::exportToERM()** (p. 238).

References OSRExportToERM().

Referenced by OSRExportToERM().

12.18.2.16 OGRErr OSRExportToMICoordSys (OGRSpatialReferenceH *hSRS,* char ** *ppszReturn*)

Export coordinate system in Mapinfo style CoordSys format.

This method is the equivalent of the C++ method **OGRSpatialReference::exportToMICoordSys** (p. 238)

References OSRExportToMICoordSys().

Referenced by OSRExportToMICoordSys().

12.18.2.17 OGRErr OSRExportToPCI (OGRSpatialReferenceH *hSRS,* char ** *ppszProj,* char ** *ppszUnits,* double ** *ppadfPrjParams*)

Export coordinate system in PCI projection definition.

This function is the same as **OGRSpatialReference::exportToPCI()** (p. 239).

References OSRExportToPCI().

Referenced by OSRExportToPCI().

12.18.2.18 OGRErr CPL_STDCALL OSRExportToPrettyWkt (OGRSpatialReferenceH *hSRS,* char ** *ppszReturn,* int *bSimplify*)

Convert this SRS into a a nicely formatted WKT string for display to a person.

This function is the same as **OGRSpatialReference::exportToPrettyWkt()** (p. 240).

References OSRExportToPrettyWkt().

Referenced by OSRExportToPrettyWkt().

12.18.2.19 OGRErr CPL_STDCALL OSRExportToProj4 (OGRSpatialReferenceH *hSRS*, char ** *ppszReturn*)

Export coordinate system in PROJ.4 format.

This function is the same as **OGRSpatialReference::exportToProj4()** (p. 240).

References OSRExportToProj4().

Referenced by OSRExportToProj4().

12.18.2.20 OGRErr OSRExportToUSGS (OGRSpatialReferenceH *hSRS*, long * *piProjSys*, long * *piZone*, double ** *ppadfPrjParams*, long * *piDatum*)

Export coordinate system in USGS GCTP projection definition.

This function is the same as **OGRSpatialReference::exportToUSGS()** (p. 240).

References OSRExportToUSGS().

Referenced by OSRExportToUSGS().

12.18.2.21 OGRErr CPL_STDCALL OSRExportToWkt (OGRSpatialReferenceH *hSRS*, char ** *ppszReturn*)

Convert this SRS into WKT format.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. 241).

References OSRExportToWkt().

Referenced by OSRExportToWkt().

12.18.2.22 OGRErr OSRExportToXML (OGRSpatialReferenceH *hSRS*, char ** *ppszRawXML*, const char * *pszDialect*)

Export coordinate system in XML format.

This function is the same as **OGRSpatialReference::exportToXML()** (p. 241).

References OSRExportToXML().

Referenced by OSRExportToXML().

12.18.2.23 OGRErr OSRFixup (OGRSpatialReferenceH *hSRS*)

Fixup as needed.

This function is the same as **OGRSpatialReference::Fixup()** (p. 242).

References OSRFixup().

Referenced by OSRFixup().

12.18.2.24 OGRErr OSRFixupOrdering (OGRSpatialReferenceH *hSRS*)

Correct parameter ordering to match CT Specification.

This function is the same as **OGRSpatialReference::FixupOrdering()** (p. 242).

References OSRFixupOrdering().

Referenced by OSRFixupOrdering().

12.18.2.25 double OSRGetAngularUnits (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch angular geographic coordinate system units.

This function is the same as **OGRSpatialReference::GetAngularUnits()** (p. 243)

References OSRGetAngularUnits().

Referenced by OSRGetAngularUnits().

12.18.2.26 const char* CPL_STDCALL OSRGetAttrValue (OGRSpatialReferenceH *hSRS*, const char * *pszKey*, int *iChild*)

Fetch indicated attribute of named node.

This function is the same as **OGRSpatialReference::GetAttrValue()** (p. 243)

References OSRGetAttrValue().

Referenced by OSRGetAttrValue().

12.18.2.27 const char* OSRGetAuthorityCode (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*)

Get the authority code for a node.

This function is the same as **OGRSpatialReference::GetAuthorityCode()** (p. 244).

References OSRGetAuthorityCode().

Referenced by OSRGetAuthorityCode().

12.18.2.28 const char* OSRGetAuthorityName (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*)

Get the authority name for a node.

This function is the same as **OGRSpatialReference::GetAuthorityName()** (p. 244).

References OSRGetAuthorityName().

Referenced by OSRGetAuthorityName().

12.18.2.29 const char* OSRGetAxis (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*, int *iAxis*, OGRAxisOrientation * *peOrientation*)

Fetch the orientation of one axis.

This method is the equivalent of the C++ method **OGRSpatialReference::GetAxis** (p. 245)

References OSRGetAxis().

Referenced by OSRGetAxis().

12.18.2.30 double OSRGetInvFlattening (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid inverse flattening.

This function is the same as **OGRSpatialReference::GetInvFlattening()** (p. 245)

References OSRGetInvFlattening().

Referenced by OSRGetInvFlattening().

12.18.2.31 double OSRGetLinearUnits (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetLinearUnits()** (p. 246)

References OSRGetLinearUnits().

Referenced by OSRGetLinearUnits().

12.18.2.32 double OSRGetNormProjParm (OGRSpatialReferenceH *hSRS*, const char * *pszName*, double *dfDefaultValue*, OGRErr * *pnErr*)

This function is the same as **OGRSpatialReference** (p. 230)::

This function is the same as **OGRSpatialReference::GetNormProjParm()** (p. 246)

References OSRGetNormProjParm().

Referenced by OSRGetNormProjParm().

12.18.2.33 double OSRGetPrimeMeridian (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch prime meridian info.

This function is the same as **OGRSpatialReference::GetPrimeMeridian()** (p. 246)

References OSRGetPrimeMeridian().

Referenced by OSRGetPrimeMeridian().

12.18.2.34 double OSRGetProjParm (OGRSpatialReferenceH *hSRS*, const char * *pszName*, double *dfDefaultValue*, OGRErr * *pnErr*)

Fetch a projection parameter value.

This function is the same as **OGRSpatialReference::GetProjParm()** (p. 247)

References OSRGetProjParm().

Referenced by OSRGetProjParm().

12.18.2.35 double OSRGetSemiMajor (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid semi major axis.

This function is the same as **OGRSpatialReference::GetSemiMajor()** (p. 247)

References OSRGetSemiMajor().

Referenced by OSRGetSemiMajor().

12.18.2.36 double OSRGetSemiMinor (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid semi minor axis.

This function is the same as **OGRSpatialReference::GetSemiMinor()** (p. 248)

References OSRGetSemiMinor().

Referenced by OSRGetSemiMinor().

12.18.2.37 double OSRGetTargetLinearUnits (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*, char ** *ppszName*)

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetTargetLinearUnits()** (p. 248)

Since

OGR 1.9.0

References OSRGetTargetLinearUnits().

Referenced by OSRGetTargetLinearUnits().

12.18.2.38 OGRErr OSRGetTOWGS84 (OGRSpatialReferenceH *hSRS*, double * *padfCoeff*, int *nCoeffCount*)

Fetch TOWGS84 parameters, if available.

This function is the same as **OGRSpatialReference::GetTOWGS84()** (p. 249).

References OSRGetTOWGS84().

Referenced by OSRGetTOWGS84().

12.18.2.39 int OSRGetUTMZone (OGRSpatialReferenceH *hSRS*, int * *pbNorth*)

Get utm zone information.

This is the same as the C++ method **OGRSpatialReference::GetUTMZone()** (p. 249)

References OSRGetUTMZone().

Referenced by OSRGetUTMZone().

12.18.2.40 OGRErr CPL_STDCALL OSRImportFromEPSG (OGRSpatialReferenceH *hSRS*, int *nCode*)

Initialize SRS based on EPSG GCS or PCS code.

This function is the same as **OGRSpatialReference::importFromEPSG()** (p. 250).

12.18.2.41 OGRErr CPL_STDCALL OSRImportFromEPSGA (OGRSpatialReferenceH *hSRS*, int *nCode*)

Initialize SRS based on EPSG GCS or PCS code.

This function is the same as **OGRSpatialReference::importFromEPSGA()** (p. 250).

12.18.2.42 OGRErr OSRImportFromERM (OGRSpatialReferenceH *hSRS*, const char * *pszProj*, const char * *pszDatum*, const char * *pszUnits*)

Create OGR WKT from ERMapper projection definitions.

This function is the same as **OGRSpatialReference::importFromERM()** (p. 251).

References OSRImportFromERM().

Referenced by OSRImportFromERM().

12.18.2.43 OGRErr OSRImportFromESRI (OGRSpatialReferenceH *hSRS*, char ** *papszPrj*)

Import coordinate system from ESRI .prj format(s).

This function is the same as the C++ method **OGRSpatialReference::importFromESRI()** (p. 251)

References OSRImportFromESRI().

Referenced by OSRImportFromESRI().

12.18.2.44 OGRErr OSRImportFromMICoordSys (OGRSpatialReferenceH *hSRS,* const char * *pszCoordSys*)

Import Mapinfo style CoordSys definition.

This method is the equivalent of the C++ method **OGRSpatialReference::importFromMICoordSys** (p. 252)

References OSRImportFromMICoordSys().

Referenced by OSRImportFromMICoordSys().

12.18.2.45 OGRErr OSRImportFromPCI (OGRSpatialReferenceH *hSRS,* const char * *pszProj,* const char * *pszUnits,* double * *padfPrjParams*)

Import coordinate system from PCI projection definition.

This function is the same as **OGRSpatialReference::importFromPCI()** (p. 254).

References OSRImportFromPCI().

Referenced by OSRImportFromPCI().

12.18.2.46 OGRErr OSRImportFromProj4 (OGRSpatialReferenceH *hSRS,* const char * *pszProj4*)

Import PROJ.4 coordinate string.

This function is the same as **OGRSpatialReference::importFromProj4()** (p. 254).

References OSRImportFromProj4().

Referenced by OSRImportFromProj4().

12.18.2.47 OGRErr OSRImportFromUrl (OGRSpatialReferenceH *hSRS,* const char * *pszUrl*)

Set spatial reference from a URL.

This function is the same as **OGRSpatialReference::importFromUrl()** (p. 255)

References OSRImportFromUrl().

Referenced by OSRImportFromUrl().

12.18.2.48 OGRErr OSRImportFromUSGS (OGRSpatialReferenceH *hSRS,* long *iProjsys,* long *iZone,* double * *padfPrjParams,* long *iDatum*)

Import coordinate system from USGS projection definition.

This function is the same as **OGRSpatialReference::importFromUSGS()** (p. 256).

References OSRImportFromUSGS().

Referenced by OSRImportFromUSGS().

12.18.2.49 OGRErr OSRImportFromWkt (OGRSpatialReferenceH *hSRS,* char ** *ppszInput*)

Import from WKT string.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. 260).

References OSRImportFromWkt().

Referenced by OSRImportFromWkt().

12.18.2.50 OGRErr OSRImportFromXML (OGRSpatialReferenceH *hSRS*, const char * *pszXML*)

Import coordinate system from XML format (GML only currently).

This function is the same as **OGRSpatialReference::importFromXML()** (p. 260).

References OSRImportFromXML().

Referenced by OSRImportFromXML().

12.18.2.51 int OSRIsCompound (OGRSpatialReferenceH *hSRS*)

Check if the coordinate system is compound.

This function is the same as **OGRSpatialReference::IsCompound()** (p. 261).

References OSRIsCompound().

Referenced by OSRIsCompound().

12.18.2.52 int OSRIsGeocentric (OGRSpatialReferenceH *hSRS*)

Check if geocentric coordinate system.

This function is the same as **OGRSpatialReference::IsGeocentric()** (p. 261).

Since

OGR 1.9.0

References OSRIsGeocentric().

Referenced by OSRIsGeocentric().

12.18.2.53 int OSRIsGeographic (OGRSpatialReferenceH *hSRS*)

Check if geographic coordinate system.

This function is the same as **OGRSpatialReference::IsGeographic()** (p. 261).

References OSRIsGeographic().

Referenced by OSRIsGeographic().

12.18.2.54 int OSRIsLocal (OGRSpatialReferenceH *hSRS*)

Check if local coordinate system.

This function is the same as **OGRSpatialReference::IsLocal()** (p. 261).

References OSRIsLocal().

Referenced by OSRIsLocal().

12.18.2.55 int OSRIsProjected (OGRSpatialReferenceH *hSRS*)

Check if projected coordinate system.

This function is the same as **OGRSpatialReference::IsProjected()** (p. 262).

References OSRIsProjected().

Referenced by OSRIsProjected().

12.18.2.56 int OSRIsSame (OGRSpatialReferenceH *hSRS1,* OGRSpatialReferenceH *hSRS2*)

Do these two spatial references describe the same system ?

This function is the same as **OGRSpatialReference::IsSame()** (p. 262).

References OSRIsSame().

Referenced by OSRIsSame().

12.18.2.57 int OSRIsSameGeogCS (OGRSpatialReferenceH *hSRS1,* OGRSpatialReferenceH *hSRS2*)

Do the GeogCS'es match?

This function is the same as **OGRSpatialReference::IsSameGeogCS()** (p. 262).

References OSRIsSameGeogCS().

Referenced by OSRIsSameGeogCS().

12.18.2.58 int OSRIsSameVertCS (OGRSpatialReferenceH *hSRS1,* OGRSpatialReferenceH *hSRS2*)

Do the VertCS'es match?

This function is the same as **OGRSpatialReference::IsSameVertCS()** (p. 262).

References OSRIsSameVertCS().

Referenced by OSRIsSameVertCS().

12.18.2.59 int OSRIsVertical (OGRSpatialReferenceH *hSRS*)

Check if vertical coordinate system.

This function is the same as **OGRSpatialReference::IsVertical()** (p. 263).

Since

OGR 1.8.0

References OSRIsVertical().

Referenced by OSRIsVertical().

12.18.2.60 OGRErr OSRMorphFromESRI (OGRSpatialReferenceH *hSRS*)

Convert in place from ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphFromESRI()** (p. 263)

References OSRMorphFromESRI().

Referenced by OSRMorphFromESRI().

12.18.2.61 OGRErr OSRMorphToESRI (OGRSpatialReferenceH *hSRS*)

Convert in place to ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphToESRI()** (p. 264)

References OSRMorphToESRI().

Referenced by OSRMorphToESRI().

12.18.2.62 OGRSpatialReferenceH CPL_STDCALL OSRNewSpatialReference (const char * *pszWKT*)

Constructor.

This function is the same as OGRSpatialReference::OGRSpatialReference()

References OGRSpatialReference::importFromWkt(), and OSRNewSpatialReference().

Referenced by OSRNewSpatialReference().

12.18.2.63 int OSRReference (OGRSpatialReferenceH *hSRS*)

Increments the reference count by one.

This function is the same as **OGRSpatialReference::Reference()** (p. 264)

References OSRReference().

Referenced by OSRReference().

12.18.2.64 void OSRRelease (OGRSpatialReferenceH *hSRS*)

Decrements the reference count by one, and destroy if zero.

This function is the same as **OGRSpatialReference::Release()** (p. 265)

References OSRRelease().

Referenced by OSRRelease().

12.18.2.65 OGRErr OSRSetACEA (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Albers Conic Equal Area

References OSRSetACEA().

Referenced by OSRSetACEA().

12.18.2.66 OGRErr OSRSetAE (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Azimuthal Equidistant

References OSRSetAE().

Referenced by OSRSetAE().

12.18.2.67 OGRErr OSRSetAngularUnits (OGRSpatialReferenceH *hSRS,* const char * *pszUnits,* double *dfInRadians*)

Set the angular units for the geographic coordinate system.

This function is the same as **OGRSpatialReference::SetAngularUnits()** (p. 265)

References OSRSetAngularUnits().

Referenced by OSRSetAngularUnits().

12.18.2.68 OGRErr CPL_STDCALL OSRSetAttrValue (OGRSpatialReferenceH *hSRS*, const char * *pszPath*, const char * *pszValue*)

Set attribute value in spatial reference.

This function is the same as **OGRSpatialReference::SetNode()** (p. 275)

References OSRSetAttrValue().

Referenced by OSRSetAttrValue().

12.18.2.69 OGRErr OSRSetAuthority (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*, const char * *pszAuthority*, int *nCode*)

Set the authority for a node.

This function is the same as **OGRSpatialReference::SetAuthority()** (p. 266).

References OSRSetAuthority().

Referenced by OSRSetAuthority().

12.18.2.70 OGRErr OSRSetBonne (OGRSpatialReferenceH *hSRS*, double *dfStandardParallel*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Bonne

References OSRSetBonne().

Referenced by OSRSetBonne().

12.18.2.71 OGRErr OSRSetCEA (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cylindrical Equal Area

References OSRSetCEA().

Referenced by OSRSetCEA().

12.18.2.72 OGRErr OSRSetCompoundCS (OGRSpatialReferenceH *hSRS*, const char * *pszName*, OGRSpatialReferenceH *hHorizSRS*, OGRSpatialReferenceH *hVertSRS*)

Setup a compound coordinate system.

This function is the same as **OGRSpatialReference::SetCompoundCS()** (p. 267)

References OSRSetCompoundCS().

Referenced by OSRSetCompoundCS().

12.18.2.73 OGRErr OSRSetCS (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cassini-Soldner

References OSRSetCS().

Referenced by OSRSetCS().

12.18.2.74 OGRErr OSRSetEC (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equidistant Conic

References OSRSetEC().

Referenced by OSRSetEC().

12.18.2.75 OGRErr OSRSetEckert (OGRSpatialReferenceH *hSRS,* int *nVariation,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert I-VI

References OSRSetEckert().

Referenced by OSRSetEckert().

12.18.2.76 OGRErr OSRSetEckertIV (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert IV

References OSRSetEckertIV().

Referenced by OSRSetEckertIV().

12.18.2.77 OGRErr OSRSetEckertVI (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert VI

References OSRSetEckertVI().

Referenced by OSRSetEckertVI().

12.18.2.78 OGRErr OSRSetEquirectangular (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular

References OSRSetEquirectangular().

Referenced by OSRSetEquirectangular().

12.18.2.79 OGRErr OSRSetEquirectangular2 (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfPseudoStdParallel1,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular generalized form

References OSRSetEquirectangular2().

Referenced by OSRSetEquirectangular2().

12.18.2.80 OGRErr CPL_STDCALL OSRSetFromUserInput (OGRSpatialReferenceH *hSRS*, const char * *pszDef*)

Set spatial reference from various text formats.

This function is the same as **OGRSpatialReference::SetFromUserInput()** (p. 268)

References OSRSetFromUserInput().

Referenced by OSRSetFromUserInput().

12.18.2.81 OGRErr OSRSetGaussSchreiberTMercator (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References OSRSetGaussSchreiberTMercator().

Referenced by OSRSetGaussSchreiberTMercator().

12.18.2.82 OGRErr OSRSetGeocCS (OGRSpatialReferenceH *hSRS*, const char * *pszName*)

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetGeocCS()** (p. 269)

Since

OGR 1.9.0

References OSRSetGeocCS().

Referenced by OSRSetGeocCS().

12.18.2.83 OGRErr OSRSetGeogCS (OGRSpatialReferenceH *hSRS*, const char * *pszGeogName*, const char * *pszDatumName*, const char * *pszSpheroidName*, double *dfSemiMajor*, double *dfInvFlattening*, const char * *pszPMName*, double *dfPMOffset*, const char * *pszAngularUnits*, double *dfConvertToRadians*)

Set geographic coordinate system.

This function is the same as **OGRSpatialReference::SetGeogCS()** (p. 270)

References OSRSetGeogCS().

Referenced by OSRSetGeogCS().

12.18.2.84 OGRErr OSRSetGEOS (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing*)

GEOS - Geostationary Satellite View

References OSRSetGEOS().

Referenced by OSRSetGEOS().

12.18.2.85 OGRErr OSRSetGH (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Goode Homolosine

References OSRSetGH().

Referenced by OSRSetGH().

12.18.2.86 OGRErr OSRSetGnomonic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gnomonic

References OSRSetGnomonic().

Referenced by OSRSetGnomonic().

12.18.2.87 OGRErr OSRSetGS (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gall Stereograpic

References OSRSetGS().

Referenced by OSRSetGS().

12.18.2.88 OGRErr OSRSetHOM (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfAzimuth,* double *dfRectToSkew,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Set a Hotine Oblique Mercator projection using azimuth angle.

Hotine Oblique Mercator using azimuth angle

This is the same as the C++ method **OGRSpatialReference::SetHOM()** (p. 271)

References OSRSetHOM().

Referenced by OSRSetHOM().

12.18.2.89 OGRErr OSRSetHOM2PNO (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfLat1,* double *dfLong1,* double *dfLat2,* double *dfLong2,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Set a Hotine Oblique Mercator projection using two points on projection centerline.

Hotine Oblique Mercator using two points on centerline

This is the same as the C++ method **OGRSpatialReference::SetHOM2PNO()** (p. 271)

References OSRSetHOM2PNO().

Referenced by OSRSetHOM2PNO().

12.18.2.90 OGRErr OSRSetIGH (OGRSpatialReferenceH *hSRS*)

Interrupted Goode Homolosine

References OSRSetIGH().

Referenced by OSRSetIGH().

12.18.2.91 OGRErr OSRSetIWMPolyconic (OGRSpatialReferenceH *hSRS,* double *dfLat1,* double *dfLat2,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

International Map of the World Polyconic

References OSRSetIWMPolyconic().

Referenced by OSRSetIWMPolyconic().

12.18.2.92 OGRErr OSRSetKrovak (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfAzimuth,* double *dfPseudoStdParallelLat,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

References OSRSetKrovak().

Referenced by OSRSetKrovak().

12.18.2.93 OGRErr OSRSetLAEA (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References OSRSetLAEA().

Referenced by OSRSetLAEA().

12.18.2.94 OGRErr OSRSetLCC (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic

References OSRSetLCC().

Referenced by OSRSetLCC().

12.18.2.95 OGRErr OSRSetLCC1SP (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References OSRSetLCC1SP().

Referenced by OSRSetLCC1SP().

12.18.2.96 OGRErr OSRSetLCCB (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References OSRSetLCCB().

Referenced by OSRSetLCCB().

12.18.2.97 OGRErr OSRSetLinearUnits (OGRSpatialReferenceH *hSRS,* const char * *pszUnits,* double *dfInMeters*)

Set the linear units for the projection.

This function is the same as **OGRSpatialReference::SetLinearUnits()** (p. 273)

References OSRSetLinearUnits().

Referenced by OSRSetLinearUnits().

12.18.2.98 OGRErr OSRSetLinearUnitsAndUpdateParameters (OGRSpatialReferenceH *hSRS,* const char * *pszUnits,* double *dfInMeters*)

Set the linear units for the projection.

This function is the same as **OGRSpatialReference::SetLinearUnitsAndUpdateParameters()** (p. 273)

References OSRSetLinearUnitsAndUpdateParameters().

Referenced by OSRSetLinearUnitsAndUpdateParameters().

12.18.2.99 OGRErr OSRSetLocalCS (OGRSpatialReferenceH *hSRS,* const char * *pszName*)

Set the user visible LOCAL_CS name.

This function is the same as **OGRSpatialReference::SetLocalCS()** (p. 274)

References OSRSetLocalCS().

Referenced by OSRSetLocalCS().

12.18.2.100 OGRErr OSRSetMC (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Miller Cylindrical

References OSRSetMC().

Referenced by OSRSetMC().

12.18.2.101 OGRErr OSRSetMercator (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mercator

References OSRSetMercator().

Referenced by OSRSetMercator().

12.18.2.102 OGRErr OSRSetMollweide (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mollweide

References OSRSetMollweide().

Referenced by OSRSetMollweide().

12.18.2.103 OGRErr OSRSetNormProjParm (OGRSpatialReferenceH *hSRS,* const char * *pszParmName,* double *dfValue*)

Set a projection parameter with a normalized value.

This function is the same as **OGRSpatialReference::SetNormProjParm()** (p. 275)

References OSRSetNormProjParm().

Referenced by OSRSetNormProjParm().

12.18.2.104 OGRErr OSRSetNZMG (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

New Zealand Map Grid

References OSRSetNZMG().

Referenced by OSRSetNZMG().

12.18.2.105 OGRErr OSRSetOrthographic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Orthographic

References OSRSetOrthographic().

Referenced by OSRSetOrthographic().

12.18.2.106 OGRErr OSRSetOS (OGRSpatialReferenceH *hSRS,* double *dfOriginLat,* double *dfCMeridian,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Oblique Stereographic

References OSRSetOS().

Referenced by OSRSetOS().

12.18.2.107 OGRErr OSRSetPolyconic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polyconic

References OSRSetPolyconic().

Referenced by OSRSetPolyconic().

12.18.2.108 OGRErr OSRSetProjCS (OGRSpatialReferenceH *hSRS,* const char * *pszName*)

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetProjCS()** (p. 276)

References OSRSetProjCS().

Referenced by OSRSetProjCS().

12.18.2.109 OGRErr OSRSetProjection (OGRSpatialReferenceH *hSRS,* const char * *pszProjection*)

Set a projection name.

This function is the same as **OGRSpatialReference::SetProjection()** (p. 276)

References OSRSetProjection().

Referenced by OSRSetProjection().

12.18.2.110 OGRErr OSRSetProjParm (OGRSpatialReferenceH *hSRS,* const char * *pszParmName,* double *dfValue*)

Set a projection parameter value.

This function is the same as **OGRSpatialReference::SetProjParm()** (p. 277)

References OSRSetProjParm().

Referenced by OSRSetProjParm().

12.18.2.111 OGRErr OSRSetPS (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polar Stereographic

References OSRSetPS().

Referenced by OSRSetPS().

12.18.2.112 OGRErr OSRSetRobinson (OGRSpatialReferenceH *hSRS,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Robinson

References OSRSetRobinson().

Referenced by OSRSetRobinson().

12.18.2.113 OGRErr OSRSetSinusoidal (OGRSpatialReferenceH *hSRS,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Sinusoidal

References OSRSetSinusoidal().

Referenced by OSRSetSinusoidal().

12.18.2.114 OGRErr OSRSetSOC (OGRSpatialReferenceH *hSRS,* double *dfLatitudeOfOrigin,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References OSRSetSOC().

Referenced by OSRSetSOC().

12.18.2.115 OGRErr OSRSetStatePlane (OGRSpatialReferenceH *hSRS,* int *nZone,* int *bNAD83*)

Set State Plane projection definition.

This function is the same as **OGRSpatialReference::SetStatePlane()** (p. 278).

12.18.2.116 OGRErr OSRSetStatePlaneWithUnits (OGRSpatialReferenceH *hSRS,* int *nZone,* int *bNAD83,* const char * *pszOverrideUnitName,* double *dfOverrideUnit*)

Set State Plane projection definition.

This function is the same as **OGRSpatialReference::SetStatePlane()** (p. 278).

12.18.2.117 OGRErr OSRSetStereographic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Stereographic

References OSRSetStereographic().

Referenced by OSRSetStereographic().

12.18.2.118 OGRErr OSRSetTargetLinearUnits (OGRSpatialReferenceH *hSRS,* const char * *pszTargetKey,* const char * *pszUnits,* double *dfInMeters*)

Set the linear units for the target node.

This function is the same as **OGRSpatialReference::SetTargetLinearUnits()** (p. 279)

Since

OGR 1.9.0

References OSRSetTargetLinearUnits().

Referenced by OSRSetTargetLinearUnits().

12.18.2.119 OGRErr OSRSetTM (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator

References OSRSetTM().

Referenced by OSRSetTM().

12.18.2.120 OGRErr OSRSetTMG (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Tunesia Mining Grid

References OSRSetTMG().

Referenced by OSRSetTMG().

12.18.2.121 OGRErr OSRSetTMSO (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References OSRSetTMSO().

Referenced by OSRSetTMSO().

12.18.2.122 OGRErr OSRSetTMVariant (OGRSpatialReferenceH *hSRS*, const char * *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator variant

References OSRSetTMVariant().

Referenced by OSRSetTMVariant().

12.18.2.123 OGRErr OSRSetTOWGS84 (OGRSpatialReferenceH *hSRS*, double *dfDX*, double *dfDY*, double *dfDZ*, double *dfEX*, double *dfEY*, double *dfEZ*, double *dfPPM*)

Set the Bursa-Wolf conversion to WGS84.

This function is the same as **OGRSpatialReference::SetTOWGS84()** (p. 280).

References OSRSetTOWGS84().

Referenced by OSRSetTOWGS84().

12.18.2.124 OGRErr OSRSetUTM (OGRSpatialReferenceH *hSRS*, int *nZone*, int *bNorth*)

Set UTM projection definition.

This is the same as the C++ method **OGRSpatialReference::SetUTM()** (p. 280)

References OSRSetUTM().

Referenced by OSRSetUTM().

12.18.2.125 OGRErr OSRSetVDG (OGRSpatialReferenceH *hSRS,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

VanDerGrinten

References OSRSetVDG().

Referenced by OSRSetVDG().

12.18.2.126 OGRErr OSRSetVertCS (OGRSpatialReferenceH *hSRS,* const char * *pszVertCSName,* const char * *pszVertDatumName,* int *nVertDatumType*)

Setup the vertical coordinate system.

This function is the same as **OGRSpatialReference::SetVertCS()** (p. 281)

Since

OGR 1.9.0

References OSRSetVertCS().

Referenced by OSRSetVertCS().

12.18.2.127 OGRErr OSRSetWagner (OGRSpatialReferenceH *hSRS,* int *nVariation,* double *dfFalseEasting,* double *dfFalseNorthing*)

Wagner I – VII

12.18.2.128 OGRErr OSRSetWellKnownGeogCS (OGRSpatialReferenceH *hSRS,* const char * *pszName*)

Set a GeogCS based on well known name.

This function is the same as **OGRSpatialReference::SetWellKnownGeogCS()** (p. 282)

References OSRSetWellKnownGeogCS().

Referenced by OSRSetWellKnownGeogCS().

12.18.2.129 OGRErr OSRStripCTParms (OGRSpatialReferenceH *hSRS*)

Strip OGC CT Parameters.

This function is the same as **OGRSpatialReference::StripCTParms()** (p. 282).

References OSRStripCTParms().

Referenced by OSRStripCTParms().

12.18.2.130 OGRErr OSRValidate (OGRSpatialReferenceH *hSRS*)

Validate SRS tokens.

This function is the same as the C++ method **OGRSpatialReference::Validate()** (p. 283).

References OSRValidate().

Referenced by OSRValidate().

12.19 ogrsf_frmts.h File Reference

```
#include "ogr_feature.h"
#include "ogr_featurestyle.h"
```

Classes

- class **OGRLayer**
- class **OGRDataSource**
- class **OGRSFDriver**
- class **OGRSFDriverRegistrar**

Functions

- void **OGRRegisterAll** ()

  Register all drivers.

12.19.1 Detailed Description

Classes related to registration of format support, and opening datasets.

Index

~OGRSpatialReference
    OGRSpatialReference, 236
~OGRStyleMgr
    OGRStyleMgr, 285
_CPLAssert
    cpl_error.h, 334
_CPLHashSet, 51
_CPLList, 51
    pData, 51
    psNext, 51
_CPLQuadTree, 52
_QuadTreeNode, 52
_sPolyExtended, 52

AddChild
    OGR_SRSNode, 75
AddFieldDefn
    OGRFeatureDefn, 111
addGeometry
    OGRGeometryCollection, 147
addGeometryDirectly
    OGRGeometryCollection, 148
    OGRMultiLineString, 195
    OGRMultiPoint, 198
    OGRMultiPolygon, 201
AddNameValue
    CPLStringList, 68
AddPart
    OGRStyleMgr, 285
addPoint
    OGRLineString, 183
addRing
    OGRPolygon, 213
addRingDirectly
    OGRPolygon, 213
AddString
    CPLStringList, 68
AddStringDirectly
    CPLStringList, 68
AddStyle
    OGRStyleMgr, 286
    OGRStyleTable, 290
addSubLineString
    OGRLineString, 183
AlterFieldDefn
    OGRLayer, 164
Append
    CPLODBCStatement, 58
AppendEscaped
    CPLODBCStatement, 59
Appendf
    CPLODBCStatement, 59
applyRemapper
    OGR_SRSNode, 76
approximateArcAngles
    OGRGeometryFactory, 157
Assign
    CPLStringList, 68
assignSpatialReference
    OGRGeometry, 129
AutoIdentifyEPSG
    OGRSpatialReference, 236
AutoLoadDrivers
    OGRSFDriverRegistrar, 226

Boundary
    OGRGeometry, 129
Buffer
    OGRGeometry, 129

CPL_LSBINT16PTR
    cpl_port.h, 354
CPL_LSBINT32PTR
    cpl_port.h, 354
CPLAddXMLChild
    cpl_minixml.h, 348
CPLAddXMLSibling
    cpl_minixml.h, 348
CPLAtof
    cpl_conv.h, 312
CPLAtofDelim
    cpl_conv.h, 313
CPLAtofM
    cpl_conv.h, 313
CPLBinaryToHex
    cpl_string.h, 358
CPLCalloc
    cpl_conv.h, 314
CPLCheckForFile
    cpl_conv.h, 314
CPLCleanTrailingSlash
    cpl_conv.h, 314
CPLCleanXMLElementName
    cpl_minixml.h, 348
CPLCloneXMLTree
    cpl_minixml.h, 348
CPLCloseShared
    cpl_conv.h, 315
CPLCorrespondingPaths
    cpl_conv.h, 315

CPLCreateXMLElementAndValue
cpl_minixml.h, 349
CPLCreateXMLNode
cpl_minixml.h, 349
CPLDebug
cpl_error.h, 334
CPLDecToPackedDMS
cpl_conv.h, 316
CPLDestroyXMLNode
cpl_minixml.h, 349
CPLDumpSharedList
cpl_conv.h, 316
CPLEmergencyError
cpl_error.h, 334
CPLEncodingCharSize
cpl_string.h, 359
CPLError
cpl_error.h, 334
CPLErrorContext, 53
CPLErrorReset
cpl_error.h, 335
CPLEscapeString
cpl_string.h, 359
CPLExtractRelativePath
cpl_conv.h, 316
CPLFGets
cpl_conv.h, 317
CPLForceToASCII
cpl_string.h, 360
CPLFormCIFilename
cpl_conv.h, 317
CPLFormFilename
cpl_conv.h, 317
CPLGenerateTempFilename
cpl_conv.h, 318
CPLGetBasename
cpl_conv.h, 318
CPLGetConfigOption
cpl_conv.h, 318
CPLGetCurrentDir
cpl_conv.h, 319
CPLGetDirname
cpl_conv.h, 319
CPLGetErrorHandlerUserData
cpl_error.h, 335
CPLGetExecPath
cpl_conv.h, 320
CPLGetExtension
cpl_conv.h, 320
CPLGetFilename
cpl_conv.h, 320
CPLGetLastErrorMsg
cpl_error.h, 335
CPLGetLastErrorNo
cpl_error.h, 335
CPLGetLastErrorType
cpl_error.h, 335
CPLGetPath
cpl_conv.h, 321
CPLGetSharedList
cpl_conv.h, 321
CPLGetSymbol
cpl_conv.h, 321
CPLGetValueType
cpl_string.h, 360
CPLGetXMLNode
cpl_minixml.h, 350
CPLGetXMLValue
cpl_minixml.h, 350
CPLHTTPDestroyResult
cpl_http.h, 341
CPLHTTPEnabled
cpl_http.h, 341
CPLHTTPFetch
cpl_http.h, 342
CPLHTTPParseMultipartMime
cpl_http.h, 342
CPLHTTPResult, 53
nDataLen, 53
nMimePartCount, 53
nStatus, 53
pabyData, 53
papszHeaders, 53
pasMimePart, 54
pszContentType, 54
pszErrBuf, 54
CPLHashSetDestroy
cpl_hash_set.h, 338
CPLHashSetEqualPointer
cpl_hash_set.h, 338
CPLHashSetEqualStr
cpl_hash_set.h, 338
CPLHashSetForeach
cpl_hash_set.h, 338
CPLHashSetHashPointer
cpl_hash_set.h, 339
CPLHashSetHashStr
cpl_hash_set.h, 339
CPLHashSetInsert
cpl_hash_set.h, 339
CPLHashSetLookup
cpl_hash_set.h, 339
CPLHashSetNew
cpl_hash_set.h, 340
CPLHashSetRemove
cpl_hash_set.h, 340
CPLHashSetSize
cpl_hash_set.h, 340
CPLHexToBinary
cpl_string.h, 360
CPLIsFilenameRelative
cpl_conv.h, 322
CPLIsUTF8
cpl_string.h, 360
CPLKeywordParser, 54
CPLList

cpl_list.h, 343
CPLListAppend
cpl_list.h, 344
CPLListCount
cpl_list.h, 344
CPLListDestroy
cpl_list.h, 344
CPLListGet
cpl_list.h, 344
CPLListGetData
cpl_list.h, 344
CPLListGetLast
cpl_list.h, 345
CPLListGetNext
cpl_list.h, 345
CPLListInsert
cpl_list.h, 345
CPLListRemove
cpl_list.h, 346
CPLLocaleC, 54
CPLMalloc
cpl_conv.h, 322
CPLMimePart, 54
nDataLen, 55
pabyData, 55
papszHeaders, 55
CPLMutexHolder, 55
CPLODBCDriverInstaller, 55
InstallDriver, 56
RemoveDriver, 56
CPLODBCSession, 56
EstablishSession, 57
GetLastError, 57
CPLODBCStatement, 57
Append, 58
AppendEscaped, 59
Appendf, 59
Clear, 59
DumpResult, 59
ExecuteSQL, 59
Fetch, 60
GetColCount, 60
GetColData, 60, 61
GetColId, 61
GetColName, 61
GetColNullable, 61
GetColPrecision, 62
GetColSize, 62
GetColType, 62
GetColTypeName, 63
GetColumns, 63
GetPrimaryKeys, 63
GetTables, 64
GetTypeMapping, 64
GetTypeName, 64
CPLOpenShared
cpl_conv.h, 323
CPLPackedDMSToDec
cpl_conv.h, 323
CPLParseNameValue
cpl_string.h, 361
CPLParseXMLFile
cpl_minixml.h, 351
CPLParseXMLString
cpl_minixml.h, 351
CPLPopErrorHandler
cpl_error.h, 336
CPLPrintDouble
cpl_conv.h, 324
CPLPrintInt32
cpl_conv.h, 324
CPLPrintPointer
cpl_conv.h, 324
CPLPrintString
cpl_conv.h, 325
CPLPrintStringFill
cpl_conv.h, 325
CPLPrintTime
cpl_conv.h, 325
CPLPrintUIntBig
cpl_conv.h, 326
CPLProjectRelativeFilename
cpl_conv.h, 326
CPLPushErrorHandler
cpl_error.h, 336
CPLPushErrorHandlerEx
cpl_error.h, 336
CPLQuadTreeCreate
cpl_quad_tree.h, 355
CPLQuadTreeDestroy
cpl_quad_tree.h, 356
CPLQuadTreeForeach
cpl_quad_tree.h, 356
CPLQuadTreeGetAdvisedMaxDepth
cpl_quad_tree.h, 356
CPLQuadTreeInsert
cpl_quad_tree.h, 356
CPLQuadTreeSearch
cpl_quad_tree.h, 356
CPLQuadTreeSetBucketCapacity
cpl_quad_tree.h, 357
CPLQuadTreeSetMaxDepth
cpl_quad_tree.h, 357
CPLReadLine
cpl_conv.h, 327
CPLReadLine2L
cpl_conv.h, 327
CPLReadLineL
cpl_conv.h, 327
CPLRealloc
cpl_conv.h, 328
CPLRecode
cpl_string.h, 361
CPLRecodeFromWChar
cpl_string.h, 362
CPLRecodeToWChar

cpl_string.h, 362
CPLRectObj, 65
CPLRemoveXMLChild
cpl_minixml.h, 351
CPLResetExtension
cpl_conv.h, 328
CPLScanDouble
cpl_conv.h, 328
CPLScanLong
cpl_conv.h, 329
CPLScanPointer
cpl_conv.h, 329
CPLScanString
cpl_conv.h, 329
CPLScanUIntBig
cpl_conv.h, 329
CPLScanULong
cpl_conv.h, 330
CPLSearchXMLNode
cpl_minixml.h, 352
CPLSerializeXMLTree
cpl_minixml.h, 352
CPLSerializeXMLTreeToFile
cpl_minixml.h, 352
CPLSetConfigOption
cpl_conv.h, 330
CPLSetErrorHandler
cpl_error.h, 336
CPLSetErrorHandlerEx
cpl_error.h, 337
CPLSetThreadLocalConfigOption
cpl_conv.h, 330
CPLSetXMLValue
cpl_minixml.h, 353
CPLSharedFileInfo, 65
CPLStdCallThreadInfo, 65
CPLStrdup
cpl_conv.h, 331
CPLString, 65
FormatC, 65
ifind, 66
tolower, 66
toupper, 66
Trim, 67
CPLStringList, 67
AddNameValue, 68
AddString, 68
AddStringDirectly, 68
Assign, 68
CPLStringList, 68
Clear, 69
Count, 69
CPLStringList, 68
FetchBoolean, 69
FetchNameValue, 69
FetchNameValueDef, 70
FindName, 70
InsertString, 70
InsertStringDirectly, 70
operator[], 71
SetNameValue, 71
Sort, 71
StealList, 71
CPLStripXMLNamespace
cpl_minixml.h, 353
CPLStrlcat
cpl_string.h, 363
CPLStrlcpy
cpl_string.h, 363
CPLStrlwr
cpl_conv.h, 331
CPLStrnlen
cpl_string.h, 364
CPLStrtod
cpl_conv.h, 331
CPLStrtodDelim
cpl_conv.h, 332
CPLStrtof
cpl_conv.h, 332
CPLStrtofDelim
cpl_conv.h, 332
CPLURLAddKVP
cpl_string.h, 364
CPLURLGetValue
cpl_string.h, 365
CPLUnescapeString
cpl_string.h, 364
CPLUnlinkTree
cpl_conv.h, 333
CPLXMLNode, 72
cpl_minixml.h, 347
eType, 72
psChild, 72
psNext, 73
pszValue, 73
CPLXMLNodeType
cpl_minixml.h, 347
CSLCount
cpl_string.h, 365
CSLDestroy
cpl_string.h, 365
CSLDuplicate
cpl_string.h, 366
CSLFindName
cpl_string.h, 366
CSLFindString
cpl_string.h, 366
CSLLoad
cpl_string.h, 366
CSLLoad2
cpl_string.h, 367
CSLMerge
cpl_string.h, 367
CSLPartialFindString
cpl_string.h, 367
CSLSetNameValue

cpl_string.h, 368
CSLSetNameValueSeparator
cpl_string.h, 368
CSLTestBoolean
cpl_string.h, 368
CSLTokenizeString2
cpl_string.h, 369
CXT_Attribute
cpl_minixml.h, 348
CXT_Comment
cpl_minixml.h, 348
CXT_Element
cpl_minixml.h, 347
CXT_Literal
cpl_minixml.h, 348
CXT_Text
cpl_minixml.h, 347
CachedConnection, 52
CachedDirList, 52
CachedFileProp, 52
CachedRegion, 52
Centroid
OGRGeometry, 130
Clear
CPLODBCStatement, 59
CPLStringList, 69
OGRSpatialReference, 236
Clone
OGR_SRSNode, 76
OGRFeature, 96
OGRFeatureDefn, 111
OGRSpatialReference, 236
OGRStyleTable, 290
clone
OGRGeometry, 130
OGRGeometryCollection, 148
OGRLinearRing, 179
OGRLineString, 184
OGRMultiLineString, 196
OGRMultiPoint, 199
OGRMultiPolygon, 201
OGRPoint, 205
OGRPolygon, 213
CloneGeogCS
OGRSpatialReference, 237
closeRings
OGRGeometry, 130
OGRGeometryCollection, 148
OGRLinearRing, 179
OGRPolygon, 213
Contains
OGRGeometry, 130
ConvexHull
OGRGeometry, 131
CopyDataSource
OGRSFDriver, 223
CopyGeogCSFrom
OGRSpatialReference, 237
CopyLayer
OGRDataSource, 87
Count
CPLStringList, 69
cpl_conv.h, 311
CPLAtof, 312
CPLAtofDelim, 313
CPLAtofM, 313
CPLCalloc, 314
CPLCheckForFile, 314
CPLCleanTrailingSlash, 314
CPLCloseShared, 315
CPLCorrespondingPaths, 315
CPLDecToPackedDMS, 316
CPLDumpSharedList, 316
CPLExtractRelativePath, 316
CPLFGets, 317
CPLFormCIFilename, 317
CPLFormFilename, 317
CPLGenerateTempFilename, 318
CPLGetBasename, 318
CPLGetConfigOption, 318
CPLGetCurrentDir, 319
CPLGetDirname, 319
CPLGetExecPath, 320
CPLGetExtension, 320
CPLGetFilename, 320
CPLGetPath, 321
CPLGetSharedList, 321
CPLGetSymbol, 321
CPLIsFilenameRelative, 322
CPLMalloc, 322
CPLOpenShared, 323
CPLPackedDMSToDec, 323
CPLPrintDouble, 324
CPLPrintInt32, 324
CPLPrintPointer, 324
CPLPrintString, 325
CPLPrintStringFill, 325
CPLPrintTime, 325
CPLPrintUIntBig, 326
CPLProjectRelativeFilename, 326
CPLReadLine, 327
CPLReadLine2L, 327
CPLReadLineL, 327
CPLRealloc, 328
CPLResetExtension, 328
CPLScanDouble, 328
CPLScanLong, 329
CPLScanPointer, 329
CPLScanString, 329
CPLScanUIntBig, 329
CPLScanULong, 330
CPLSetConfigOption, 330
CPLSetThreadLocalConfigOption, 330
CPLStrdup, 331
CPLStrlwr, 331
CPLStrtod, 331

CPLStrtodDelim, 332
CPLStrtof, 332
CPLStrtofDelim, 332
CPLUnlinkTree, 333
cpl_error.h, 333
_CPLAssert, 334
CPLDebug, 334
CPLEmergencyError, 334
CPLError, 334
CPLErrorReset, 335
CPLGetErrorHandlerUserData, 335
CPLGetLastErrorMsg, 335
CPLGetLastErrorNo, 335
CPLGetLastErrorType, 335
CPLPopErrorHandler, 336
CPLPushErrorHandler, 336
CPLPushErrorHandlerEx, 336
CPLSetErrorHandler, 336
CPLSetErrorHandlerEx, 337
cpl_hash_set.h, 337
CPLHashSetDestroy, 338
CPLHashSetEqualPointer, 338
CPLHashSetEqualStr, 338
CPLHashSetForeach, 338
CPLHashSetHashPointer, 339
CPLHashSetHashStr, 339
CPLHashSetInsert, 339
CPLHashSetLookup, 339
CPLHashSetNew, 340
CPLHashSetRemove, 340
CPLHashSetSize, 340
cpl_http.h, 341
CPLHTTPDestroyResult, 341
CPLHTTPEnabled, 341
CPLHTTPFetch, 342
CPLHTTPParseMultipartMime, 342
cpl_list.h, 343
CPLList, 343
CPLListAppend, 344
CPLListCount, 344
CPLListDestroy, 344
CPLListGet, 344
CPLListGetData, 344
CPLListGetLast, 345
CPLListGetNext, 345
CPLListInsert, 345
CPLListRemove, 346
cpl_minixml.h, 346
CPLAddXMLChild, 348
CPLAddXMLSibling, 348
CPLCleanXMLElementName, 348
CPLCloneXMLTree, 348
CPLCreateXMLElementAndValue, 349
CPLCreateXMLNode, 349
CPLDestroyXMLNode, 349
CPLGetXMLNode, 350
CPLGetXMLValue, 350
CPLParseXMLFile, 351
CPLParseXMLString, 351
CPLRemoveXMLChild, 351
CPLSearchXMLNode, 352
CPLSerializeXMLTree, 352
CPLSerializeXMLTreeToFile, 352
CPLSetXMLValue, 353
CPLStripXMLNamespace, 353
CPLXMLNode, 347
CPLXMLNodeType, 347
CXT_Attribute, 348
CXT_Comment, 348
CXT_Element, 347
CXT_Literal, 348
CXT_Text, 347
cpl_odbc.h, 354
cpl_port.h, 354
CPL_LSBINT16PTR, 354
CPL_LSBINT32PTR, 354
cpl_quad_tree.h, 355
CPLQuadTreeCreate, 355
CPLQuadTreeDestroy, 356
CPLQuadTreeForeach, 356
CPLQuadTreeGetAdvisedMaxDepth, 356
CPLQuadTreeInsert, 356
CPLQuadTreeSearch, 356
CPLQuadTreeSetBucketCapacity, 357
CPLQuadTreeSetMaxDepth, 357
cpl_string.h, 357
CPLBinaryToHex, 358
CPLEncodingCharSize, 359
CPLEscapeString, 359
CPLForceToASCII, 360
CPLGetValueType, 360
CPLHexToBinary, 360
CPLIsUTF8, 360
CPLParseNameValue, 361
CPLRecode, 361
CPLRecodeFromWChar, 362
CPLRecodeToWChar, 362
CPLStrlcat, 363
CPLStrlcpy, 363
CPLStrnlen, 364
CPLURLAddKVP, 364
CPLURLGetValue, 365
CPLUnescapeString, 364
CSLCount, 365
CSLDestroy, 365
CSLDuplicate, 366
CSLFindName, 366
CSLFindString, 366
CSLLoad, 366
CSLLoad2, 367
CSLMerge, 367
CSLPartialFindString, 367
CSLSetNameValue, 368
CSLSetNameValueSeparator, 368
CSLTestBoolean, 368
CSLTokenizeString2, 369

cpl_vsi.h, 370
VSIFCloseL, 371
VSIFEofL, 372
VSIFFlushL, 372
VSIFOpenL, 373
VSIFPrintfL, 373
VSIFReadL, 374
VSIFReadMultiRangeL, 374
VSIFSeekL, 375
VSIFTellL, 375
VSIFTruncateL, 375
VSIFWriteL, 376
VSIFileFromMemBuffer, 372
VSIGetMemFileBuffer, 376
VSIInstallCurlFileHandler, 377
VSIInstallGZipFileHandler, 377
VSIInstallMemFileHandler, 377
VSIInstallSparseFileHandler, 378
VSIInstallStdinHandler, 379
VSIInstallStdoutHandler, 379
VSIInstallSubFileHandler, 379
VSIInstallTarFileHandler, 380
VSIInstallZipFileHandler, 380
VSIIsCaseSensitiveFS, 381
VSIMalloc2, 381
VSIMalloc3, 381
VSIMkdir, 381
VSIReadDir, 382
VSIRename, 382
VSIRmdir, 383
VSIStatExL, 383
VSIStatL, 384
VSIUnlink, 384
CreateDataSource
OGRSFDriver, 223
CreateFeature
OGRFeature, 97
OGRLayer, 165
CreateField
OGRLayer, 165
createFromFgf
OGRGeometryFactory, 157
createFromGML
OGRGeometryFactory, 158
createFromWkb
OGRGeometryFactory, 158
createFromWkt
OGRGeometryFactory, 159
createGeometry
OGRGeometryFactory, 159
CreateLayer
OGRDataSource, 87
Crosses
OGRGeometry, 131
ctb, 73
curfile_info, 73

DefaultCSVFileNameTLS, 74
DeleteDataSource
OGRSFDriver, 224
DeleteFeature
OGRLayer, 165
DeleteField
OGRLayer, 166
DeleteFieldDefn
OGRFeatureDefn, 111
DeleteLayer
OGRDataSource, 88
Dereference
OGRDataSource, 88
OGRFeatureDefn, 112
OGRLayer, 166
OGRSpatialReference, 237
DeregisterDriver
OGRSFDriverRegistrar, 227
DestroyCT
OGRCoordinateTransformation, 82
DestroyChild
OGR_SRSNode, 76
DestroyDataSource
OGRDataSource, 88
DestroyFeature
OGRFeature, 97
destroyGeometry
OGRGeometryFactory, 160
DestroySpatialReference
OGRSpatialReference, 237
Difference
OGRGeometry, 131
Disjoint
OGRGeometry, 132
Distance
OGRGeometry, 132
DumpReadable
OGRFeature, 97
dumpReadable
OGRGeometry, 132
DumpResult
CPLODBCStatement, 59

EPSGTreatsAsLatLong
OGRSpatialReference, 238
eType
CPLXMLNode, 72
empty
OGRGeometry, 133
OGRGeometryCollection, 148
OGRLineString, 184
OGRPoint, 205
OGRPolygon, 214
EndPoint
OGRCurve, 84
OGRLineString, 184
Equal
OGRFeature, 98
Equals
OGRGeometry, 133
OGRGeometryCollection, 149

OGRLineString, 184
OGRPoint, 205
OGRPolygon, 214
errHandler, 74
EstablishSession
CPLODBCSession, 57
ExecuteSQL
CPLODBCStatement, 59
OGRDataSource, 88
exportToERM
OGRSpatialReference, 238
exportToGML
OGRGeometry, 133
exportToJson
OGRGeometry, 134
exportToKML
OGRGeometry, 134
exportToMICoordSys
OGRSpatialReference, 238
exportToPCI
OGRSpatialReference, 239
exportToPanorama
OGRSpatialReference, 239
exportToPrettyWkt
OGRSpatialReference, 240
exportToProj4
OGRSpatialReference, 240
exportToUSGS
OGRSpatialReference, 240
exportToWkb
OGRGeometry, 134
OGRGeometryCollection, 149
OGRLinearRing, 179
OGRLineString, 185
OGRPoint, 205
OGRPolygon, 214
exportToWkt
OGR_SRSNode, 77
OGRGeometry, 135
OGRGeometryCollection, 149
OGRLineString, 185
OGRMultiLineString, 196
OGRMultiPoint, 199
OGRMultiPolygon, 202
OGRPoint, 206
OGRPolygon, 214
OGRSpatialReference, 241
exportToXML
OGRSpatialReference, 241

Fetch
CPLODBCStatement, 60
FetchBoolean
CPLStringList, 69
FetchNameValue
CPLStringList, 69
FetchNameValueDef
CPLStringList, 70
file_in_zip_read_info_s, 74
Find
OGRStyleTable, 290
FindChild
OGR_SRSNode, 77
FindFileTLS, 74
FindName
CPLStringList, 70
FindProjParm
OGRSpatialReference, 241
Fixup
OGRSpatialReference, 242
FixupOrdering
OGRSpatialReference, 242
flattenTo2D
OGRGeometry, 135
OGRGeometryCollection, 150
OGRLineString, 185
OGRPoint, 206
OGRPolygon, 215
forceToMultiLineString
OGRGeometryFactory, 160
forceToMultiPoint
OGRGeometryFactory, 160
forceToMultiPolygon
OGRGeometryFactory, 161
forceToPolygon
OGRGeometryFactory, 161
FormatC
CPLString, 65

GDAL_CHECK_VERSION
ogr_core.h, 484
GDALCheckVersion
ogr_core.h, 487
GZipSnapshot, 74
get_Area
OGRGeometryCollection, 150
OGRLinearRing, 179
OGRMultiPolygon, 202
OGRPolygon, 215
OGRSurface, 293
get_IsClosed
OGRCurve, 84
get_Length
OGRCurve, 84
OGRGeometryCollection, 150
OGRLineString, 186
GetAngularUnits
OGRSpatialReference, 242
GetAttrNode
OGRSpatialReference, 243
GetAttrValue
OGRSpatialReference, 243
GetAuthorityCode
OGRSpatialReference, 244
GetAuthorityName
OGRSpatialReference, 244
GetAxis
OGRSpatialReference, 245

getBoundary
OGRGeometry, 135
GetChild
OGR_SRSNode, 77
GetChildCount
OGR_SRSNode, 78
GetColCount
CPLODBCStatement, 60
GetColData
CPLODBCStatement, 60, 61
GetColId
CPLODBCStatement, 61
GetColName
CPLODBCStatement, 61
GetColNullable
CPLODBCStatement, 61
GetColPrecision
CPLODBCStatement, 62
GetColSize
CPLODBCStatement, 62
GetColType
CPLODBCStatement, 62
GetColTypeName
CPLODBCStatement, 63
GetColumns
CPLODBCStatement, 63
getCoordinateDimension
OGRGeometry, 135
GetDefnRef
OGRFeature, 98
getDimension
OGRGeometry, 136
OGRGeometryCollection, 150
OGRLineString, 186
OGRPoint, 206
OGRPolygon, 215
GetDriver
OGRDataSource, 89
OGRSFDriverRegistrar, 227
GetDriverByName
OGRSFDriverRegistrar, 227
GetDriverCount
OGRSFDriverRegistrar, 228
getEnvelope
OGRGeometry, 136
OGRGeometryCollection, 151
OGRLineString, 186
OGRPoint, 206, 207
OGRPolygon, 215, 216
GetExtension
OGRSpatialReference, 245
GetExtent
OGRGenSQLResultsLayer, 122
OGRLayer, 166
getExteriorRing
OGRPolygon, 216
GetFID
OGRFeature, 98
GetFIDColumn
OGRLayer, 168
GetFeature
OGRGenSQLResultsLayer, 122
OGRLayer, 167
GetFeatureCount
OGRGenSQLResultsLayer, 123
OGRLayer, 168
GetFieldAsBinary
OGRFeature, 98
GetFieldAsDateTime
OGRFeature, 99
GetFieldAsDouble
OGRFeature, 99
GetFieldAsDoubleList
OGRFeature, 100
GetFieldAsInteger
OGRFeature, 100
GetFieldAsIntegerList
OGRFeature, 100
GetFieldAsString
OGRFeature, 101
GetFieldAsStringList
OGRFeature, 101
GetFieldCount
OGRFeature, 102
OGRFeatureDefn, 112
GetFieldDefn
OGRFeatureDefn, 112
GetFieldDefnRef
OGRFeature, 102
GetFieldIndex
OGRFeature, 102
OGRFeatureDefn, 112
GetFieldTypeName
OGRFieldDefn, 117
GetGeomType
OGRFeatureDefn, 113
OGRLayer, 168
GetGeometryColumn
OGRLayer, 168
getGeometryName
OGRGeometry, 137
OGRGeometryCollection, 151
OGRLinearRing, 180
OGRLineString, 187
OGRMultiLineString, 196
OGRMultiPoint, 199
OGRMultiPolygon, 202
OGRPoint, 207
OGRPolygon, 216
GetGeometryRef
OGRFeature, 102
getGeometryRef
OGRGeometryCollection, 152
getGeometryType
OGRGeometry, 137
OGRGeometryCollection, 152

OGRLineString, 187
OGRMultiLineString, 197
OGRMultiPoint, 199
OGRMultiPolygon, 202
OGRPoint, 207
OGRPolygon, 217
GetInfo
OGRLayer, 169
getInteriorRing
OGRPolygon, 217
GetInvFlattening
OGRSpatialReference, 245
GetJustify
OGRFieldDefn, 118
GetLastError
CPLODBCSession, 57
GetLayer
OGRDataSource, 89
GetLayerByName
OGRDataSource, 89
GetLayerCount
OGRDataSource, 90
GetLayerDefn
OGRGenSQLResultsLayer, 123
OGRLayer, 169
GetLinearUnits
OGRSpatialReference, 246
GetName
OGRDataSource, 90
OGRFeatureDefn, 113
OGRLayer, 169
OGRSFDriver, 224
GetNameRef
OGRFieldDefn, 118
GetNextFeature
OGRGenSQLResultsLayer, 123
OGRLayer, 170
GetNode
OGR_SRSNode, 78
GetNormProjParm
OGRSpatialReference, 246
getNumGeometries
OGRGeometryCollection, 152
getNumInteriorRings
OGRPolygon, 217
getNumPoints
OGRLineString, 187
GetOpenDS
OGRSFDriverRegistrar, 228
GetOpenDSCount
OGRSFDriverRegistrar, 228
GetPart
OGRStyleMgr, 286
GetPartCount
OGRStyleMgr, 286
getPoint
OGRLineString, 187
getPoints
OGRLineString, 188
GetPrecision
OGRFieldDefn, 118
GetPrimaryKeys
CPLODBCStatement, 63
GetPrimeMeridian
OGRSpatialReference, 246
GetProjParm
OGRSpatialReference, 247
GetRawFieldRef
OGRFeature, 103
GetRefCount
OGRDataSource, 90
OGRLayer, 170
GetReferenceCount
OGRFeatureDefn, 113
OGRSpatialReference, 247
GetRegistrar
OGRSFDriverRegistrar, 228
GetSemiMajor
OGRSpatialReference, 247
GetSemiMinor
OGRSpatialReference, 248
GetSourceCS
OGRCoordinateTransformation, 82
OGRProj4CT, 221
GetSpatialFilter
OGRGenSQLResultsLayer, 123
OGRLayer, 170
GetSpatialRef
OGRGenSQLResultsLayer, 124
OGRLayer, 170
getSpatialReference
OGRGeometry, 137
GetStyleByName
OGRStyleMgr, 287
GetStyleName
OGRStyleMgr, 287
OGRStyleTable, 291
GetStyleString
OGRFeature, 103
OGRStyleMgr, 287
GetStyleTable
OGRDataSource, 90
OGRLayer, 171
GetSummaryRefCount
OGRDataSource, 91
GetTOWGS84
OGRSpatialReference, 248
GetTables
CPLODBCStatement, 64
GetTargetCS
OGRCoordinateTransformation, 82
OGRProj4CT, 221
GetTargetLinearUnits
OGRSpatialReference, 248
GetType
OGRFieldDefn, 118

GetTypeMapping
CPLODBCStatement, 64
GetTypeName
CPLODBCStatement, 64
GetUTMZone
OGRSpatialReference, 249
GetUsedFields
OGRFeatureQuery, 115
GetValue
OGR_SRSNode, 79
GetWidth
OGRFieldDefn, 119
getX
OGRLineString, 188
OGRPoint, 207
getY
OGRLineString, 189
OGRPoint, 208
getZ
OGRLineString, 189
OGRPoint, 208

haveGEOS
OGRGeometryFactory, 161

ifind
CPLString, 66
importFromDict
OGRSpatialReference, 249
importFromEPSG
OGRSpatialReference, 250
importFromEPSGA
OGRSpatialReference, 250
importFromERM
OGRSpatialReference, 251
importFromESRI
OGRSpatialReference, 251
importFromMICoordSys
OGRSpatialReference, 251
importFromOzi
OGRSpatialReference, 252
importFromPCI
OGRSpatialReference, 254
importFromPanorama
OGRSpatialReference, 252
importFromProj4
OGRSpatialReference, 254
importFromURN
OGRSpatialReference, 256
importFromUSGS
OGRSpatialReference, 256
importFromUrl
OGRSpatialReference, 255
importFromWMSAUTO
OGRSpatialReference, 260
importFromWkb
OGRGeometry, 138
OGRGeometryCollection, 153
OGRLinearRing, 180
OGRLineString, 189
OGRPoint, 208
OGRPolygon, 217
importFromWkt
OGR_SRSNode, 79
OGRGeometry, 138
OGRGeometryCollection, 153
OGRLineString, 190
OGRMultiLineString, 197
OGRMultiPoint, 200
OGRMultiPolygon, 203
OGRPoint, 209
OGRPolygon, 218
OGRSpatialReference, 259
importFromXML
OGRSpatialReference, 260
InitFromFeature
OGRStyleMgr, 287
InitStyleString
OGRStyleMgr, 288
InsertChild
OGR_SRSNode, 79
InsertString
CPLStringList, 70
InsertStringDirectly
CPLStringList, 70
InstallDriver
CPLODBCDriverInstaller, 56
Intersection
OGRGeometry, 138
Intersects
OGRGeometry, 139
isClockwise
OGRLinearRing, 180
IsCompound
OGRSpatialReference, 260
IsEmpty
OGRGeometry, 139
OGRGeometryCollection, 153
OGRLineString, 190
OGRPoint, 209
OGRPolygon, 218
IsExist
OGRStyleTable, 291
IsFieldSet
OGRFeature, 103
IsGeocentric
OGRSpatialReference, 261
IsGeographic
OGRSpatialReference, 261
IsGeometryIgnored
OGRFeatureDefn, 113
IsIgnored
OGRFieldDefn, 119
IsLocal
OGRSpatialReference, 261
IsProjected
OGRSpatialReference, 261

IsRing
OGRGeometry, 139
IsSame
OGRSpatialReference, 262
IsSameGeogCS
OGRSpatialReference, 262
IsSameVertCS
OGRSpatialReference, 262
IsSimple
OGRGeometry, 140
IsStyleIgnored
OGRFeatureDefn, 114
IsValid
OGRGeometry, 140
IsVertical
OGRSpatialReference, 263

linkedlist_data_s, 74
linkedlist_datablock_internal_s, 74
LoadStyleTable
OGRStyleTable, 291

MakeValueSafe
OGR_SRSNode, 80
ModifyStyle
OGRStyleTable, 291
morphFromESRI
OGRSpatialReference, 263
morphToESRI
OGRSpatialReference, 264

nDataLen
CPLHTTPResult, 53
CPLMimePart, 55
nMimePartCount
CPLHTTPResult, 53
nStatus
CPLHTTPResult, 53

OCTDestroyCoordinateTransformation
ogr_srs_api.h, 496
OCTNewCoordinateTransformation
ogr_srs_api.h, 496
OFTBinary
ogr_core.h, 486
OFTDate
ogr_core.h, 486
OFTDateTime
ogr_core.h, 486
OFTInteger
ogr_core.h, 486
OFTIntegerList
ogr_core.h, 486
OFTReal
ogr_core.h, 486
OFTRealList
ogr_core.h, 486
OFTString
ogr_core.h, 486
OFTStringList
ogr_core.h, 486
OFTTime
ogr_core.h, 486
OFTWideString
ogr_core.h, 486
OFTWideStringList
ogr_core.h, 486
OGR_DS_CopyLayer
ogr_api.h, 397
OGR_DS_CreateLayer
ogr_api.h, 398
OGR_DS_DeleteLayer
ogr_api.h, 398
OGR_DS_Destroy
ogr_api.h, 399
OGR_DS_ExecuteSQL
ogr_api.h, 399
OGR_DS_GetDriver
ogr_api.h, 400
OGR_DS_GetLayer
ogr_api.h, 400
OGR_DS_GetLayerByName
ogr_api.h, 400
OGR_DS_GetLayerCount
ogr_api.h, 401
OGR_DS_GetName
ogr_api.h, 401
OGR_DS_ReleaseResultSet
ogr_api.h, 401
OGR_DS_SyncToDisk
ogr_api.h, 402
OGR_DS_TestCapability
ogr_api.h, 402
OGR_Dr_CopyDataSource
ogr_api.h, 395
OGR_Dr_CreateDataSource
ogr_api.h, 395
OGR_Dr_DeleteDataSource
ogr_api.h, 396
OGR_Dr_GetName
ogr_api.h, 396
OGR_Dr_Open
ogr_api.h, 396
OGR_Dr_TestCapability
ogr_api.h, 397
OGR_F_Clone
ogr_api.h, 402
OGR_F_Create
ogr_api.h, 403
OGR_F_Destroy
ogr_api.h, 403
OGR_F_DumpReadable
ogr_api.h, 403
OGR_F_Equal
ogr_api.h, 404
OGR_F_GetDefnRef
ogr_api.h, 404

OGR_F_GetFID
ogr_api.h, 404
OGR_F_GetFieldAsBinary
ogr_api.h, 405
OGR_F_GetFieldAsDateTime
ogr_api.h, 405
OGR_F_GetFieldAsDouble
ogr_api.h, 406
OGR_F_GetFieldAsDoubleList
ogr_api.h, 406
OGR_F_GetFieldAsInteger
ogr_api.h, 406
OGR_F_GetFieldAsIntegerList
ogr_api.h, 407
OGR_F_GetFieldAsString
ogr_api.h, 407
OGR_F_GetFieldAsStringList
ogr_api.h, 407
OGR_F_GetFieldCount
ogr_api.h, 408
OGR_F_GetFieldDefnRef
ogr_api.h, 408
OGR_F_GetFieldIndex
ogr_api.h, 408
OGR_F_GetGeometryRef
ogr_api.h, 409
OGR_F_GetRawFieldRef
ogr_api.h, 409
OGR_F_GetStyleString
ogr_api.h, 409
OGR_F_IsFieldSet
ogr_api.h, 410
OGR_F_SetFID
ogr_api.h, 410
OGR_F_SetFieldBinary
ogr_api.h, 411
OGR_F_SetFieldDateTime
ogr_api.h, 411
OGR_F_SetFieldDouble
ogr_api.h, 411
OGR_F_SetFieldDoubleList
ogr_api.h, 412
OGR_F_SetFieldInteger
ogr_api.h, 412
OGR_F_SetFieldIntegerList
ogr_api.h, 412
OGR_F_SetFieldRaw
ogr_api.h, 412
OGR_F_SetFieldString
ogr_api.h, 413
OGR_F_SetFieldStringList
ogr_api.h, 413
OGR_F_SetFrom
ogr_api.h, 413
OGR_F_SetFromWithMap
ogr_api.h, 414
OGR_F_SetGeometry
ogr_api.h, 414
OGR_F_SetGeometryDirectly
ogr_api.h, 415
OGR_F_SetStyleString
ogr_api.h, 415
OGR_F_SetStyleStringDirectly
ogr_api.h, 415
OGR_F_StealGeometry
ogr_api.h, 416
OGR_F_UnsetField
ogr_api.h, 416
OGR_FD_AddFieldDefn
ogr_api.h, 416
OGR_FD_Create
ogr_api.h, 416
OGR_FD_DeleteFieldDefn
ogr_api.h, 417
OGR_FD_Dereference
ogr_api.h, 417
OGR_FD_Destroy
ogr_api.h, 418
OGR_FD_GetFieldCount
ogr_api.h, 418
OGR_FD_GetFieldDefn
ogr_api.h, 418
OGR_FD_GetFieldIndex
ogr_api.h, 418
OGR_FD_GetGeomType
ogr_api.h, 419
OGR_FD_GetName
ogr_api.h, 419
OGR_FD_GetReferenceCount
ogr_api.h, 419
OGR_FD_IsGeometryIgnored
ogr_api.h, 420
OGR_FD_IsStyleIgnored
ogr_api.h, 420
OGR_FD_Reference
ogr_api.h, 420
OGR_FD_Release
ogr_api.h, 421
OGR_FD_SetGeomType
ogr_api.h, 421
OGR_FD_SetGeometryIgnored
ogr_api.h, 421
OGR_FD_SetStyleIgnored
ogr_api.h, 421
OGR_Fld_Create
ogr_api.h, 422
OGR_Fld_Destroy
ogr_api.h, 422
OGR_Fld_GetJustify
ogr_api.h, 422
OGR_Fld_GetNameRef
ogr_api.h, 423
OGR_Fld_GetPrecision
ogr_api.h, 423
OGR_Fld_GetType
ogr_api.h, 423

OGR_Fld_GetWidth
ogr_api.h, 423
OGR_Fld_IsIgnored
ogr_api.h, 424
OGR_Fld_Set
ogr_api.h, 424
OGR_Fld_SetIgnored
ogr_api.h, 424
OGR_Fld_SetJustify
ogr_api.h, 425
OGR_Fld_SetName
ogr_api.h, 425
OGR_Fld_SetPrecision
ogr_api.h, 425
OGR_Fld_SetType
ogr_api.h, 425
OGR_Fld_SetWidth
ogr_api.h, 426
OGR_G_AddGeometry
ogr_api.h, 426
OGR_G_AddGeometryDirectly
ogr_api.h, 426
OGR_G_AddPoint
ogr_api.h, 427
OGR_G_AddPoint_2D
ogr_api.h, 427
OGR_G_ApproximateArcAngles
ogr_api.h, 427
OGR_G_Area
ogr_api.h, 428
OGR_G_AssignSpatialReference
ogr_api.h, 428
OGR_G_Boundary
ogr_api.h, 429
OGR_G_Buffer
ogr_api.h, 429
OGR_G_Centroid
ogr_api.h, 430
OGR_G_Clone
ogr_api.h, 430
OGR_G_CloseRings
ogr_api.h, 430
OGR_G_Contains
ogr_api.h, 431
OGR_G_ConvexHull
ogr_api.h, 431
OGR_G_CreateFromGML
ogr_api.h, 431
OGR_G_CreateFromWkb
ogr_api.h, 432
OGR_G_CreateFromWkt
ogr_api.h, 432
OGR_G_CreateGeometry
ogr_api.h, 433
OGR_G_Crosses
ogr_api.h, 433
OGR_G_DestroyGeometry
ogr_api.h, 434
OGR_G_Difference
ogr_api.h, 434
OGR_G_Disjoint
ogr_api.h, 434
OGR_G_Distance
ogr_api.h, 435
OGR_G_DumpReadable
ogr_api.h, 435
OGR_G_Empty
ogr_api.h, 435
OGR_G_Equals
ogr_api.h, 436
OGR_G_ExportToGML
ogr_api.h, 436
OGR_G_ExportToGMLEx
ogr_api.h, 436
OGR_G_ExportToJson
ogr_api.h, 437
OGR_G_ExportToJsonEx
ogr_api.h, 437
OGR_G_ExportToKML
ogr_api.h, 438
OGR_G_ExportToWkb
ogr_api.h, 438
OGR_G_ExportToWkt
ogr_api.h, 438
OGR_G_FlattenTo2D
ogr_api.h, 439
OGR_G_ForceToMultiLineString
ogr_api.h, 439
OGR_G_ForceToMultiPoint
ogr_api.h, 439
OGR_G_ForceToMultiPolygon
ogr_api.h, 440
OGR_G_ForceToPolygon
ogr_api.h, 440
OGR_G_GetArea
ogr_api.h, 440
OGR_G_GetBoundary
ogr_api.h, 441
OGR_G_GetCoordinateDimension
ogr_api.h, 441
OGR_G_GetDimension
ogr_api.h, 441
OGR_G_GetEnvelope
ogr_api.h, 442
OGR_G_GetEnvelope3D
ogr_api.h, 442
OGR_G_GetGeometryCount
ogr_api.h, 442
OGR_G_GetGeometryName
ogr_api.h, 442
OGR_G_GetGeometryRef
ogr_api.h, 443
OGR_G_GetGeometryType
ogr_api.h, 443
OGR_G_GetPoint
ogr_api.h, 444

OGR_G_GetPointCount
ogr_api.h, 444
OGR_G_GetPoints
ogr_api.h, 444
OGR_G_GetSpatialReference
ogr_api.h, 445
OGR_G_GetX
ogr_api.h, 445
OGR_G_GetY
ogr_api.h, 445
OGR_G_GetZ
ogr_api.h, 446
OGR_G_ImportFromWkb
ogr_api.h, 446
OGR_G_ImportFromWkt
ogr_api.h, 446
OGR_G_Intersection
ogr_api.h, 447
OGR_G_Intersects
ogr_api.h, 447
OGR_G_IsEmpty
ogr_api.h, 447
OGR_G_IsRing
ogr_api.h, 448
OGR_G_IsSimple
ogr_api.h, 448
OGR_G_IsValid
ogr_api.h, 448
OGR_G_Length
ogr_api.h, 449
OGR_G_Overlaps
ogr_api.h, 449
OGR_G_Polygonize
ogr_api.h, 450
OGR_G_RemoveGeometry
ogr_api.h, 450
OGR_G_Segmentize
ogr_api.h, 450
OGR_G_SetCoordinateDimension
ogr_api.h, 451
OGR_G_SetPoint
ogr_api.h, 451
OGR_G_SetPoint_2D
ogr_api.h, 451
OGR_G_Simplify
ogr_api.h, 452
OGR_G_SimplifyPreserveTopology
ogr_api.h, 452
OGR_G_SymDifference
ogr_api.h, 452
OGR_G_SymmetricDifference
ogr_api.h, 453
OGR_G_Touches
ogr_api.h, 453
OGR_G_Transform
ogr_api.h, 454
OGR_G_TransformTo
ogr_api.h, 454
OGR_G_Union
ogr_api.h, 455
OGR_G_UnionCascaded
ogr_api.h, 455
OGR_G_Within
ogr_api.h, 455
OGR_G_WkbSize
ogr_api.h, 456
OGR_GetFieldTypeName
ogr_api.h, 456
OGR_L_AlterFieldDefn
ogr_api.h, 456
OGR_L_CommitTransaction
ogr_api.h, 457
OGR_L_CreateFeature
ogr_api.h, 457
OGR_L_CreateField
ogr_api.h, 458
OGR_L_DeleteFeature
ogr_api.h, 458
OGR_L_DeleteField
ogr_api.h, 459
OGR_L_GetExtent
ogr_api.h, 459
OGR_L_GetFIDColumn
ogr_api.h, 461
OGR_L_GetFeature
ogr_api.h, 460
OGR_L_GetFeatureCount
ogr_api.h, 460
OGR_L_GetGeomType
ogr_api.h, 461
OGR_L_GetGeometryColumn
ogr_api.h, 461
OGR_L_GetLayerDefn
ogr_api.h, 462
OGR_L_GetName
ogr_api.h, 462
OGR_L_GetNextFeature
ogr_api.h, 462
OGR_L_GetSpatialFilter
ogr_api.h, 463
OGR_L_GetSpatialRef
ogr_api.h, 463
OGR_L_ReorderField
ogr_api.h, 463
OGR_L_ReorderFields
ogr_api.h, 464
OGR_L_ResetReading
ogr_api.h, 465
OGR_L_RollbackTransaction
ogr_api.h, 465
OGR_L_SetAttributeFilter
ogr_api.h, 465
OGR_L_SetFeature
ogr_api.h, 466
OGR_L_SetIgnoredFields
ogr_api.h, 466

OGR_L_SetNextByIndex
ogr_api.h, 467
OGR_L_SetSpatialFilter
ogr_api.h, 467
OGR_L_SetSpatialFilterRect
ogr_api.h, 468
OGR_L_StartTransaction
ogr_api.h, 468
OGR_L_SyncToDisk
ogr_api.h, 468
OGR_L_TestCapability
ogr_api.h, 469
OGR_SM_AddPart
ogr_api.h, 470
OGR_SM_AddStyle
ogr_api.h, 470
OGR_SM_Create
ogr_api.h, 471
OGR_SM_Destroy
ogr_api.h, 471
OGR_SM_GetPart
ogr_api.h, 471
OGR_SM_GetPartCount
ogr_api.h, 472
OGR_SM_InitFromFeature
ogr_api.h, 472
OGR_SM_InitStyleString
ogr_api.h, 472
OGR_SRSNode, 75
AddChild, 75
applyRemapper, 76
Clone, 76
DestroyChild, 76
exportToWkt, 77
FindChild, 77
GetChild, 77
GetChildCount, 78
GetNode, 78
GetValue, 79
importFromWkt, 79
InsertChild, 79
MakeValueSafe, 80
OGR_SRSNode, 75
OGR_SRSNode, 75
SetValue, 80
StripNodes, 80
OGR_ST_Create
ogr_api.h, 472
OGR_ST_Destroy
ogr_api.h, 473
OGR_ST_GetParamDbl
ogr_api.h, 473
OGR_ST_GetParamNum
ogr_api.h, 473
OGR_ST_GetParamStr
ogr_api.h, 474
OGR_ST_GetRGBFromString
ogr_api.h, 474
OGR_ST_GetStyleString
ogr_api.h, 475
OGR_ST_GetType
ogr_api.h, 475
OGR_ST_GetUnit
ogr_api.h, 475
OGR_ST_SetParamDbl
ogr_api.h, 475
OGR_ST_SetParamNum
ogr_api.h, 476
OGR_ST_SetParamStr
ogr_api.h, 476
OGR_ST_SetUnit
ogr_api.h, 476
OGR_STBL_Create
ogr_api.h, 477
OGR_STBL_Destroy
ogr_api.h, 477
OGR_STBL_Find
ogr_api.h, 477
OGR_STBL_GetLastStyleName
ogr_api.h, 477
OGR_STBL_GetNextStyle
ogr_api.h, 478
OGR_STBL_LoadStyleTable
ogr_api.h, 478
OGR_STBL_ResetStyleStringReading
ogr_api.h, 478
OGR_STBL_SaveStyleTable
ogr_api.h, 479
OGRAttrIndex, 81
OGRBuildPolygonFromEdges
ogr_api.h, 479
OGRCleanupAll
ogr_api.h, 479
OGRCoordinateTransformation, 81
DestroyCT, 82
GetSourceCS, 82
GetTargetCS, 82
Transform, 82
TransformEx, 83
OGRCreateCoordinateTransformation
ogr_spatialref.h, 490
OGRCurve, 83
EndPoint, 84
get_IsClosed, 84
get_Length, 84
StartPoint, 84
Value, 85
OGRDataSource, 85
CopyLayer, 87
CreateLayer, 87
DeleteLayer, 88
Dereference, 88
DestroyDataSource, 88
ExecuteSQL, 88
GetDriver, 89
GetLayer, 89

GetLayerByName, 89
GetLayerCount, 90
GetName, 90
GetRefCount, 90
GetStyleTable, 90
GetSummaryRefCount, 91
Reference, 91
Release, 91
ReleaseResultSet, 91
SetDriver, 92
SetStyleTable, 92
SetStyleTableDirectly, 92
SyncToDisk, 92
TestCapability, 93
OGRDeregisterDriver
ogr_api.h, 480
OGREnvelope, 93
OGREnvelope3D, 94
OGRFeature, 94
Clone, 96
CreateFeature, 97
DestroyFeature, 97
DumpReadable, 97
Equal, 98
GetDefnRef, 98
GetFID, 98
GetFieldAsBinary, 98
GetFieldAsDateTime, 99
GetFieldAsDouble, 99
GetFieldAsDoubleList, 100
GetFieldAsInteger, 100
GetFieldAsIntegerList, 100
GetFieldAsString, 101
GetFieldAsStringList, 101
GetFieldCount, 102
GetFieldDefnRef, 102
GetFieldIndex, 102
GetGeometryRef, 102
GetRawFieldRef, 103
GetStyleString, 103
IsFieldSet, 103
OGRFeature, 96
OGRFeature, 96
SetFID, 103
SetField, 104–106
SetFrom, 107
SetGeometry, 107
SetGeometryDirectly, 108
SetStyleString, 108
SetStyleStringDirectly, 108
StealGeometry, 109
UnsetField, 109
OGRFeatureDefn, 109
AddFieldDefn, 111
Clone, 111
DeleteFieldDefn, 111
Dereference, 112
GetFieldCount, 112
GetFieldDefn, 112
GetFieldIndex, 112
GetGeomType, 113
GetName, 113
GetReferenceCount, 113
IsGeometryIgnored, 113
IsStyleIgnored, 114
OGRFeatureDefn, 110
OGRFeatureDefn, 110
Reference, 114
ReorderFieldDefns, 114
SetGeomType, 115
SetGeometryIgnored, 114
SetStyleIgnored, 115
OGRFeatureQuery, 115
GetUsedFields, 115
OGRField, 116
OGRFieldDefn, 116
GetFieldTypeName, 117
GetJustify, 118
GetNameRef, 118
GetPrecision, 118
GetType, 118
GetWidth, 119
IsIgnored, 119
OGRFieldDefn, 117
OGRFieldDefn, 117
Set, 119
SetDefault, 119
SetIgnored, 119
SetJustify, 120
SetName, 120
SetPrecision, 120
SetType, 120
SetWidth, 121
OGRFieldType
ogr_core.h, 486
OGRGenSQLResultsLayer, 121
GetExtent, 122
GetFeature, 122
GetFeatureCount, 123
GetLayerDefn, 123
GetNextFeature, 123
GetSpatialFilter, 123
GetSpatialRef, 124
ResetReading, 124
SetNextByIndex, 124
TestCapability, 125
OGRGeometry, 126
assignSpatialReference, 129
Boundary, 129
Buffer, 129
Centroid, 130
clone, 130
closeRings, 130
Contains, 130
ConvexHull, 131
Crosses, 131

Difference, 131
Disjoint, 132
Distance, 132
dumpReadable, 132
empty, 133
Equals, 133
exportToGML, 133
exportToJson, 134
exportToKML, 134
exportToWkb, 134
exportToWkt, 135
flattenTo2D, 135
getBoundary, 135
getCoordinateDimension, 135
getDimension, 136
getEnvelope, 136
getGeometryName, 137
getGeometryType, 137
getSpatialReference, 137
importFromWkb, 138
importFromWkt, 138
Intersection, 138
Intersects, 139
IsEmpty, 139
IsRing, 139
IsSimple, 140
IsValid, 140
Overlaps, 140
Polygonize, 140
segmentize, 141
setCoordinateDimension, 141
Simplify, 141
SimplifyPreserveTopology, 142
swapXY, 142
SymDifference, 142
SymmetricDifference, 143
Touches, 143
transform, 143
transformTo, 144
Union, 144
UnionCascaded, 145
Within, 145
WkbSize, 145
OGRGeometryCollection, 146
addGeometry, 147
addGeometryDirectly, 148
clone, 148
closeRings, 148
empty, 148
Equals, 149
exportToWkb, 149
exportToWkt, 149
flattenTo2D, 150
get_Area, 150
get_Length, 150
getDimension, 150
getEnvelope, 151
getGeometryName, 151
getGeometryRef, 152
getGeometryType, 152
getNumGeometries, 152
importFromWkb, 153
importFromWkt, 153
IsEmpty, 153
removeGeometry, 154
segmentize, 154
setCoordinateDimension, 154
swapXY, 155
transform, 155
WkbSize, 155
OGRGeometryFactory, 156
approximateArcAngles, 157
createFromFgf, 157
createFromGML, 158
createFromWkb, 158
createFromWkt, 159
createGeometry, 159
destroyGeometry, 160
forceToMultiLineString, 160
forceToMultiPoint, 160
forceToMultiPolygon, 161
forceToPolygon, 161
haveGEOS, 161
organizePolygons, 161
OGRGeometryTypeToName
ogr_core.h, 487
OGRGetDriver
ogr_api.h, 480
OGRGetDriverByName
ogr_api.h, 480
OGRGetDriverCount
ogr_api.h, 481
OGRGetOpenDS
ogr_api.h, 481
OGRGetOpenDSCount
ogr_api.h, 481
OGRJustification
ogr_core.h, 486
OGRLayer, 162
AlterFieldDefn, 164
CreateFeature, 165
CreateField, 165
DeleteFeature, 165
DeleteField, 166
Dereference, 166
GetExtent, 166
GetFIDColumn, 168
GetFeature, 167
GetFeatureCount, 168
GetGeomType, 168
GetGeometryColumn, 168
GetInfo, 169
GetLayerDefn, 169
GetName, 169
GetNextFeature, 170
GetRefCount, 170

GetSpatialFilter, 170
GetSpatialRef, 170
GetStyleTable, 171
Reference, 171
ReorderField, 171
ReorderFields, 172
ResetReading, 172
SetAttributeFilter, 173
SetFeature, 173
SetIgnoredFields, 173
SetNextByIndex, 174
SetSpatialFilter, 174
SetSpatialFilterRect, 175
SetStyleTable, 175
SetStyleTableDirectly, 175
SyncToDisk, 176
TestCapability, 176
OGRLayerAttrIndex, 177
OGRLineString, 181
addPoint, 183
addSubLineString, 183
clone, 184
empty, 184
EndPoint, 184
Equals, 184
exportToWkb, 185
exportToWkt, 185
flattenTo2D, 185
get_Length, 186
getDimension, 186
getEnvelope, 186
getGeometryName, 187
getGeometryType, 187
getNumPoints, 187
getPoint, 187
getPoints, 188
getX, 188
getY, 189
getZ, 189
importFromWkb, 189
importFromWkt, 190
IsEmpty, 190
segmentize, 190
setCoordinateDimension, 191
setNumPoints, 191
setPoint, 191, 192
setPoints, 192
StartPoint, 192
swapXY, 193
transform, 193
Value, 193
WkbSize, 194
OGRLinearRing, 178
clone, 179
closeRings, 179
exportToWkb, 179
get_Area, 179
getGeometryName, 180
importFromWkb, 180
isClockwise, 180
WkbSize, 180
OGRMIAttrIndex, 194
OGRMILayerAttrIndex, 194
OGRMergeGeometryTypes
ogr_core.h, 487
OGRMultiLineString, 195
addGeometryDirectly, 195
clone, 196
exportToWkt, 196
getGeometryName, 196
getGeometryType, 197
importFromWkt, 197
OGRMultiPoint, 198
addGeometryDirectly, 198
clone, 199
exportToWkt, 199
getGeometryName, 199
getGeometryType, 199
importFromWkt, 200
OGRMultiPolygon, 200
addGeometryDirectly, 201
clone, 201
exportToWkt, 202
get_Area, 202
getGeometryName, 202
getGeometryType, 202
importFromWkt, 203
OGROpen
ogr_api.h, 481
OGRParseDate
ogr_core.h, 488
OGRPoint, 203
clone, 205
empty, 205
Equals, 205
exportToWkb, 205
exportToWkt, 206
flattenTo2D, 206
getDimension, 206
getEnvelope, 206, 207
getGeometryName, 207
getGeometryType, 207
getX, 207
getY, 208
getZ, 208
importFromWkb, 208
importFromWkt, 209
IsEmpty, 209
setCoordinateDimension, 209
setX, 210
setY, 210
setZ, 210
swapXY, 210
transform, 210
WkbSize, 211
OGRPolygon, 211

addRing, 213
addRingDirectly, 213
clone, 213
closeRings, 213
empty, 214
Equals, 214
exportToWkb, 214
exportToWkt, 214
flattenTo2D, 215
get_Area, 215
getDimension, 215
getEnvelope, 215, 216
getExteriorRing, 216
getGeometryName, 216
getGeometryType, 217
getInteriorRing, 217
getNumInteriorRings, 217
importFromWkb, 217
importFromWkt, 218
IsEmpty, 218
PointOnSurface, 219
segmentize, 219
setCoordinateDimension, 219
swapXY, 219
transform, 220
WkbSize, 220
OGRProj4CT, 220
GetSourceCS, 221
GetTargetCS, 221
Transform, 221
TransformEx, 221
OGRProj4Datum, 222
OGRProj4PM, 222
OGRRawPoint, 222
OGRRegisterDriver
ogr_api.h, 482
OGRReleaseDataSource
ogr_api.h, 482
OGRSFDriver, 222
CopyDataSource, 223
CreateDataSource, 223
DeleteDataSource, 224
GetName, 224
Open, 224
TestCapability, 225
OGRSFDriverRegistrar, 226
AutoLoadDrivers, 226
DeregisterDriver, 227
GetDriver, 227
GetDriverByName, 227
GetDriverCount, 228
GetOpenDS, 228
GetOpenDSCount, 228
GetRegistrar, 228
Open, 229
RegisterDriver, 229
OGRSTBrushParam
ogr_core.h, 485
OGRSTClassId
ogr_core.h, 485
OGRSTLabelParam
ogr_core.h, 485
OGRSTPenParam
ogr_core.h, 485
OGRSTSymbolParam
ogr_core.h, 485
OGRSTUnitId
ogr_core.h, 485
OGRSetGenerate_DB2_V72_BYTE_ORDER
ogr_api.h, 483
OGRSpatialReference, 230
∼OGRSpatialReference, 236
AutoIdentifyEPSG, 236
Clear, 236
Clone, 236
CloneGeogCS, 237
CopyGeogCSFrom, 237
Dereference, 237
DestroySpatialReference, 237
EPSGTreatsAsLatLong, 238
exportToERM, 238
exportToMICoordSys, 238
exportToPCI, 239
exportToPanorama, 239
exportToPrettyWkt, 240
exportToProj4, 240
exportToUSGS, 240
exportToWkt, 241
exportToXML, 241
FindProjParm, 241
Fixup, 242
FixupOrdering, 242
GetAngularUnits, 242
GetAttrNode, 243
GetAttrValue, 243
GetAuthorityCode, 244
GetAuthorityName, 244
GetAxis, 245
GetExtension, 245
GetInvFlattening, 245
GetLinearUnits, 246
GetNormProjParm, 246
GetPrimeMeridian, 246
GetProjParm, 247
GetReferenceCount, 247
GetSemiMajor, 247
GetSemiMinor, 248
GetTOWGS84, 248
GetTargetLinearUnits, 248
GetUTMZone, 249
importFromDict, 249
importFromEPSG, 250
importFromEPSGA, 250
importFromERM, 251
importFromESRI, 251
importFromMICoordSys, 251

importFromOzi, 252
importFromPCI, 254
importFromPanorama, 252
importFromProj4, 254
importFromURN, 256
importFromUSGS, 256
importFromUrl, 255
importFromWMSAUTO, 260
importFromWkt, 259
importFromXML, 260
IsCompound, 260
IsGeocentric, 261
IsGeographic, 261
IsLocal, 261
IsProjected, 261
IsSame, 262
IsSameGeogCS, 262
IsSameVertCS, 262
IsVertical, 263
morphFromESRI, 263
morphToESRI, 264
OGRSpatialReference, 235
OGRSpatialReference, 235
Reference, 264
Release, 265
SetACEA, 265
SetAE, 265
SetAngularUnits, 265
SetAuthority, 265
SetAxes, 266
SetBonne, 266
SetCEA, 266
SetCS, 267
SetCompoundCS, 267
SetEC, 267
SetEckert, 267
SetEquirectangular, 267
SetEquirectangular2, 268
SetExtension, 268
SetFromUserInput, 268
SetGEOS, 270
SetGH, 270
SetGS, 271
SetGaussSchreiberTMercator, 269
SetGeocCS, 269
SetGeogCS, 270
SetGnomonic, 271
SetHOM, 271
SetHOM2PNO, 271
SetIGH, 272
SetIWMPolyconic, 272
SetKrovak, 272
SetLAEA, 272
SetLCC, 272
SetLCC1SP, 273
SetLCCB, 273
SetLinearUnits, 273
SetLinearUnitsAndUpdateParameters, 273
SetLocalCS, 274
SetMC, 274
SetMercator, 274
SetMollweide, 274
SetNZMG, 275
SetNode, 274
SetNormProjParm, 275
SetOS, 276
SetOrthographic, 275
SetPS, 277
SetPolyconic, 276
SetProjCS, 276
SetProjParm, 277
SetProjection, 276
SetRobinson, 277
SetRoot, 277
SetSOC, 278
SetSinusoidal, 278
SetStatePlane, 278
SetStereographic, 278
SetTM, 279
SetTMG, 279
SetTMSO, 279
SetTMVariant, 280
SetTOWGS84, 280
SetTPED, 280
SetTargetLinearUnits, 279
SetUTM, 280
SetVDG, 281
SetVertCS, 281
SetWagner, 281
SetWellKnownGeogCS, 281
StripCTParms, 282
StripVertical, 282
Validate, 283
OGRStyleBrush, 283
OGRStyleLabel, 284
OGRStyleMgr, 284
~OGRStyleMgr, 285
AddPart, 285
AddStyle, 286
GetPart, 286
GetPartCount, 286
GetStyleByName, 287
GetStyleName, 287
GetStyleString, 287
InitFromFeature, 287
InitStyleString, 288
OGRStyleMgr, 285
OGRStyleMgr, 285
SetFeatureStyleString, 288
OGRStylePen, 288
OGRStyleSymbol, 289
OGRStyleTable, 289
AddStyle, 290
Clone, 290
Find, 290
GetStyleName, 291

IsExist, 291
LoadStyleTable, 291
ModifyStyle, 291
Print, 292
RemoveStyle, 292
SaveStyleTable, 292
OGRStyleTool, 292
OGRSurface, 293
get_Area, 293
PointOnSurface, 294
OGRwkbGeometryType
ogr_core.h, 486
OPTGetParameterInfo
ogr_srs_api.h, 496
OPTGetParameterList
ogr_srs_api.h, 497
OPTGetProjectionMethods
ogr_srs_api.h, 497
OSRAutoIdentifyEPSG
ogr_srs_api.h, 497
OSRAxisEnumToName
ogr_srs_api.h, 497
OSRCleanup
ogr_srs_api.h, 498
OSRClone
ogr_srs_api.h, 498
OSRCloneGeogCS
ogr_srs_api.h, 498
OSRCopyGeogCSFrom
ogr_srs_api.h, 498
OSRDereference
ogr_srs_api.h, 498
OSRDestroySpatialReference
ogr_srs_api.h, 498
OSREPSGTreatsAsLatLong
ogr_srs_api.h, 499
OSRExportToERM
ogr_srs_api.h, 499
OSRExportToMICoordSys
ogr_srs_api.h, 499
OSRExportToPCI
ogr_srs_api.h, 499
OSRExportToPrettyWkt
ogr_srs_api.h, 499
OSRExportToProj4
ogr_srs_api.h, 499
OSRExportToUSGS
ogr_srs_api.h, 500
OSRExportToWkt
ogr_srs_api.h, 500
OSRExportToXML
ogr_srs_api.h, 500
OSRFixup
ogr_srs_api.h, 500
OSRFixupOrdering
ogr_srs_api.h, 500
OSRGetAngularUnits
ogr_srs_api.h, 500
OSRGetAttrValue
ogr_srs_api.h, 501
OSRGetAuthorityCode
ogr_srs_api.h, 501
OSRGetAuthorityName
ogr_srs_api.h, 501
OSRGetAxis
ogr_srs_api.h, 501
OSRGetInvFlattening
ogr_srs_api.h, 501
OSRGetLinearUnits
ogr_srs_api.h, 501
OSRGetNormProjParm
ogr_srs_api.h, 502
OSRGetPrimeMeridian
ogr_srs_api.h, 502
OSRGetProjParm
ogr_srs_api.h, 502
OSRGetSemiMajor
ogr_srs_api.h, 502
OSRGetSemiMinor
ogr_srs_api.h, 502
OSRGetTOWGS84
ogr_srs_api.h, 503
OSRGetTargetLinearUnits
ogr_srs_api.h, 502
OSRGetUTMZone
ogr_srs_api.h, 503
OSRImportFromEPSG
ogr_srs_api.h, 503
OSRImportFromEPSGA
ogr_srs_api.h, 503
OSRImportFromERM
ogr_srs_api.h, 503
OSRImportFromESRI
ogr_srs_api.h, 503
OSRImportFromMICoordSys
ogr_srs_api.h, 504
OSRImportFromPCI
ogr_srs_api.h, 504
OSRImportFromProj4
ogr_srs_api.h, 504
OSRImportFromUSGS
ogr_srs_api.h, 504
OSRImportFromUrl
ogr_srs_api.h, 504
OSRImportFromWkt
ogr_srs_api.h, 504
OSRImportFromXML
ogr_srs_api.h, 505
OSRIsCompound
ogr_srs_api.h, 505
OSRIsGeocentric
ogr_srs_api.h, 505
OSRIsGeographic
ogr_srs_api.h, 505
OSRIsLocal
ogr_srs_api.h, 505

OSRIsProjected
ogr_srs_api.h, 505
OSRIsSame
ogr_srs_api.h, 506
OSRIsSameGeogCS
ogr_srs_api.h, 506
OSRIsSameVertCS
ogr_srs_api.h, 506
OSRIsVertical
ogr_srs_api.h, 506
OSRMorphFromESRI
ogr_srs_api.h, 506
OSRMorphToESRI
ogr_srs_api.h, 506
OSRNewSpatialReference
ogr_srs_api.h, 507
OSRReference
ogr_srs_api.h, 507
OSRRelease
ogr_srs_api.h, 507
OSRSetACEA
ogr_srs_api.h, 507
OSRSetAE
ogr_srs_api.h, 507
OSRSetAngularUnits
ogr_srs_api.h, 507
OSRSetAttrValue
ogr_srs_api.h, 508
OSRSetAuthority
ogr_srs_api.h, 508
OSRSetBonne
ogr_srs_api.h, 508
OSRSetCEA
ogr_srs_api.h, 508
OSRSetCS
ogr_srs_api.h, 508
OSRSetCompoundCS
ogr_srs_api.h, 508
OSRSetEC
ogr_srs_api.h, 509
OSRSetEckert
ogr_srs_api.h, 509
OSRSetEckertIV
ogr_srs_api.h, 509
OSRSetEckertVI
ogr_srs_api.h, 509
OSRSetEquirectangular
ogr_srs_api.h, 509
OSRSetEquirectangular2
ogr_srs_api.h, 509
OSRSetFromUserInput
ogr_srs_api.h, 509
OSRSetGEOS
ogr_srs_api.h, 510
OSRSetGH
ogr_srs_api.h, 510
OSRSetGS
ogr_srs_api.h, 511
OSRSetGaussSchreiberTMercator
ogr_srs_api.h, 510
OSRSetGeocCS
ogr_srs_api.h, 510
OSRSetGeogCS
ogr_srs_api.h, 510
OSRSetGnomonic
ogr_srs_api.h, 510
OSRSetHOM
ogr_srs_api.h, 511
OSRSetHOM2PNO
ogr_srs_api.h, 511
OSRSetIGH
ogr_srs_api.h, 511
OSRSetIWMPolyconic
ogr_srs_api.h, 511
OSRSetKrovak
ogr_srs_api.h, 511
OSRSetLAEA
ogr_srs_api.h, 512
OSRSetLCC
ogr_srs_api.h, 512
OSRSetLCC1SP
ogr_srs_api.h, 512
OSRSetLCCB
ogr_srs_api.h, 512
OSRSetLinearUnits
ogr_srs_api.h, 512
OSRSetLinearUnitsAndUpdateParameters
ogr_srs_api.h, 512
OSRSetLocalCS
ogr_srs_api.h, 513
OSRSetMC
ogr_srs_api.h, 513
OSRSetMercator
ogr_srs_api.h, 513
OSRSetMollweide
ogr_srs_api.h, 513
OSRSetNZMG
ogr_srs_api.h, 513
OSRSetNormProjParm
ogr_srs_api.h, 513
OSRSetOS
ogr_srs_api.h, 514
OSRSetOrthographic
ogr_srs_api.h, 513
OSRSetPS
ogr_srs_api.h, 514
OSRSetPolyconic
ogr_srs_api.h, 514
OSRSetProjCS
ogr_srs_api.h, 514
OSRSetProjParm
ogr_srs_api.h, 514
OSRSetProjection
ogr_srs_api.h, 514
OSRSetRobinson
ogr_srs_api.h, 515

OSRSetSOC
ogr_srs_api.h, 515
OSRSetSinusoidal
ogr_srs_api.h, 515
OSRSetStatePlane
ogr_srs_api.h, 515
OSRSetStatePlaneWithUnits
ogr_srs_api.h, 515
OSRSetStereographic
ogr_srs_api.h, 515
OSRSetTM
ogr_srs_api.h, 516
OSRSetTMG
ogr_srs_api.h, 516
OSRSetTMSO
ogr_srs_api.h, 516
OSRSetTMVariant
ogr_srs_api.h, 516
OSRSetTOWGS84
ogr_srs_api.h, 516
OSRSetTargetLinearUnits
ogr_srs_api.h, 515
OSRSetUTM
ogr_srs_api.h, 516
OSRSetVDG
ogr_srs_api.h, 517
OSRSetVertCS
ogr_srs_api.h, 517
OSRSetWagner
ogr_srs_api.h, 517
OSRSetWellKnownGeogCS
ogr_srs_api.h, 517
OSRStripCTParms
ogr_srs_api.h, 517
OSRValidate
ogr_srs_api.h, 517
OZIDatums, 294
ogr_api.h, 384
OGR_DS_CopyLayer, 397
OGR_DS_CreateLayer, 398
OGR_DS_DeleteLayer, 398
OGR_DS_Destroy, 399
OGR_DS_ExecuteSQL, 399
OGR_DS_GetDriver, 400
OGR_DS_GetLayer, 400
OGR_DS_GetLayerByName, 400
OGR_DS_GetLayerCount, 401
OGR_DS_GetName, 401
OGR_DS_ReleaseResultSet, 401
OGR_DS_SyncToDisk, 402
OGR_DS_TestCapability, 402
OGR_Dr_CopyDataSource, 395
OGR_Dr_CreateDataSource, 395
OGR_Dr_DeleteDataSource, 396
OGR_Dr_GetName, 396
OGR_Dr_Open, 396
OGR_Dr_TestCapability, 397
OGR_F_Clone, 402
OGR_F_Create, 403
OGR_F_Destroy, 403
OGR_F_DumpReadable, 403
OGR_F_Equal, 404
OGR_F_GetDefnRef, 404
OGR_F_GetFID, 404
OGR_F_GetFieldAsBinary, 405
OGR_F_GetFieldAsDateTime, 405
OGR_F_GetFieldAsDouble, 406
OGR_F_GetFieldAsDoubleList, 406
OGR_F_GetFieldAsInteger, 406
OGR_F_GetFieldAsIntegerList, 407
OGR_F_GetFieldAsString, 407
OGR_F_GetFieldAsStringList, 407
OGR_F_GetFieldCount, 408
OGR_F_GetFieldDefnRef, 408
OGR_F_GetFieldIndex, 408
OGR_F_GetGeometryRef, 409
OGR_F_GetRawFieldRef, 409
OGR_F_GetStyleString, 409
OGR_F_IsFieldSet, 410
OGR_F_SetFID, 410
OGR_F_SetFieldBinary, 411
OGR_F_SetFieldDateTime, 411
OGR_F_SetFieldDouble, 411
OGR_F_SetFieldDoubleList, 412
OGR_F_SetFieldInteger, 412
OGR_F_SetFieldIntegerList, 412
OGR_F_SetFieldRaw, 412
OGR_F_SetFieldString, 413
OGR_F_SetFieldStringList, 413
OGR_F_SetFrom, 413
OGR_F_SetFromWithMap, 414
OGR_F_SetGeometry, 414
OGR_F_SetGeometryDirectly, 415
OGR_F_SetStyleString, 415
OGR_F_SetStyleStringDirectly, 415
OGR_F_StealGeometry, 416
OGR_F_UnsetField, 416
OGR_FD_AddFieldDefn, 416
OGR_FD_Create, 416
OGR_FD_DeleteFieldDefn, 417
OGR_FD_Dereference, 417
OGR_FD_Destroy, 418
OGR_FD_GetFieldCount, 418
OGR_FD_GetFieldDefn, 418
OGR_FD_GetFieldIndex, 418
OGR_FD_GetGeomType, 419
OGR_FD_GetName, 419
OGR_FD_GetReferenceCount, 419
OGR_FD_IsGeometryIgnored, 420
OGR_FD_IsStyleIgnored, 420
OGR_FD_Reference, 420
OGR_FD_Release, 421
OGR_FD_SetGeomType, 421
OGR_FD_SetGeometryIgnored, 421
OGR_FD_SetStyleIgnored, 421
OGR_Fld_Create, 422

OGR_Fld_Destroy, 422
OGR_Fld_GetJustify, 422
OGR_Fld_GetNameRef, 423
OGR_Fld_GetPrecision, 423
OGR_Fld_GetType, 423
OGR_Fld_GetWidth, 423
OGR_Fld_IsIgnored, 424
OGR_Fld_Set, 424
OGR_Fld_SetIgnored, 424
OGR_Fld_SetJustify, 425
OGR_Fld_SetName, 425
OGR_Fld_SetPrecision, 425
OGR_Fld_SetType, 425
OGR_Fld_SetWidth, 426
OGR_G_AddGeometry, 426
OGR_G_AddGeometryDirectly, 426
OGR_G_AddPoint, 427
OGR_G_AddPoint_2D, 427
OGR_G_ApproximateArcAngles, 427
OGR_G_Area, 428
OGR_G_AssignSpatialReference, 428
OGR_G_Boundary, 429
OGR_G_Buffer, 429
OGR_G_Centroid, 430
OGR_G_Clone, 430
OGR_G_CloseRings, 430
OGR_G_Contains, 431
OGR_G_ConvexHull, 431
OGR_G_CreateFromGML, 431
OGR_G_CreateFromWkb, 432
OGR_G_CreateFromWkt, 432
OGR_G_CreateGeometry, 433
OGR_G_Crosses, 433
OGR_G_DestroyGeometry, 434
OGR_G_Difference, 434
OGR_G_Disjoint, 434
OGR_G_Distance, 435
OGR_G_DumpReadable, 435
OGR_G_Empty, 435
OGR_G_Equals, 436
OGR_G_ExportToGML, 436
OGR_G_ExportToGMLEx, 436
OGR_G_ExportToJson, 437
OGR_G_ExportToJsonEx, 437
OGR_G_ExportToKML, 438
OGR_G_ExportToWkb, 438
OGR_G_ExportToWkt, 438
OGR_G_FlattenTo2D, 439
OGR_G_ForceToMultiLineString, 439
OGR_G_ForceToMultiPoint, 439
OGR_G_ForceToMultiPolygon, 440
OGR_G_ForceToPolygon, 440
OGR_G_GetArea, 440
OGR_G_GetBoundary, 441
OGR_G_GetCoordinateDimension, 441
OGR_G_GetDimension, 441
OGR_G_GetEnvelope, 442
OGR_G_GetEnvelope3D, 442
OGR_G_GetGeometryCount, 442
OGR_G_GetGeometryName, 442
OGR_G_GetGeometryRef, 443
OGR_G_GetGeometryType, 443
OGR_G_GetPoint, 444
OGR_G_GetPointCount, 444
OGR_G_GetPoints, 444
OGR_G_GetSpatialReference, 445
OGR_G_GetX, 445
OGR_G_GetY, 445
OGR_G_GetZ, 446
OGR_G_ImportFromWkb, 446
OGR_G_ImportFromWkt, 446
OGR_G_Intersection, 447
OGR_G_Intersects, 447
OGR_G_IsEmpty, 447
OGR_G_IsRing, 448
OGR_G_IsSimple, 448
OGR_G_IsValid, 448
OGR_G_Length, 449
OGR_G_Overlaps, 449
OGR_G_Polygonize, 450
OGR_G_RemoveGeometry, 450
OGR_G_Segmentize, 450
OGR_G_SetCoordinateDimension, 451
OGR_G_SetPoint, 451
OGR_G_SetPoint_2D, 451
OGR_G_Simplify, 452
OGR_G_SimplifyPreserveTopology, 452
OGR_G_SymDifference, 452
OGR_G_SymmetricDifference, 453
OGR_G_Touches, 453
OGR_G_Transform, 454
OGR_G_TransformTo, 454
OGR_G_Union, 455
OGR_G_UnionCascaded, 455
OGR_G_Within, 455
OGR_G_WkbSize, 456
OGR_GetFieldTypeName, 456
OGR_L_AlterFieldDefn, 456
OGR_L_CommitTransaction, 457
OGR_L_CreateFeature, 457
OGR_L_CreateField, 458
OGR_L_DeleteFeature, 458
OGR_L_DeleteField, 459
OGR_L_GetExtent, 459
OGR_L_GetFIDColumn, 461
OGR_L_GetFeature, 460
OGR_L_GetFeatureCount, 460
OGR_L_GetGeomType, 461
OGR_L_GetGeometryColumn, 461
OGR_L_GetLayerDefn, 462
OGR_L_GetName, 462
OGR_L_GetNextFeature, 462
OGR_L_GetSpatialFilter, 463
OGR_L_GetSpatialRef, 463
OGR_L_ReorderField, 463
OGR_L_ReorderFields, 464

OGR_L_ResetReading, 465
OGR_L_RollbackTransaction, 465
OGR_L_SetAttributeFilter, 465
OGR_L_SetFeature, 466
OGR_L_SetIgnoredFields, 466
OGR_L_SetNextByIndex, 467
OGR_L_SetSpatialFilter, 467
OGR_L_SetSpatialFilterRect, 468
OGR_L_StartTransaction, 468
OGR_L_SyncToDisk, 468
OGR_L_TestCapability, 469
OGR_SM_AddPart, 470
OGR_SM_AddStyle, 470
OGR_SM_Create, 471
OGR_SM_Destroy, 471
OGR_SM_GetPart, 471
OGR_SM_GetPartCount, 472
OGR_SM_InitFromFeature, 472
OGR_SM_InitStyleString, 472
OGR_ST_Create, 472
OGR_ST_Destroy, 473
OGR_ST_GetParamDbl, 473
OGR_ST_GetParamNum, 473
OGR_ST_GetParamStr, 474
OGR_ST_GetRGBFromString, 474
OGR_ST_GetStyleString, 475
OGR_ST_GetType, 475
OGR_ST_GetUnit, 475
OGR_ST_SetParamDbl, 475
OGR_ST_SetParamNum, 476
OGR_ST_SetParamStr, 476
OGR_ST_SetUnit, 476
OGR_STBL_Create, 477
OGR_STBL_Destroy, 477
OGR_STBL_Find, 477
OGR_STBL_GetLastStyleName, 477
OGR_STBL_GetNextStyle, 478
OGR_STBL_LoadStyleTable, 478
OGR_STBL_ResetStyleStringReading, 478
OGR_STBL_SaveStyleTable, 479
OGRBuildPolygonFromEdges, 479
OGRCleanupAll, 479
OGRDeregisterDriver, 480
OGRGetDriver, 480
OGRGetDriverByName, 480
OGRGetDriverCount, 481
OGRGetOpenDS, 481
OGRGetOpenDSCount, 481
OGROpen, 481
OGRRegisterDriver, 482
OGRReleaseDataSource, 482
OGRSetGenerate_DB2_V72_BYTE_ORDER, 483
ogr_core.h, 483
GDAL_CHECK_VERSION, 484
GDALCheckVersion, 487
OFTBinary, 486
OFTDate, 486
OFTDateTime, 486
OFTInteger, 486
OFTIntegerList, 486
OFTReal, 486
OFTRealList, 486
OFTString, 486
OFTStringList, 486
OFTTime, 486
OFTWideString, 486
OFTWideStringList, 486
OGRFieldType, 486
OGRGeometryTypeToName, 487
OGRJustification, 486
OGRMergeGeometryTypes, 487
OGRParseDate, 488
OGRSTBrushParam, 485
OGRSTClassId, 485
OGRSTLabelParam, 485
OGRSTPenParam, 485
OGRSTSymbolParam, 485
OGRSTUnitId, 485
OGRwkbGeometryType, 486
ogr_style_tool_class_id, 485
ogr_style_tool_param_brush_id, 485
ogr_style_tool_param_label_id, 485
ogr_style_tool_param_pen_id, 485
ogr_style_tool_param_symbol_id, 485
ogr_style_tool_units_id, 485
wkbGeometryCollection, 486
wkbGeometryCollection25D, 486
wkbLineString, 486
wkbLineString25D, 486
wkbLinearRing, 486
wkbMultiLineString, 486
wkbMultiLineString25D, 486
wkbMultiPoint, 486
wkbMultiPoint25D, 486
wkbMultiPolygon, 486
wkbMultiPolygon25D, 486
wkbNone, 486
wkbPoint, 486
wkbPoint25D, 486
wkbPolygon, 486
wkbPolygon25D, 486
wkbUnknown, 486
ogr_feature.h, 488
ogr_featurestyle.h, 489
ogr_geometry.h, 489
ogr_spatialref.h, 490
OGRCreateCoordinateTransformation, 490
ogr_srs_api.h, 490
OCTDestroyCoordinateTransformation, 496
OCTNewCoordinateTransformation, 496
OPTGetParameterInfo, 496
OPTGetParameterList, 497
OPTGetProjectionMethods, 497
OSRAutoIdentifyEPSG, 497
OSRAxisEnumToName, 497
OSRCleanup, 498

OSRClone, 498
OSRCloneGeogCS, 498
OSRCopyGeogCSFrom, 498
OSRDereference, 498
OSRDestroySpatialReference, 498
OSREPSGTreatsAsLatLong, 499
OSRExportToERM, 499
OSRExportToMICoordSys, 499
OSRExportToPCI, 499
OSRExportToPrettyWkt, 499
OSRExportToProj4, 499
OSRExportToUSGS, 500
OSRExportToWkt, 500
OSRExportToXML, 500
OSRFixup, 500
OSRFixupOrdering, 500
OSRGetAngularUnits, 500
OSRGetAttrValue, 501
OSRGetAuthorityCode, 501
OSRGetAuthorityName, 501
OSRGetAxis, 501
OSRGetInvFlattening, 501
OSRGetLinearUnits, 501
OSRGetNormProjParm, 502
OSRGetPrimeMeridian, 502
OSRGetProjParm, 502
OSRGetSemiMajor, 502
OSRGetSemiMinor, 502
OSRGetTOWGS84, 503
OSRGetTargetLinearUnits, 502
OSRGetUTMZone, 503
OSRImportFromEPSG, 503
OSRImportFromEPSGA, 503
OSRImportFromERM, 503
OSRImportFromESRI, 503
OSRImportFromMICoordSys, 504
OSRImportFromPCI, 504
OSRImportFromProj4, 504
OSRImportFromUSGS, 504
OSRImportFromUrl, 504
OSRImportFromWkt, 504
OSRImportFromXML, 505
OSRIsCompound, 505
OSRIsGeocentric, 505
OSRIsGeographic, 505
OSRIsLocal, 505
OSRIsProjected, 505
OSRIsSame, 506
OSRIsSameGeogCS, 506
OSRIsSameVertCS, 506
OSRIsVertical, 506
OSRMorphFromESRI, 506
OSRMorphToESRI, 506
OSRNewSpatialReference, 507
OSRReference, 507
OSRRelease, 507
OSRSetACEA, 507
OSRSetAE, 507
OSRSetAngularUnits, 507
OSRSetAttrValue, 508
OSRSetAuthority, 508
OSRSetBonne, 508
OSRSetCEA, 508
OSRSetCS, 508
OSRSetCompoundCS, 508
OSRSetEC, 509
OSRSetEckert, 509
OSRSetEckertIV, 509
OSRSetEckertVI, 509
OSRSetEquirectangular, 509
OSRSetEquirectangular2, 509
OSRSetFromUserInput, 509
OSRSetGEOS, 510
OSRSetGH, 510
OSRSetGS, 511
OSRSetGaussSchreiberTMercator, 510
OSRSetGeocCS, 510
OSRSetGeogCS, 510
OSRSetGnomonic, 510
OSRSetHOM, 511
OSRSetHOM2PNO, 511
OSRSetIGH, 511
OSRSetIWMPolyconic, 511
OSRSetKrovak, 511
OSRSetLAEA, 512
OSRSetLCC, 512
OSRSetLCC1SP, 512
OSRSetLCCB, 512
OSRSetLinearUnits, 512
OSRSetLinearUnitsAndUpdateParameters, 512
OSRSetLocalCS, 513
OSRSetMC, 513
OSRSetMercator, 513
OSRSetMollweide, 513
OSRSetNZMG, 513
OSRSetNormProjParm, 513
OSRSetOS, 514
OSRSetOrthographic, 513
OSRSetPS, 514
OSRSetPolyconic, 514
OSRSetProjCS, 514
OSRSetProjParm, 514
OSRSetProjection, 514
OSRSetRobinson, 515
OSRSetSOC, 515
OSRSetSinusoidal, 515
OSRSetStatePlane, 515
OSRSetStatePlaneWithUnits, 515
OSRSetStereographic, 515
OSRSetTM, 516
OSRSetTMG, 516
OSRSetTMSO, 516
OSRSetTMVariant, 516
OSRSetTOWGS84, 516
OSRSetTargetLinearUnits, 515
OSRSetUTM, 516

OSRSetVDG, 517
OSRSetVertCS, 517
OSRSetWagner, 517
OSRSetWellKnownGeogCS, 517
OSRStripCTParms, 517
OSRValidate, 517
ogr_style_param, 80
ogr_style_tool_class_id
ogr_core.h, 485
ogr_style_tool_param_brush_id
ogr_core.h, 485
ogr_style_tool_param_label_id
ogr_core.h, 485
ogr_style_tool_param_pen_id
ogr_core.h, 485
ogr_style_tool_param_symbol_id
ogr_core.h, 485
ogr_style_tool_units_id
ogr_core.h, 485
ogr_style_value, 81
ogrsf_frmts.h, 518
Open
OGRSFDriver, 224
OGRSFDriverRegistrar, 229
operator[]
CPLStringList, 71
organizePolygons
OGRGeometryFactory, 161
Overlaps
OGRGeometry, 140

PCIDatums, 294
pData
_CPLList, 51
pabyData
CPLHTTPResult, 53
CPLMimePart, 55
papszHeaders
CPLHTTPResult, 53
CPLMimePart, 55
ParseContext, 294
pasMimePart
CPLHTTPResult, 54
PointOnSurface
OGRPolygon, 219
OGRSurface, 294
Polygonize
OGRGeometry, 140
Print
OGRStyleTable, 292
projUV, 294
psChild
CPLXMLNode, 72
psNext
_CPLList, 51
CPLXMLNode, 73
pszContentType
CPLHTTPResult, 54
pszErrBuf
CPLHTTPResult, 54
pszValue
CPLXMLNode, 73

Reference
OGRDataSource, 91
OGRFeatureDefn, 114
OGRLayer, 171
OGRSpatialReference, 264
RegisterDriver
OGRSFDriverRegistrar, 229
Release
OGRDataSource, 91
OGRSpatialReference, 265
ReleaseResultSet
OGRDataSource, 91
RemoveDriver
CPLODBCDriverInstaller, 56
removeGeometry
OGRGeometryCollection, 154
RemoveStyle
OGRStyleTable, 292
ReorderField
OGRLayer, 171
ReorderFieldDefns
OGRFeatureDefn, 114
ReorderFields
OGRLayer, 172
ResetReading
OGRGenSQLResultsLayer, 124
OGRLayer, 172

SFRegion, 295
SaveStyleTable
OGRStyleTable, 292
segmentize
OGRGeometry, 141
OGRGeometryCollection, 154
OGRLineString, 190
OGRPolygon, 219
Set
OGRFieldDefn, 119
SetACEA
OGRSpatialReference, 265
SetAE
OGRSpatialReference, 265
SetAngularUnits
OGRSpatialReference, 265
SetAttributeFilter
OGRLayer, 173
SetAuthority
OGRSpatialReference, 265
SetAxes
OGRSpatialReference, 266
SetBonne
OGRSpatialReference, 266
SetCEA
OGRSpatialReference, 266
SetCS

OGRSpatialReference, 267
SetCompoundCS
OGRSpatialReference, 267
setCoordinateDimension
OGRGeometry, 141
OGRGeometryCollection, 154
OGRLineString, 191
OGRPoint, 209
OGRPolygon, 219
SetDefault
OGRFieldDefn, 119
SetDriver
OGRDataSource, 92
SetEC
OGRSpatialReference, 267
SetEckert
OGRSpatialReference, 267
SetEquirectangular
OGRSpatialReference, 267
SetEquirectangular2
OGRSpatialReference, 268
SetExtension
OGRSpatialReference, 268
SetFID
OGRFeature, 103
SetFeature
OGRLayer, 173
SetFeatureStyleString
OGRStyleMgr, 288
SetField
OGRFeature, 104–106
SetFrom
OGRFeature, 107
SetFromUserInput
OGRSpatialReference, 268
SetGEOS
OGRSpatialReference, 270
SetGH
OGRSpatialReference, 270
SetGS
OGRSpatialReference, 271
SetGaussSchreiberTMercator
OGRSpatialReference, 269
SetGeocCS
OGRSpatialReference, 269
SetGeogCS
OGRSpatialReference, 270
SetGeomType
OGRFeatureDefn, 115
SetGeometry
OGRFeature, 107
SetGeometryDirectly
OGRFeature, 108
SetGeometryIgnored
OGRFeatureDefn, 114
SetGnomonic
OGRSpatialReference, 271
SetHOM
OGRSpatialReference, 271
SetHOM2PNO
OGRSpatialReference, 271
SetIGH
OGRSpatialReference, 272
SetIWMPolyconic
OGRSpatialReference, 272
SetIgnored
OGRFieldDefn, 119
SetIgnoredFields
OGRLayer, 173
SetJustify
OGRFieldDefn, 120
SetKrovak
OGRSpatialReference, 272
SetLAEA
OGRSpatialReference, 272
SetLCC
OGRSpatialReference, 272
SetLCC1SP
OGRSpatialReference, 273
SetLCCB
OGRSpatialReference, 273
SetLinearUnits
OGRSpatialReference, 273
SetLinearUnitsAndUpdateParameters
OGRSpatialReference, 273
SetLocalCS
OGRSpatialReference, 274
SetMC
OGRSpatialReference, 274
SetMercator
OGRSpatialReference, 274
SetMollweide
OGRSpatialReference, 274
SetNZMG
OGRSpatialReference, 275
SetName
OGRFieldDefn, 120
SetNameValue
CPLStringList, 71
SetNextByIndex
OGRGenSQLResultsLayer, 124
OGRLayer, 174
SetNode
OGRSpatialReference, 274
SetNormProjParm
OGRSpatialReference, 275
setNumPoints
OGRLineString, 191
SetOS
OGRSpatialReference, 276
SetOrthographic
OGRSpatialReference, 275
SetPS
OGRSpatialReference, 277
setPoint
OGRLineString, 191, 192

setPoints
OGRLineString, 192
SetPolyconic
OGRSpatialReference, 276
SetPrecision
OGRFieldDefn, 120
SetProjCS
OGRSpatialReference, 276
SetProjParm
OGRSpatialReference, 277
SetProjection
OGRSpatialReference, 276
SetRobinson
OGRSpatialReference, 277
SetRoot
OGRSpatialReference, 277
SetSOC
OGRSpatialReference, 278
SetSinusoidal
OGRSpatialReference, 278
SetSpatialFilter
OGRLayer, 174
SetSpatialFilterRect
OGRLayer, 175
SetStatePlane
OGRSpatialReference, 278
SetStereographic
OGRSpatialReference, 278
SetStyleIgnored
OGRFeatureDefn, 115
SetStyleString
OGRFeature, 108
SetStyleStringDirectly
OGRFeature, 108
SetStyleTable
OGRDataSource, 92
OGRLayer, 175
SetStyleTableDirectly
OGRDataSource, 92
OGRLayer, 175
SetTM
OGRSpatialReference, 279
SetTMG
OGRSpatialReference, 279
SetTMSO
OGRSpatialReference, 279
SetTMVariant
OGRSpatialReference, 280
SetTOWGS84
OGRSpatialReference, 280
SetTPED
OGRSpatialReference, 280
SetTargetLinearUnits
OGRSpatialReference, 279
SetType
OGRFieldDefn, 120
SetUTM
OGRSpatialReference, 280
SetVDG
OGRSpatialReference, 281
SetValue
OGR_SRSNode, 80
SetVertCS
OGRSpatialReference, 281
SetWagner
OGRSpatialReference, 281
SetWellKnownGeogCS
OGRSpatialReference, 281
SetWidth
OGRFieldDefn, 121
setX
OGRPoint, 210
setY
OGRPoint, 210
setZ
OGRPoint, 210
Simplify
OGRGeometry, 141
SimplifyPreserveTopology
OGRGeometry, 142
Sort
CPLStringList, 71
StackContext, 295
StartPoint
OGRCurve, 84
OGRLineString, 192
StealGeometry
OGRFeature, 109
StealList
CPLStringList, 71
StripCTParms
OGRSpatialReference, 282
StripNodes
OGR_SRSNode, 80
StripVertical
OGRSpatialReference, 282
swapXY
OGRGeometry, 142
OGRGeometryCollection, 155
OGRLineString, 193
OGRPoint, 210
OGRPolygon, 219
swq_col_def, 295
swq_expr_node, 295
swq_field_list, 295
swq_join_def, 295
swq_op_registrar, 295
swq_operation, 296
swq_order_def, 296
swq_parse_context, 296
swq_select, 296
swq_summary, 296
swq_table_def, 296
SymDifference
OGRGeometry, 142
SymmetricDifference

OGRGeometry, 143
SyncToDisk
OGRDataSource, 92
OGRLayer, 176

TestCapability
OGRDataSource, 93
OGRGenSQLResultsLayer, 125
OGRLayer, 176
OGRSFDriver, 225
tm_unz_s, 296
tm_zip_s, 296
tolower
CPLString, 66
Touches
OGRGeometry, 143
toupper
CPLString, 66
Transform
OGRCoordinateTransformation, 82
OGRProj4CT, 221
transform
OGRGeometry, 143
OGRGeometryCollection, 155
OGRLineString, 193
OGRPoint, 210
OGRPolygon, 220
TransformEx
OGRCoordinateTransformation, 83
OGRProj4CT, 221
transformTo
OGRGeometry, 144
Trim
CPLString, 67

Union
OGRGeometry, 144
UnionCascaded
OGRGeometry, 145
UnsetField
OGRFeature, 109
unz_file_info_internal_s, 297
unz_file_info_s, 297
unz_file_pos_s, 297
unz_global_info_s, 297
unz_s, 297

VSIArchiveContent, 297
VSIArchiveEntry, 297
VSIArchiveEntryFileOffset, 298
VSIArchiveFilesystemHandler, 298
VSIArchiveReader, 298
VSIBufferedReaderHandle, 299
VSICacheChunk, 299
VSICachedFile, 299
VSICurlFilesystemHandler, 299
VSICurlHandle, 300
VSIFCloseL
cpl_vsi.h, 371
VSIFEofL
cpl_vsi.h, 372
VSIFFlushL
cpl_vsi.h, 372
VSIFOpenL
cpl_vsi.h, 373
VSIFPrintfL
cpl_vsi.h, 373
VSIFReadL
cpl_vsi.h, 374
VSIFReadMultiRangeL
cpl_vsi.h, 374
VSIFSeekL
cpl_vsi.h, 375
VSIFTellL
cpl_vsi.h, 375
VSIFTruncateL
cpl_vsi.h, 375
VSIFWriteL
cpl_vsi.h, 376
VSIFileFromMemBuffer
cpl_vsi.h, 372
VSIFileManager, 300
VSIFilesystemHandler, 300
VSIGZipFilesystemHandler, 301
VSIGZipHandle, 301
VSIGZipWriteHandle, 301
VSIGetMemFileBuffer
cpl_vsi.h, 376
VSIInstallCurlFileHandler
cpl_vsi.h, 377
VSIInstallGZipFileHandler
cpl_vsi.h, 377
VSIInstallMemFileHandler
cpl_vsi.h, 377
VSIInstallSparseFileHandler
cpl_vsi.h, 378
VSIInstallStdinHandler
cpl_vsi.h, 379
VSIInstallStdoutHandler
cpl_vsi.h, 379
VSIInstallSubFileHandler
cpl_vsi.h, 379
VSIInstallTarFileHandler
cpl_vsi.h, 380
VSIInstallZipFileHandler
cpl_vsi.h, 380
VSIIsCaseSensitiveFS
cpl_vsi.h, 381
VSIMalloc2
cpl_vsi.h, 381
VSIMalloc3
cpl_vsi.h, 381
VSIMemFile, 302
VSIMemFilesystemHandler, 302
VSIMemHandle, 302
VSIMkdir
cpl_vsi.h, 381

VSIReadDir
cpl_vsi.h, 382
VSIRename
cpl_vsi.h, 382
VSIRmdir
cpl_vsi.h, 383
VSISparseFileFilesystemHandler, 302
VSISparseFileHandle, 303
VSIStatExL
cpl_vsi.h, 383
VSIStatL
cpl_vsi.h, 384
VSIStdinFilesystemHandler, 303
VSIStdinHandle, 303
VSIStdoutFilesystemHandler, 303
VSIStdoutHandle, 304
VSIStdoutRedirectFilesystemHandler, 304
VSIStdoutRedirectHandle, 304
VSISubFileFilesystemHandler, 305
VSISubFileHandle, 305
VSITarEntryFileOffset, 305
VSITarFilesystemHandler, 305
VSITarReader, 306
VSIUnixStdioFilesystemHandler, 306
VSIUnixStdioHandle, 306
VSIUnlink
cpl_vsi.h, 384
VSIVirtualHandle, 307
VSIZipEntryFileOffset, 307
VSIZipFilesystemHandler, 308
VSIZipReader, 308
VSIZipWriteHandle, 308
Validate
OGRSpatialReference, 283
Value
OGRCurve, 85
OGRLineString, 193

Within
OGRGeometry, 145
wkbGeometryCollection
ogr_core.h, 486
wkbGeometryCollection25D
ogr_core.h, 486
wkbLineString
ogr_core.h, 486
wkbLineString25D
ogr_core.h, 486
wkbLinearRing
ogr_core.h, 486
wkbMultiLineString
ogr_core.h, 486
wkbMultiLineString25D
ogr_core.h, 486
wkbMultiPoint
ogr_core.h, 486
wkbMultiPoint25D
ogr_core.h, 486
wkbMultiPolygon
ogr_core.h, 486
wkbMultiPolygon25D
ogr_core.h, 486
wkbNone
ogr_core.h, 486
wkbPoint
ogr_core.h, 486
wkbPoint25D
ogr_core.h, 486
wkbPolygon
ogr_core.h, 486
wkbPolygon25D
ogr_core.h, 486
WkbSize
OGRGeometry, 145
OGRGeometryCollection, 155
OGRLinearRing, 180
OGRLineString, 194
OGRPoint, 211
OGRPolygon, 220
wkbUnknown
ogr_core.h, 486
WriteFuncStruct, 309

yyalloc, 309

zip_fileinfo, 309
zip_internal, 309
zlib_filefunc_def_s, 309