

OGR

Contents

1	OGR Simple Feature Library	1
2	OGR API Tutorial	5
2.1	Reading From OGR	5
2.2	Writing To OGR	12
3	OGR Architecture	21
3.1	Class Overview	21
3.2	Geometry	22
3.3	Spatial Reference	22
3.4	Feature / Feature Definition	23
3.5	Layer	23
3.6	Data Source	24
3.7	Drivers	25
4	OGR Driver Implementation Tutorial	27
4.1	Overall Approach	27
4.2	Contents	27
4.3	Implementing OGRSFDriver	27
4.4	Basic Read Only Data Source	29
4.5	Read Only Layer	31
5	OGR SQL	35
5.1	Supported SQL syntax	35
5.2	SELECT	36
5.2.1	Field List Operators	36
5.2.1.1	Using the field name alias	37

5.2.1.2	Changing the type of the fields	38
5.2.1.3	Field List Limitations	38
5.2.2	WHERE	38
5.2.3	WHERE Limitations	39
5.2.4	ORDER BY	40
5.2.5	JOINS	40
5.2.6	JOIN Limitations	41
5.3	SPECIAL FIELDS	42
5.3.1	FID	42
5.3.2	OGR_GEOMETRY	42
5.3.3	OGR_GEOM_WKT	42
5.3.4	OGR_GEOM_AREA	43
5.3.5	OGR_STYLE	43
5.4	CREATE INDEX	43
5.4.1	Index Limitations	43
5.5	DROP INDEX	44
5.6	ExecuteSQL()	44
5.7	Non-OGR SQL	44
6	OGR Projections Tutorial	45
6.1	Introduction	45
6.2	Defining a Geographic Coordinate System	45
6.3	Defining a Projected Coordinate System	47
6.4	Querying Coordinate System	48
6.5	Coordinate Transformation	48
6.6	Alternate Interfaces	50
6.7	Internal Implementation	51
7	Deprecated List	53
8	Directory Hierarchy	55
8.1	Directories	55
9	Class Index	57
9.1	Class Hierarchy	57

10 Class Index	61
10.1 Class List	61
11 File Index	65
11.1 File List	65
12 Directory Documentation	67
12.1 ogrsf_frmts/generic/ Directory Reference	67
12.2 ogrsf_frmts/ Directory Reference	67
12.3 /builddir/build/BUILD/gdal-1.7.3-fedora/port/ Directory Reference	67
13 Class Documentation	71
13.1 _CPLHashSet Struct Reference	71
13.2 _CPLList Struct Reference	71
13.2.1 Detailed Description	71
13.2.2 Member Data Documentation	71
13.2.2.1 pData	71
13.2.2.2 psNext	72
13.3 _CPLQuadTree Struct Reference	72
13.4 _QuadTreeNode Struct Reference	72
13.5 _sPolyExtended Struct Reference	72
13.6 CPLErrorContext Struct Reference	72
13.7 CPLHTTPResult Struct Reference	72
13.7.1 Detailed Description	73
13.7.2 Member Data Documentation	73
13.7.2.1 nDataLen	73
13.7.2.2 nMimePartCount	73
13.7.2.3 nStatus	73
13.7.2.4 pabyData	73
13.7.2.5 pasMimePart	73
13.7.2.6 pszContentType	74
13.7.2.7 pszErrBuf	74
13.8 CPLKeywordParser Class Reference	74
13.9 CPLLocaleC Class Reference	74
13.10 CPLMimePart Struct Reference	74

13.10.1 Detailed Description	75
13.10.2 Member Data Documentation	75
13.10.2.1 nDataLen	75
13.10.2.2 pabyData	75
13.10.2.3 papszHeaders	75
13.11 CPLMutexHolder Class Reference	75
13.12 CPLODBCDriverInstaller Class Reference	75
13.12.1 Detailed Description	76
13.12.2 Member Function Documentation	76
13.12.2.1 InstallDriver	76
13.12.2.2 RemoveDriver	76
13.13 CPLODBCSession Class Reference	77
13.13.1 Detailed Description	77
13.13.2 Member Function Documentation	77
13.13.2.1 EstablishSession	77
13.13.2.2 GetLastError	77
13.14 CPLODBCStatement Class Reference	78
13.14.1 Detailed Description	79
13.14.2 Member Function Documentation	79
13.14.2.1 Append	79
13.14.2.2 Append	79
13.14.2.3 Append	79
13.14.2.4 AppendEscaped	79
13.14.2.5 Appendf	80
13.14.2.6 Clear	80
13.14.2.7 DumpResult	80
13.14.2.8 ExecuteSQL	81
13.14.2.9 Fetch	81
13.14.2.10 GetColCount	81
13.14.2.11 GetColData	82
13.14.2.12 GetColData	82
13.14.2.13 GetColId	83
13.14.2.14 GetColName	83
13.14.2.15 GetColNullable	83

13.14.2.16GetColPrecision	83
13.14.2.17GetColSize	84
13.14.2.18GetColType	84
13.14.2.19GetColTypeName	84
13.14.2.20GetColumns	85
13.14.2.21GetPrimaryKeys	85
13.14.2.22GetTables	86
13.14.2.23GetTypeMapping	86
13.14.2.24GetTypeNames	86
13.15CPLRectObj Struct Reference	87
13.16CPLSharedFileInfo Struct Reference	87
13.17CPLStdCallThreadInfo Struct Reference	87
13.18CPLString Class Reference	87
13.18.1 Member Function Documentation	88
13.18.1.1 FormatC	88
13.18.1.2 Trim	88
13.19CPLXMLNode Struct Reference	88
13.19.1 Detailed Description	89
13.19.2 Member Data Documentation	89
13.19.2.1 eType	89
13.19.2.2 psChild	89
13.19.2.3 psNext	89
13.19.2.4 pszValue	90
13.20ctb Struct Reference	90
13.21DefaultCSVFileNameTLS Struct Reference	90
13.22errHandler Struct Reference	91
13.23file_in_zip_read_info_s Struct Reference	91
13.24FindFileTLS Struct Reference	91
13.25GZipSnapshot Struct Reference	91
13.26OGR_SRSNode Class Reference	91
13.26.1 Detailed Description	92
13.26.2 Constructor & Destructor Documentation	92
13.26.2.1 OGR_SRSNode	92
13.26.3 Member Function Documentation	92

13.26.3.1 AddChild	92
13.26.3.2 applyRemapper	93
13.26.3.3 Clone	93
13.26.3.4 DestroyChild	94
13.26.3.5 exportToWkt	94
13.26.3.6 FindChild	94
13.26.3.7 GetChild	95
13.26.3.8 GetChildCount	96
13.26.3.9 GetNode	96
13.26.3.10 GetValue	96
13.26.3.11 importFromWkt	97
13.26.3.12 InsertChild	97
13.26.3.13 MakeValueSafe	98
13.26.3.14 SetValue	98
13.26.3.15 StripNodes	98
13.27 ogr_style_param Struct Reference	99
13.28 ogr_style_value Struct Reference	99
13.29 OGRAttrIndex Class Reference	99
13.30 OGRCoordinateTransformation Class Reference	99
13.30.1 Detailed Description	100
13.30.2 Member Function Documentation	100
13.30.2.1 DestroyCT	100
13.30.2.2 GetSourceCS	101
13.30.2.3 GetTargetCS	101
13.30.2.4 Transform	101
13.30.2.5 TransformEx	102
13.31 OGRCurve Class Reference	102
13.31.1 Detailed Description	103
13.31.2 Member Function Documentation	103
13.31.2.1 EndPoint	103
13.31.2.2 get_IsClosed	104
13.31.2.3 get_Length	104
13.31.2.4 StartPoint	104
13.31.2.5 Value	104

13.32OGRDataSource Class Reference	105
13.32.1 Detailed Description	106
13.32.2 Member Function Documentation	106
13.32.2.1 CopyLayer	106
13.32.2.2 CreateLayer	107
13.32.2.3 DeleteLayer	108
13.32.2.4 Dereference	108
13.32.2.5 DestroyDataSource	109
13.32.2.6 ExecuteSQL	109
13.32.2.7 GetDriver	110
13.32.2.8 GetLayer	110
13.32.2.9 GetLayerByName	110
13.32.2.10GetLayerCount	111
13.32.2.11GetName	111
13.32.2.12GetRefCount	111
13.32.2.13GetStyleTable	112
13.32.2.14GetSummaryRefCount	112
13.32.2.15Reference	112
13.32.2.16Release	112
13.32.2.17ReleaseResultSet	113
13.32.2.18SetDriver	113
13.32.2.19SetStyleTable	113
13.32.2.20SetStyleTableDirectly	114
13.32.2.21SyncToDisk	114
13.32.2.22TestCapability	114
13.33OGREnvelope Class Reference	115
13.33.1 Detailed Description	115
13.34OGRFeature Class Reference	115
13.34.1 Detailed Description	118
13.34.2 Constructor & Destructor Documentation	118
13.34.2.1 OGRFeature	118
13.34.3 Member Function Documentation	118
13.34.3.1 Clone	118
13.34.3.2 CreateFeature	118

13.34.3.3 DestroyFeature	119
13.34.3.4 DumpReadable	119
13.34.3.5 Equal	120
13.34.3.6 GetDefnRef	120
13.34.3.7 GetFID	120
13.34.3.8 GetFieldAsBinary	121
13.34.3.9 GetFieldAsDateTime	121
13.34.3.10GetFieldAsDouble	122
13.34.3.11GetFieldAsDoubleList	122
13.34.3.12GetFieldAsInteger	123
13.34.3.13GetFieldAsIntegerList	123
13.34.3.14GetFieldAsString	123
13.34.3.15GetFieldAsStringList	124
13.34.3.16GetFieldCount	124
13.34.3.17GetFieldDefnRef	125
13.34.3.18GetFieldIndex	125
13.34.3.19GetGeometryRef	125
13.34.3.20GetRawFieldRef	126
13.34.3.21GetStyleString	126
13.34.3.22sFieldSet	126
13.34.3.23SetFID	127
13.34.3.24SetField	127
13.34.3.25SetField	128
13.34.3.26SetField	128
13.34.3.27SetField	128
13.34.3.28SetField	129
13.34.3.29SetField	129
13.34.3.30SetField	129
13.34.3.31SetField	130
13.34.3.32SetField	130
13.34.3.33SetFrom	130
13.34.3.34SetFrom	131
13.34.3.35SetGeometry	132
13.34.3.36SetGeometryDirectly	132

13.34.3.37SetStyleString	133
13.34.3.38SetStyleStringDirectly	133
13.34.3.39StealGeometry	133
13.34.3.40UnsetField	133
13.35OGRFeatureDefn Class Reference	134
13.35.1 Detailed Description	135
13.35.2 Constructor & Destructor Documentation	135
13.35.2.1 OGRFeatureDefn	135
13.35.3 Member Function Documentation	135
13.35.3.1 AddFieldDefn	135
13.35.3.2 Clone	136
13.35.3.3 Dereference	136
13.35.3.4 GetFieldCount	136
13.35.3.5 GetFieldDefn	136
13.35.3.6 GetFieldIndex	137
13.35.3.7 GetGeomType	137
13.35.3.8 GetName	138
13.35.3.9 GetReferenceCount	138
13.35.3.10Reference	138
13.35.3.11SetGeomType	138
13.36OGRFeatureQuery Class Reference	139
13.36.1 Member Function Documentation	139
13.36.1.1 GetUsedFields	139
13.37OGRField Union Reference	139
13.37.1 Detailed Description	140
13.38OGRFieldDefn Class Reference	140
13.38.1 Detailed Description	141
13.38.2 Constructor & Destructor Documentation	141
13.38.2.1 OGRFieldDefn	141
13.38.2.2 OGRFieldDefn	141
13.38.3 Member Function Documentation	141
13.38.3.1 GetFieldTypeNames	141
13.38.3.2 GetJustify	142
13.38.3.3 GetNameRef	142

13.38.3.4 GetPrecision	142
13.38.3.5 GetType	142
13.38.3.6 GetWidth	143
13.38.3.7 Set	143
13.38.3.8 SetDefault	143
13.38.3.9 SetJustify	144
13.38.3.10SetName	144
13.38.3.11SetPrecision	144
13.38.3.12SetType	144
13.38.3.13SetWidth	145
13.39OGRGenSQLResultsLayer Class Reference	145
13.39.1 Member Function Documentation	146
13.39.1.1 GetExtent	146
13.39.1.2 GetFeature	146
13.39.1.3 GetFeatureCount	147
13.39.1.4 GetLayerDefn	148
13.39.1.5 GetNextFeature	148
13.39.1.6 GetSpatialFilter	148
13.39.1.7 GetSpatialRef	149
13.39.1.8 ResetReading	149
13.39.1.9 SetNextByIndex	149
13.39.1.10TestCapability	150
13.40OGRGeometry Class Reference	151
13.40.1 Detailed Description	154
13.40.2 Member Function Documentation	154
13.40.2.1 assignSpatialReference	154
13.40.2.2 Buffer	155
13.40.2.3 clone	155
13.40.2.4 closeRings	156
13.40.2.5 Contains	156
13.40.2.6 ConvexHull	156
13.40.2.7 Crosses	157
13.40.2.8 Difference	157
13.40.2.9 Disjoint	157

13.40.2.10Distance	158
13.40.2.11dumpReadable	158
13.40.2.12empty	159
13.40.2.13Equals	159
13.40.2.14exportToGML	159
13.40.2.15exportToJson	160
13.40.2.16exportToKML	160
13.40.2.17exportToWkb	160
13.40.2.18exportToWkt	161
13.40.2.19flattenTo2D	161
13.40.2.20getBoundary	161
13.40.2.21getCoordinateDimension	162
13.40.2.22getDimension	162
13.40.2.23getEnvelope	163
13.40.2.24getGeometryName	163
13.40.2.25getGeometryType	163
13.40.2.26getSpatialReference	164
13.40.2.27importFromWkb	164
13.40.2.28importFromWkt	165
13.40.2.29Intersection	165
13.40.2.30Intersects	166
13.40.2.31IsEmpty	166
13.40.2.32IsRing	167
13.40.2.33IsSimple	167
13.40.2.34IsValid	167
13.40.2.35Overlaps	168
13.40.2.36segmentize	168
13.40.2.37setCoordinateDimension	168
13.40.2.38SymmetricDifference	169
13.40.2.39Touches	169
13.40.2.40transform	170
13.40.2.41transformTo	170
13.40.2.42Union	171
13.40.2.43Within	171

13.40.2.44	WkbSize	172
13.41	OGRGeometryCollection Class Reference	172
13.41.1	Detailed Description	174
13.41.2	Member Function Documentation	174
13.41.2.1	addGeometry	174
13.41.2.2	addGeometryDirectly	174
13.41.2.3	clone	175
13.41.2.4	closeRings	175
13.41.2.5	empty	176
13.41.2.6	Equals	176
13.41.2.7	exportToWkb	176
13.41.2.8	exportToWkt	177
13.41.2.9	flattenTo2D	177
13.41.2.10	get_Area	177
13.41.2.11	getDimension	178
13.41.2.12	getEnvelope	178
13.41.2.13	getGeometryName	178
13.41.2.14	getGeometryRef	179
13.41.2.15	getGeometryType	179
13.41.2.16	getNumGeometries	180
13.41.2.17	importFromWkb	180
13.41.2.18	importFromWkt	181
13.41.2.19	isEmpty	181
13.41.2.20	removeGeometry	182
13.41.2.21	segmentize	182
13.41.2.22	setCoordinateDimension	183
13.41.2.23	transform	183
13.41.2.24	WkbSize	183
13.42	OGRGeometryFactory Class Reference	184
13.42.1	Detailed Description	185
13.42.2	Member Function Documentation	185
13.42.2.1	approximateArcAngles	185
13.42.2.2	createFromFgf	186
13.42.2.3	createFromGML	187

13.42.2.4 createFromWkb	187
13.42.2.5 createFromWkt	188
13.42.2.6 createGeometry	188
13.42.2.7 destroyGeometry	189
13.42.2.8 forceToMultiLineString	189
13.42.2.9 forceToMultiPoint	190
13.42.2.10 forceToMultiPolygon	190
13.42.2.11 forceToPolygon	190
13.42.2.12 haveGEOS	191
13.42.2.13 organizePolygons	191
13.43 OGRLayer Class Reference	192
13.43.1 Detailed Description	194
13.43.2 Member Function Documentation	194
13.43.2.1 CreateFeature	194
13.43.2.2 CreateField	194
13.43.2.3 DeleteFeature	195
13.43.2.4 Dereference	195
13.43.2.5 GetExtent	195
13.43.2.6 GetFeature	196
13.43.2.7 GetFeatureCount	197
13.43.2.8 GetFIDColumn	197
13.43.2.9 GetGeometryColumn	197
13.43.2.10 GetInfo	198
13.43.2.11 GetLayerDefn	198
13.43.2.12 GetNextFeature	198
13.43.2.13 GetRefCount	199
13.43.2.14 GetSpatialFilter	199
13.43.2.15 GetSpatialRef	199
13.43.2.16 GetStyleTable	200
13.43.2.17 Reference	200
13.43.2.18 ResetReading	200
13.43.2.19 SetAttributeFilter	201
13.43.2.20 SetFeature	201
13.43.2.21 SetNextByIndex	202

13.43.2.22SetSpatialFilter	202
13.43.2.23SetSpatialFilterRect	203
13.43.2.24SetStyleTable	203
13.43.2.25SetStyleTableDirectly	204
13.43.2.26SyncToDisk	204
13.43.2.27TestCapability	204
13.44OGRLayerAttrIndex Class Reference	206
13.45OGRLinearRing Class Reference	206
13.45.1 Detailed Description	207
13.45.2 Member Function Documentation	207
13.45.2.1 clone	207
13.45.2.2 closeRings	208
13.45.2.3 exportToWkb	208
13.45.2.4 get_Area	208
13.45.2.5 getGeometryName	209
13.45.2.6 importFromWkb	209
13.45.2.7 isClockwise	210
13.45.2.8 WkbSize	210
13.46OGRLineString Class Reference	210
13.46.1 Detailed Description	212
13.46.2 Member Function Documentation	213
13.46.2.1 addPoint	213
13.46.2.2 addPoint	213
13.46.2.3 addSubLineString	213
13.46.2.4 clone	214
13.46.2.5 empty	214
13.46.2.6 EndPoint	214
13.46.2.7 Equals	215
13.46.2.8 exportToWkb	215
13.46.2.9 exportToWkt	215
13.46.2.10flattenTo2D	216
13.46.2.11get_Length	216
13.46.2.12getDimension	216
13.46.2.13getEnvelope	217

13.46.2.14	getGeometryName	217
13.46.2.15	getGeometryType	217
13.46.2.16	getNumPoints	218
13.46.2.17	getPoint	218
13.46.2.18	getPoints	218
13.46.2.19	getX	218
13.46.2.20	getY	219
13.46.2.21	getZ	219
13.46.2.22	importFromWkb	220
13.46.2.23	importFromWkt	220
13.46.2.24	isEmpty	221
13.46.2.25	segmentize	221
13.46.2.26	setCoordinateDimension	221
13.46.2.27	setNumPoints	222
13.46.2.28	setPoint	222
13.46.2.29	setPoint	222
13.46.2.30	setPoints	223
13.46.2.31	setPoints	223
13.46.2.32	startPoint	224
13.46.2.33	transform	224
13.46.2.34	value	224
13.46.2.35	WkbSize	225
13.47	OGRMIAAttrIndex Class Reference	225
13.48	OGRMILayerAttrIndex Class Reference	226
13.49	OGRMultiLineString Class Reference	226
13.49.1	Detailed Description	227
13.49.2	Member Function Documentation	227
13.49.2.1	addGeometryDirectly	227
13.49.2.2	clone	227
13.49.2.3	exportToWkt	228
13.49.2.4	getGeometryName	228
13.49.2.5	getGeometryType	228
13.49.2.6	importFromWkt	229
13.50	OGRMultiPoint Class Reference	229

13.50.1 Detailed Description	230
13.50.2 Member Function Documentation	230
13.50.2.1 addGeometryDirectly	230
13.50.2.2 clone	231
13.50.2.3 exportToWkt	231
13.50.2.4 getGeometryName	232
13.50.2.5 getGeometryType	232
13.50.2.6 importFromWkt	232
13.51 OGRMultiPolygon Class Reference	233
13.51.1 Detailed Description	234
13.51.2 Member Function Documentation	234
13.51.2.1 addGeometryDirectly	234
13.51.2.2 clone	234
13.51.2.3 exportToWkt	235
13.51.2.4 get_Area	235
13.51.2.5 getGeometryName	235
13.51.2.6 getGeometryType	236
13.51.2.7 importFromWkt	236
13.52 OGRPoint Class Reference	237
13.52.1 Detailed Description	238
13.52.2 Member Function Documentation	238
13.52.2.1 clone	238
13.52.2.2 empty	239
13.52.2.3 Equals	239
13.52.2.4 exportToWkb	239
13.52.2.5 exportToWkt	240
13.52.2.6 flattenTo2D	240
13.52.2.7 getDimension	240
13.52.2.8 getEnvelope	241
13.52.2.9 getGeometryName	241
13.52.2.10 getGeometryType	241
13.52.2.11 getX	242
13.52.2.12 getY	242
13.52.2.13 getZ	242

13.52.2.14importFromWkb	242
13.52.2.15importFromWkt	243
13.52.2.16isEmpty	243
13.52.2.17setCoordinateDimension	244
13.52.2.18setX	244
13.52.2.19setY	244
13.52.2.20setZ	244
13.52.2.21transform	245
13.52.2.22WkbSize	245
13.53OGRPolygon Class Reference	246
13.53.1 Detailed Description	247
13.53.2 Member Function Documentation	247
13.53.2.1 addRing	247
13.53.2.2 addRingDirectly	248
13.53.2.3 Centroid	248
13.53.2.4 clone	249
13.53.2.5 closeRings	249
13.53.2.6 empty	249
13.53.2.7 Equals	249
13.53.2.8 exportToWkb	250
13.53.2.9 exportToWkt	250
13.53.2.10flattenTo2D	250
13.53.2.11get_Area	251
13.53.2.12getDimension	251
13.53.2.13getEnvelope	251
13.53.2.14getExteriorRing	252
13.53.2.15getGeometryName	252
13.53.2.16getGeometryType	252
13.53.2.17getInteriorRing	253
13.53.2.18getNumInteriorRings	253
13.53.2.19importFromWkb	253
13.53.2.20importFromWkt	254
13.53.2.21isEmpty	254
13.53.2.22PointOnSurface	255

13.53.2.23	segmentize	255
13.53.2.24	setCoordinateDimension	255
13.53.2.25	transform	256
13.53.2.26	WkbSize	256
13.54	OGRProj4CT Class Reference	257
13.54.1	Member Function Documentation	257
13.54.1.1	GetSourceCS	257
13.54.1.2	GetTargetCS	257
13.54.1.3	Transform	258
13.54.1.4	TransformEx	258
13.55	OGRProj4Datum Struct Reference	259
13.56	OGRRawPoint Class Reference	259
13.56.1	Detailed Description	259
13.57	OGRSFDriver Class Reference	259
13.57.1	Detailed Description	260
13.57.2	Member Function Documentation	260
13.57.2.1	CopyDataSource	260
13.57.2.2	CreateDataSource	260
13.57.2.3	DeleteDataSource	261
13.57.2.4	GetName	261
13.57.2.5	Open	262
13.57.2.6	TestCapability	262
13.58	OGRSFDriverRegistrar Class Reference	263
13.58.1	Detailed Description	264
13.58.2	Member Function Documentation	264
13.58.2.1	AutoLoadDrivers	264
13.58.2.2	GetDriver	264
13.58.2.3	GetDriverByName	265
13.58.2.4	GetDriverCount	265
13.58.2.5	GetOpenDS	265
13.58.2.6	GetOpenDSCount	266
13.58.2.7	GetRegistrar	266
13.58.2.8	Open	266
13.58.2.9	RegisterDriver	267

13.59 OGRSpatialReference Class Reference	268
13.59.1 Detailed Description	274
13.59.2 Constructor & Destructor Documentation	274
13.59.2.1 OGRSpatialReference	274
13.59.2.2 ~OGRSpatialReference	275
13.59.3 Member Function Documentation	275
13.59.3.1 AutIdentifyEPSG	275
13.59.3.2 Clear	275
13.59.3.3 Clone	276
13.59.3.4 CloneGeogCS	276
13.59.3.5 CopyGeogCSFrom	276
13.59.3.6 Dereference	276
13.59.3.7 DestroySpatialReference	277
13.59.3.8 EPSGTreatsAsLatLong	277
13.59.3.9 exportToERM	277
13.59.3.10 exportToMICoordSys	278
13.59.3.11 exportToPanorama	278
13.59.3.12 exportToPCI	279
13.59.3.13 exportToPrettyWkt	279
13.59.3.14 exportToProj4	280
13.59.3.15 exportToUSGS	280
13.59.3.16 exportToWkt	281
13.59.3.17 exportToXML	281
13.59.3.18 FindProjParm	282
13.59.3.19 Fixup	282
13.59.3.20 FixupOrdering	283
13.59.3.21 GetAngularUnits	283
13.59.3.22 GetAttrNode	283
13.59.3.23 GetAttrValue	284
13.59.3.24 GetAuthorityCode	285
13.59.3.25 GetAuthorityName	285
13.59.3.26 GetAxis	286
13.59.3.27 GetExtension	286
13.59.3.28 GetInvFlattening	287

13.59.3.29GetLinearUnits	287
13.59.3.30GetNormProjParm	288
13.59.3.31GetPrimeMeridian	288
13.59.3.32GetProjParm	289
13.59.3.33GetReferenceCount	289
13.59.3.34GetSemiMajor	289
13.59.3.35GetSemiMinor	290
13.59.3.36GetTOWGS84	290
13.59.3.37GetUTMZone	291
13.59.3.38ImportFromDict	291
13.59.3.39ImportFromEPSG	292
13.59.3.40ImportFromEPSGA	292
13.59.3.41ImportFromERM	293
13.59.3.42ImportFromESRI	293
13.59.3.43ImportFromMICoordSys	294
13.59.3.44ImportFromOzi	294
13.59.3.45ImportFromPanorama	295
13.59.3.46ImportFromPCI	296
13.59.3.47ImportFromProj4	297
13.59.3.48ImportFromUrl	298
13.59.3.49ImportFromURN	299
13.59.3.50ImportFromUSGS	299
13.59.3.51ImportFromWkt	304
13.59.3.52ImportFromWMSAUTO	304
13.59.3.53ImportFromXML	305
13.59.3.54IsGeographic	305
13.59.3.55IsLocal	305
13.59.3.56IsProjected	306
13.59.3.57IsSame	306
13.59.3.58IsSameGeogCS	306
13.59.3.59morphFromESRI	307
13.59.3.60morphToESRI	307
13.59.3.61Reference	308
13.59.3.62Release	308

13.59.3.63SetACEA	308
13.59.3.64SetAE	308
13.59.3.65SetAngularUnits	308
13.59.3.66SetAuthority	309
13.59.3.67SetAxes	309
13.59.3.68SetBonne	310
13.59.3.69SetCEA	310
13.59.3.70SetCS	310
13.59.3.71SetEC	311
13.59.3.72SetEckert	311
13.59.3.73SetEquirectangular	311
13.59.3.74SetEquirectangular2	311
13.59.3.75SetExtension	311
13.59.3.76SetFromUserInput	312
13.59.3.77SetGaussSchreiberTMercator	313
13.59.3.78SetGeogCS	313
13.59.3.79SetGEOS	314
13.59.3.80SetGH	314
13.59.3.81SetGnomonic	314
13.59.3.82SetGS	314
13.59.3.83SetHOM	315
13.59.3.84SetHOM2PNO	315
13.59.3.85SetIWMPolyconic	316
13.59.3.86SetKrovak	316
13.59.3.87SetLAEA	316
13.59.3.88SetLCC	316
13.59.3.89SetLCC1SP	317
13.59.3.90SetLCCB	317
13.59.3.91SetLinearUnits	317
13.59.3.92SetLinearUnitsAndUpdateParameters	317
13.59.3.93SetLocalCS	318
13.59.3.94SetMC	318
13.59.3.95SetMercator	319
13.59.3.96SetMollweide	319

13.59.3.97	SetNode	319
13.59.3.98	SetNormProjParm	320
13.59.3.99	SetNZMG	320
13.59.3.100	SetOrthographic	320
13.59.3.101	SetOS	321
13.59.3.102	SetPolyconic	321
13.59.3.103	SetProjCS	321
13.59.3.104	SetProjection	321
13.59.3.105	SetProjParm	322
13.59.3.106	SetPS	322
13.59.3.107	SetRobinson	323
13.59.3.108	SetRoot	323
13.59.3.109	SetSinusoidal	323
13.59.3.110	SetSOC	323
13.59.3.111	SetStatePlane	323
13.59.3.112	SetStereographic	324
13.59.3.113	SetTM	324
13.59.3.114	SetTMG	325
13.59.3.115	SetTMSO	325
13.59.3.116	SetTMVariant	325
13.59.3.117	SetTOWGS84	325
13.59.3.118	SetTPED	326
13.59.3.119	SetUTM	326
13.59.3.120	SetVDG	326
13.59.3.121	SetWagner	327
13.59.3.122	SetWellKnownGeogCS	327
13.59.3.123	StripCTParms	328
13.59.3.124	StripVertical	328
13.59.3.125	Validate	328
13.60	OGRStyleBrush Class Reference	329
13.60.1	Detailed Description	329
13.61	OGRStyleLabel Class Reference	329
13.61.1	Detailed Description	330
13.62	OGRStyleMgr Class Reference	330

13.62.1 Detailed Description	331
13.62.2 Constructor & Destructor Documentation	331
13.62.2.1 OGRStyleMgr	331
13.62.2.2 ~OGRStyleMgr	331
13.62.3 Member Function Documentation	331
13.62.3.1 AddPart	331
13.62.3.2 GetPart	331
13.62.3.3 GetPartCount	332
13.62.3.4 GetStyleName	332
13.62.3.5 InitFromFeature	332
13.62.3.6 InitStyleString	333
13.62.3.7 SetFeatureStyleString	333
13.63 OGRStylePen Class Reference	334
13.63.1 Detailed Description	334
13.64 OGRStyleSymbol Class Reference	334
13.64.1 Detailed Description	334
13.65 OGRStyleTable Class Reference	334
13.65.1 Detailed Description	335
13.65.2 Member Function Documentation	335
13.65.2.1 AddStyle	335
13.65.2.2 Clone	336
13.65.2.3 Find	336
13.65.2.4 GetStyleName	336
13.65.2.5 IsExist	337
13.65.2.6 LoadStyleTable	337
13.65.2.7 ModifyStyle	337
13.65.2.8 Print	337
13.65.2.9 RemoveStyle	338
13.65.2.10 SaveStyleTable	338
13.66 OGRStyleTool Class Reference	338
13.66.1 Detailed Description	339
13.67 OGRSurface Class Reference	339
13.67.1 Detailed Description	339
13.67.2 Member Function Documentation	339

13.67.2.1 Centroid	339
13.67.2.2 get_Area	340
13.67.2.3 PointOnSurface	340
13.68OZIDatums Struct Reference	341
13.69ParseContext Struct Reference	341
13.70PCIDatums Struct Reference	341
13.71projUV Struct Reference	341
13.72StackContext Struct Reference	341
13.73swq_col_def Struct Reference	341
13.74swq_field_list Struct Reference	341
13.75swq_field_op Struct Reference	342
13.76swq_join_def Struct Reference	342
13.77swq_order_def Struct Reference	342
13.78swq_select Struct Reference	342
13.79swq_summary Struct Reference	342
13.80swq_table_def Struct Reference	342
13.81tm_unz_s Struct Reference	343
13.82unz_file_info_internal_s Struct Reference	343
13.83unz_file_info_s Struct Reference	343
13.84unz_file_pos_s Struct Reference	343
13.85unz_global_info_s Struct Reference	343
13.86unz_s Struct Reference	343
13.87VSIFileManager Class Reference	343
13.88VSIFilesystemHandler Class Reference	344
13.89VSIGZipFilesystemHandler Class Reference	344
13.90VSIGZipHandle Class Reference	344
13.91VSIGZipWriteHandle Class Reference	345
13.92VSIMemFile Class Reference	345
13.93VSIMemFilesystemHandler Class Reference	345
13.94VSIMemHandle Class Reference	345
13.95VSIStdoutFilesystemHandler Class Reference	346
13.96VSIStdoutHandle Class Reference	346
13.97VSISubFileFilesystemHandler Class Reference	346
13.98VSISubFileHandle Class Reference	347

13.99	VSIUnixStdioFilesystemHandler Class Reference	347
13.100	VSIUnixStdioHandle Class Reference	347
13.101	VSVirtualHandle Class Reference	348
13.102	VSIZipFilesystemHandler Class Reference	348
13.103	ZIPContent Struct Reference	348
13.104	ZIPEntry Struct Reference	349
13.105	lib_filefunc_def_s Struct Reference	349
14	File Documentation	351
14.1	cpl_conv.h File Reference	351
14.1.1	Detailed Description	353
14.1.2	Function Documentation	353
14.1.2.1	CPLAtof	353
14.1.2.2	CPLAtofDelim	354
14.1.2.3	CPLAtofM	354
14.1.2.4	CPLCalloc	354
14.1.2.5	CPLCheckForFile	355
14.1.2.6	CPLCleanTrailingSlash	355
14.1.2.7	CPLCloseShared	356
14.1.2.8	CPLCorrespondingPaths	356
14.1.2.9	CPLDecToPackedDMS	357
14.1.2.10	CPLDumpSharedList	357
14.1.2.11	CPLExtractRelativePath	358
14.1.2.12	CPLFGets	358
14.1.2.13	CPLFormCIFilename	359
14.1.2.14	CPLFormFilename	359
14.1.2.15	CPLGenerateTempFilename	360
14.1.2.16	CPLGetBasename	360
14.1.2.17	CPLGetConfigOption	361
14.1.2.18	CPLGetCurrentDir	361
14.1.2.19	CPLGetDirname	361
14.1.2.20	CPLGetExecPath	362
14.1.2.21	CPLGetExtension	362
14.1.2.22	CPLGetFilename	363

14.1.2.23 CPLGetPath	364
14.1.2.24 CPLGetSharedList	364
14.1.2.25 CPLGetSymbol	364
14.1.2.26 CPLIsFilenameRelative	365
14.1.2.27 CPLMalloc	366
14.1.2.28 CPLOpenShared	366
14.1.2.29 CPLPackedDMSToDec	367
14.1.2.30 CPLPrintDouble	367
14.1.2.31 CPLPrintInt32	368
14.1.2.32 CPLPrintPointer	368
14.1.2.33 CPLPrintString	368
14.1.2.34 CPLPrintStringFill	369
14.1.2.35 CPLPrintTime	369
14.1.2.36 CPLPrintUIntBig	370
14.1.2.37 CPLProjectRelativeFilename	370
14.1.2.38 CPLReadLine	371
14.1.2.39 CPLReadLine2L	372
14.1.2.40 CPLReadLineL	372
14.1.2.41 CPLRealloc	372
14.1.2.42 CPLResetExtension	373
14.1.2.43 CPLScanDouble	373
14.1.2.44 CPLScanLong	374
14.1.2.45 CPLScanPointer	374
14.1.2.46 CPLScanString	374
14.1.2.47 CPLScanUIntBig	375
14.1.2.48 CPLScanULong	375
14.1.2.49 CPLSetConfigOption	376
14.1.2.50 CPLSetThreadLocalConfigOption	376
14.1.2.51 CPLStrdup	376
14.1.2.52 CPLStrlwr	377
14.1.2.53 CPLStrtod	377
14.1.2.54 CPLStrtodDelim	378
14.1.2.55 CPLStrtof	378
14.1.2.56 CPLStrtofDelim	379

14.1.2.57	CPLUnlinkTree	379
14.2	cpl_error.h File Reference	379
14.2.1	Detailed Description	380
14.2.2	Function Documentation	380
14.2.2.1	_CPLAssert	380
14.2.2.2	CPLDebug	380
14.2.2.3	CPLError	380
14.2.2.4	CPLErrorReset	381
14.2.2.5	CPLGetLastErrorMsg	381
14.2.2.6	CPLGetLastErrorNo	381
14.2.2.7	CPLGetLastErrorType	382
14.2.2.8	CPLPopErrorHandler	382
14.2.2.9	CPLPushErrorHandler	382
14.2.2.10	CPLSetErrorHandler	382
14.3	cpl_hash_set.h File Reference	383
14.3.1	Detailed Description	384
14.3.2	Function Documentation	384
14.3.2.1	CPLHashSetDestroy	384
14.3.2.2	CPLHashSetEqualPointer	384
14.3.2.3	CPLHashSetEqualStr	384
14.3.2.4	CPLHashSetForeach	385
14.3.2.5	CPLHashSetHashPointer	385
14.3.2.6	CPLHashSetHashStr	385
14.3.2.7	CPLHashSetInsert	386
14.3.2.8	CPLHashSetLookup	386
14.3.2.9	CPLHashSetNew	386
14.3.2.10	CPLHashSetRemove	387
14.3.2.11	CPLHashSetSize	387
14.4	cpl_http.h File Reference	387
14.4.1	Detailed Description	388
14.4.2	Function Documentation	388
14.4.2.1	CPLHTTPDestroyResult	388
14.4.2.2	CPLHTTPEnabled	388
14.4.2.3	CPLHTTPFetch	388

14.4.2.4	CPLHTTPParseMultipartMime	389
14.5	cpl_list.h File Reference	389
14.5.1	Detailed Description	390
14.5.2	Typedef Documentation	390
14.5.2.1	CPLList	390
14.5.3	Function Documentation	390
14.5.3.1	CPLListAppend	390
14.5.3.2	CPLListCount	391
14.5.3.3	CPLListDestroy	391
14.5.3.4	CPLListGet	391
14.5.3.5	CPLListGetData	391
14.5.3.6	CPLListGetLast	392
14.5.3.7	CPLListGetNext	392
14.5.3.8	CPLListInsert	392
14.5.3.9	CPLListRemove	393
14.6	cpl_minixml.h File Reference	393
14.6.1	Detailed Description	394
14.6.2	Typedef Documentation	395
14.6.2.1	CPLXMLNode	395
14.6.3	Enumeration Type Documentation	395
14.6.3.1	CPLXMLNodeType	395
14.6.4	Function Documentation	395
14.6.4.1	CPLAddXMLChild	395
14.6.4.2	CPLAddXMLSibling	396
14.6.4.3	CPLCleanXMLElementName	396
14.6.4.4	CPLCloneXMLTree	396
14.6.4.5	CPLCreateXMLElementAndValue	397
14.6.4.6	CPLCreateXMLNode	397
14.6.4.7	CPLDestroyXMLNode	398
14.6.4.8	CPLGetXMLNode	398
14.6.4.9	CPLGetXMLValue	399
14.6.4.10	CPLParseXMLFile	399
14.6.4.11	CPLParseXMLString	400
14.6.4.12	CPLRemoveXMLChild	400

14.6.4.13 CPLSearchXMLNode	401
14.6.4.14 CPLSerializeXMLTree	401
14.6.4.15 CPLSerializeXMLTreeToFile	402
14.6.4.16 CPLSetXMLValue	402
14.6.4.17 CPLStripXMLNamespace	403
14.7 cpl_odbc.h File Reference	403
14.7.1 Detailed Description	403
14.8 cpl_port.h File Reference	403
14.8.1 Detailed Description	404
14.8.2 Define Documentation	404
14.8.2.1 CPL_LSBINT16PTR	404
14.8.2.2 CPL_LSBINT32PTR	404
14.9 cpl_quad_tree.h File Reference	404
14.9.1 Detailed Description	405
14.9.2 Function Documentation	405
14.9.2.1 CPLQuadTreeCreate	405
14.9.2.2 CPLQuadTreeDestroy	405
14.9.2.3 CPLQuadTreeForeach	405
14.9.2.4 CPLQuadTreeGetAdvisedMaxDepth	406
14.9.2.5 CPLQuadTreeInsert	406
14.9.2.6 CPLQuadTreeSearch	406
14.9.2.7 CPLQuadTreeSetBucketCapacity	407
14.9.2.8 CPLQuadTreeSetMaxDepth	407
14.10 cpl_string.h File Reference	407
14.10.1 Detailed Description	408
14.10.2 Function Documentation	408
14.10.2.1 CPLBinaryToHex	408
14.10.2.2 CPLEscapeString	409
14.10.2.3 CPLForceToASCII	410
14.10.2.4 CPLGetValueType	410
14.10.2.5 CPLHexToBinary	410
14.10.2.6 CPLIsUTF8	411
14.10.2.7 CPLParseNameValue	411
14.10.2.8 CPLRecode	412

14.10.2.9 CPLRecodeFromWChar	412
14.10.2.10 CPLRecodeToWChar	413
14.10.2.11 CPLStrcat	414
14.10.2.12 CPLStrncpy	414
14.10.2.13 CPLStrnlen	415
14.10.2.14 CPLUnescapeString	415
14.10.2.15 CSLCount	416
14.10.2.16 CSLDestroy	416
14.10.2.17 CSLDuplicate	416
14.10.2.18 CSLFindName	417
14.10.2.19 CSLFindString	417
14.10.2.20 CSLLoad	417
14.10.2.21 CSLLoad2	418
14.10.2.22 CSLMerge	418
14.10.2.23 CSLPartialFindString	419
14.10.2.24 CSLSetNameValue	419
14.10.2.25 CSLSetNameValueSeparator	420
14.10.2.26 CSLTestBoolean	420
14.10.2.27 CSLTokenizeString2	420
14.11 cpl_vsi.h File Reference	421
14.11.1 Detailed Description	423
14.11.2 Function Documentation	423
14.11.2.1 VSIFCloseL	423
14.11.2.2 VSIFEofL	423
14.11.2.3 VSIFFlushL	424
14.11.2.4 VSIFFileFromMemBuffer	424
14.11.2.5 VSIFOpenL	425
14.11.2.6 VSIFPrintfL	425
14.11.2.7 VSIFReadL	426
14.11.2.8 VSIFSeekL	426
14.11.2.9 VSIFTellL	427
14.11.2.10 VSIFWriteL	427
14.11.2.11 VSIGetMemFileBuffer	428
14.11.2.12 VSIIInstallGZipFileHandler	428

14.11.2.13	VSInstallMemFileHandler	429
14.11.2.14	VSInstallSubFileHandler	430
14.11.2.15	VSInstallZipFileHandler	430
14.11.2.16	VSIMalloc2	430
14.11.2.17	VSIMalloc3	431
14.11.2.18	VSIMkdir	431
14.11.2.19	VSIRmdir	431
14.11.2.20	VSIRename	432
14.11.2.21	VSIRmdir	432
14.11.2.22	VSISatL	433
14.11.2.23	VSILink	433
14.12	ogr_api.h File Reference	434
14.12.1	Detailed Description	445
14.12.2	Function Documentation	445
14.12.2.1	OGR_Dr_CopyDataSource	445
14.12.2.2	OGR_Dr_CreateDataSource	445
14.12.2.3	OGR_Dr_DeleteDataSource	446
14.12.2.4	OGR_Dr_GetName	447
14.12.2.5	OGR_Dr_Open	447
14.12.2.6	OGR_Dr_TestCapability	447
14.12.2.7	OGR_DS_CopyLayer	448
14.12.2.8	OGR_DS_CreateLayer	449
14.12.2.9	OGR_DS_DeleteLayer	450
14.12.2.10	OGR_DS_Destroy	450
14.12.2.11	OGR_DS_ExecuteSQL	450
14.12.2.12	OGR_DS_GetDriver	451
14.12.2.13	OGR_DS_GetLayer	451
14.12.2.14	OGR_DS_GetLayerByName	452
14.12.2.15	OGR_DS_GetLayerCount	452
14.12.2.16	OGR_DS_GetName	453
14.12.2.17	OGR_DS_ReleaseResultSet	453
14.12.2.18	OGR_DS_SyncToDisk	454
14.12.2.19	OGR_DS_TestCapability	454
14.12.2.20	OGR_F_Clone	455

14.12.2.21OGR_F_Create	455
14.12.2.22OGR_F_Destroy	456
14.12.2.23OGR_F_DumpReadable	456
14.12.2.24OGR_F_Equal	456
14.12.2.25OGR_F_GetDefnRef	457
14.12.2.26OGR_F_GetFID	457
14.12.2.27OGR_F_GetFieldAsBinary	457
14.12.2.28OGR_F_GetFieldAsDateTime	458
14.12.2.29OGR_F_GetFieldAsDouble	458
14.12.2.30OGR_F_GetFieldAsDoubleList	459
14.12.2.31OGR_F_GetFieldAsInteger	459
14.12.2.32OGR_F_GetFieldAsIntegerList	460
14.12.2.33OGR_F_GetFieldAsString	460
14.12.2.34OGR_F_GetFieldAsStringList	461
14.12.2.35OGR_F_GetFieldCount	461
14.12.2.36OGR_F_GetFieldDefnRef	462
14.12.2.37OGR_F_GetFieldIndex	462
14.12.2.38OGR_F_GetGeometryRef	463
14.12.2.39OGR_F_GetRawFieldRef	463
14.12.2.40OGR_F_GetStyleString	463
14.12.2.41OGR_F_IsFieldSet	464
14.12.2.42OGR_F_SetFID	464
14.12.2.43OGR_F_SetFieldBinary	465
14.12.2.44OGR_F_SetFieldDateTime	465
14.12.2.45OGR_F_SetFieldDouble	465
14.12.2.46OGR_F_SetFieldDoubleList	466
14.12.2.47OGR_F_SetFieldInteger	466
14.12.2.48OGR_F_SetFieldIntegerList	467
14.12.2.49OGR_F_SetFieldRaw	467
14.12.2.50OGR_F_SetFieldString	467
14.12.2.51OGR_F_SetFieldStringList	468
14.12.2.52OGR_F_SetFrom	468
14.12.2.53OGR_F_SetFromWithMap	469
14.12.2.54OGR_F_SetGeometry	469

14.12.2.55OGR_F_SetGeometryDirectly	470
14.12.2.56OGR_F_SetStyleString	470
14.12.2.57OGR_F_SetStyleStringDirectly	471
14.12.2.58OGR_F_UnsetField	471
14.12.2.59OGR_FD_AddFieldDefn	471
14.12.2.60OGR_FD_Create	472
14.12.2.61OGR_FD_Dereference	472
14.12.2.62OGR_FD_Destroy	473
14.12.2.63OGR_FD_GetFieldCount	473
14.12.2.64OGR_FD_GetFieldDefn	473
14.12.2.65OGR_FD_GetFieldIndex	474
14.12.2.66OGR_FD_GetGeomType	474
14.12.2.67OGR_FD_GetName	474
14.12.2.68OGR_FD_GetReferenceCount	475
14.12.2.69OGR_FD_Reference	475
14.12.2.70OGR_FD_Release	476
14.12.2.71OGR_FD_SetGeomType	476
14.12.2.72OGR_Fld_Create	476
14.12.2.73OGR_Fld_Destroy	477
14.12.2.74OGR_Fld_GetJustify	477
14.12.2.75OGR_Fld_GetNameRef	477
14.12.2.76OGR_Fld_GetPrecision	478
14.12.2.77OGR_Fld_GetType	478
14.12.2.78OGR_Fld_GetWidth	478
14.12.2.79OGR_Fld_Set	479
14.12.2.80OGR_Fld_SetJustify	479
14.12.2.81OGR_Fld_SetName	480
14.12.2.82OGR_Fld_SetPrecision	480
14.12.2.83OGR_Fld_SetType	480
14.12.2.84OGR_Fld_SetWidth	480
14.12.2.85OGR_G_AddGeometry	481
14.12.2.86OGR_G_AddGeometryDirectly	481
14.12.2.87OGR_G_AddPoint	482
14.12.2.88OGR_G_AddPoint_2D	482

14.12.2.89	OGR_G_AssignSpatialReference	483
14.12.2.90	OGR_G_Buffer	483
14.12.2.91	OGR_G_Clone	484
14.12.2.92	OGR_G_Contains	484
14.12.2.93	OGR_G_ConvexHull	485
14.12.2.94	OGR_G_CreateFromWkb	485
14.12.2.95	OGR_G_CreateFromWkt	486
14.12.2.96	OGR_G_CreateGeometry	486
14.12.2.97	OGR_G_Crosses	487
14.12.2.98	OGR_G_DestroyGeometry	487
14.12.2.99	OGR_G_Difference	488
14.12.2.100	OGR_G_Disjoint	488
14.12.2.101	OGR_G_Distance	489
14.12.2.102	OGR_G_DumpReadable	489
14.12.2.103	OGR_G_Empty	490
14.12.2.104	OGR_G_Equals	490
14.12.2.105	OGR_G_ExportToWkb	490
14.12.2.106	OGR_G_ExportToWkt	491
14.12.2.107	OGR_G_FlattenTo2D	491
14.12.2.108	OGR_G_GetArea	491
14.12.2.109	OGR_G_GetBoundary	492
14.12.2.110	OGR_G_GetCoordinateDimension	492
14.12.2.111	OGR_G_GetDimension	493
14.12.2.112	OGR_G_GetEnvelope	493
14.12.2.113	OGR_G_GetGeometryCount	494
14.12.2.114	OGR_G_GetGeometryName	494
14.12.2.115	OGR_G_GetGeometryRef	494
14.12.2.116	OGR_G_GetGeometryType	495
14.12.2.117	OGR_G_GetPoint	495
14.12.2.118	OGR_G_GetPointCount	496
14.12.2.119	OGR_G_GetSpatialReference	496
14.12.2.120	OGR_G_GetX	496
14.12.2.121	OGR_G_GetY	497
14.12.2.122	OGR_G_GetZ	497

14.12.2.123	OGR_G_ImportFromWkb	497
14.12.2.124	OGR_G_ImportFromWkt	498
14.12.2.125	OGR_G_Intersection	498
14.12.2.126	OGR_G_Intersects	499
14.12.2.127	OGR_G_IsEmpty	499
14.12.2.128	OGR_G_IsRing	500
14.12.2.129	OGR_G_IsSimple	500
14.12.2.130	OGR_G_IsValid	501
14.12.2.131	OGR_G_Overlaps	501
14.12.2.132	OGR_G_RemoveGeometry	502
14.12.2.133	OGR_G_Segmentize	502
14.12.2.134	OGR_G_SetPoint	503
14.12.2.135	OGR_G_SetPoint_2D	503
14.12.2.136	OGR_G_SymmetricDifference	503
14.12.2.137	OGR_G_Touches	504
14.12.2.138	OGR_G_Transform	504
14.12.2.139	OGR_G_TransformTo	505
14.12.2.140	OGR_G_Union	506
14.12.2.141	OGR_G_Within	506
14.12.2.142	OGR_G_WkbSize	507
14.12.2.143	OGR_GetFieldTypeName	507
14.12.2.144	OGR_L_CommitTransaction	507
14.12.2.145	OGR_L_CreateFeature	508
14.12.2.146	OGR_L_CreateField	508
14.12.2.147	OGR_L_DeleteFeature	509
14.12.2.148	OGR_L_GetExtent	509
14.12.2.149	OGR_L_GetFeature	510
14.12.2.150	OGR_L_GetFeatureCount	511
14.12.2.151	OGR_L_GetFIDColumn	511
14.12.2.152	OGR_L_GetGeometryColumn	511
14.12.2.153	OGR_L_GetLayerDefn	512
14.12.2.154	OGR_L_GetNextFeature	512
14.12.2.155	OGR_L_GetSpatialFilter	513
14.12.2.156	OGR_L_GetSpatialRef	513

14.12.2.157	OGR_L_ResetReading	514
14.12.2.158	OGR_L_RollbackTransaction	514
14.12.2.159	OGR_L_SetAttributeFilter	514
14.12.2.160	OGR_L_SetFeature	515
14.12.2.161	OGR_L_SetNextByIndex	516
14.12.2.162	OGR_L_SetSpatialFilter	516
14.12.2.163	OGR_L_SetSpatialFilterRect	517
14.12.2.164	OGR_L_StartTransaction	517
14.12.2.165	OGR_L_SyncToDisk	518
14.12.2.166	OGR_L_TestCapability	518
14.12.2.167	OGR_SM_AddPart	520
14.12.2.168	OGR_SM_AddStyle	520
14.12.2.169	OGR_SM_Create	521
14.12.2.170	OGR_SM_Destroy	521
14.12.2.171	OGR_SM_GetPart	521
14.12.2.172	OGR_SM_GetPartCount	522
14.12.2.173	OGR_SM_InitFromFeature	522
14.12.2.174	OGR_SM_InitStyleString	522
14.12.2.175	OGR_ST_Create	523
14.12.2.176	OGR_ST_Destroy	523
14.12.2.177	OGR_ST_GetParamDbl	524
14.12.2.178	OGR_ST_GetParamNum	524
14.12.2.179	OGR_ST_GetParamStr	524
14.12.2.180	OGR_ST_GetRGBFromString	525
14.12.2.181	OGR_ST_GetStyleString	525
14.12.2.182	OGR_ST_GetType	526
14.12.2.183	OGR_ST_GetUnit	526
14.12.2.184	OGR_ST_SetParamDbl	526
14.12.2.185	OGR_ST_SetParamNum	527
14.12.2.186	OGR_ST_SetParamStr	527
14.12.2.187	OGR_ST_SetUnit	528
14.12.2.188	OGR_STBL_Create	528
14.12.2.189	OGR_STBL_Destroy	528
14.12.2.190	OGR_STBL_Find	528

14.12.2.190	OGR_STBL_GetLastStyleName	529
14.12.2.192	OGR_STBL_GetNextStyle	529
14.12.2.193	OGR_STBL_LoadStyleTable	530
14.12.2.194	OGR_STBL_ResetStyleStringReading	530
14.12.2.195	OGR_STBL_SaveStyleTable	530
14.12.2.196	OGRBuildPolygonFromEdges	531
14.12.2.197	OGRCleanupAll	531
14.12.2.198	OGRGetDriver	532
14.12.2.199	OGRGetDriverByName	532
14.12.2.200	OGRGetDriverCount	533
14.12.2.200	OGRGetOpenDS	533
14.12.2.202	OGRGetOpenDSCount	533
14.12.2.203	OGROpen	533
14.12.2.204	OGRRegisterDriver	534
14.12.2.205	OGRReleaseDataSource	535
14.12.2.206	OGRSetGenerate_DB2_V72_BYTE_ORDER	535
14.13	ogr_core.h File Reference	535
14.13.1	Detailed Description	537
14.13.2	Define Documentation	537
14.13.2.1	GDAL_CHECK_VERSION	537
14.13.3	Typedef Documentation	537
14.13.3.1	OGRSTBrushParam	537
14.13.3.2	OGRSTClassId	537
14.13.3.3	OGRSTLabelParam	537
14.13.3.4	OGRSTPenParam	537
14.13.3.5	OGRSTSymbolParam	537
14.13.3.6	OGRSTUnitId	537
14.13.4	Enumeration Type Documentation	537
14.13.4.1	ogr_style_tool_class_id	537
14.13.4.2	ogr_style_tool_param_brush_id	538
14.13.4.3	ogr_style_tool_param_label_id	538
14.13.4.4	ogr_style_tool_param_pen_id	538
14.13.4.5	ogr_style_tool_param_symbol_id	538
14.13.4.6	ogr_style_tool_units_id	538

14.13.4.7 OGRFieldType	538
14.13.4.8 OGRJustification	538
14.13.4.9 OGRwkbGeometryType	539
14.13.5 Function Documentation	539
14.13.5.1 GDALCheckVersion	539
14.13.5.2 OGRGeometryTypeToName	540
14.13.5.3 OGRMergeGeometryTypes	540
14.13.5.4 OGRParseDate	541
14.14ogr_feature.h File Reference	541
14.14.1 Detailed Description	541
14.15ogr_featurestyle.h File Reference	542
14.15.1 Detailed Description	542
14.16ogr_geometry.h File Reference	542
14.16.1 Detailed Description	543
14.17ogr_spatialref.h File Reference	543
14.17.1 Detailed Description	543
14.17.2 Function Documentation	543
14.17.2.1 OGRCreateCoordinateTransformation	543
14.18ogr_srs_api.h File Reference	544
14.18.1 Detailed Description	550
14.18.2 Function Documentation	550
14.18.2.1 OCTDestroyCoordinateTransformation	550
14.18.2.2 OPTGetParameterInfo	551
14.18.2.3 OPTGetParameterList	551
14.18.2.4 OPTGetProjectionMethods	552
14.18.2.5 OSRAxisEnumToName	552
14.18.2.6 OSRCleanup	552
14.18.2.7 OSRClone	552
14.18.2.8 OSRCloneGeogCS	552
14.18.2.9 OSRCopyGeogCSFrom	553
14.18.2.10OSRDereference	553
14.18.2.11OSRDestroySpatialReference	553
14.18.2.12OSRExportToMICoordSys	553
14.18.2.13OSRExportToPCI	553

14.18.2.14OSRExportToPrettyWkt	554
14.18.2.15OSRExportToProj4	554
14.18.2.16OSRExportToUSGS	554
14.18.2.17OSRExportToWkt	554
14.18.2.18OSRExportToXML	555
14.18.2.19OSRFixup	555
14.18.2.20OSRFixupOrdering	555
14.18.2.21OSRGetAngularUnits	555
14.18.2.22OSRGetAttrValue	555
14.18.2.23OSRGetAuthorityCode	556
14.18.2.24OSRGetAuthorityName	556
14.18.2.25OSRGetAxis	556
14.18.2.26OSRGetInvFlattening	556
14.18.2.27OSRGetLinearUnits	556
14.18.2.28OSRGetNormProjParm	557
14.18.2.29OSRGetPrimeMeridian	557
14.18.2.30OSRGetProjParm	557
14.18.2.31OSRGetSemiMajor	557
14.18.2.32OSRGetSemiMinor	557
14.18.2.33OSRGetTOWGS84	558
14.18.2.34OSRGetUTMZone	558
14.18.2.35OSRImportFromEPSG	558
14.18.2.36OSRImportFromEPSGA	558
14.18.2.37OSRImportFromESRI	558
14.18.2.38OSRImportFromMlCoordSys	558
14.18.2.39OSRImportFromPCI	559
14.18.2.40OSRImportFromProj4	559
14.18.2.41OSRImportFromUrl	559
14.18.2.42OSRImportFromUSGS	559
14.18.2.43OSRImportFromWkt	559
14.18.2.44OSRImportFromXML	560
14.18.2.45OSRIsGeographic	560
14.18.2.46OSRIsLocal	560
14.18.2.47OSRIsProjected	560

14.18.2.48OSRIsSame	560
14.18.2.49OSRIsSameGeogCS	561
14.18.2.50OSRMorphFromESRI	561
14.18.2.51OSRMorphToESRI	561
14.18.2.52OSRNewSpatialReference	561
14.18.2.53OSRReference	561
14.18.2.54OSRRelease	562
14.18.2.55OSRSetACEA	562
14.18.2.56OSRSetAE	562
14.18.2.57OSRSetAngularUnits	562
14.18.2.58OSRSetAttrValue	562
14.18.2.59OSRSetAuthority	563
14.18.2.60OSRSetBonne	563
14.18.2.61OSRSetCEA	563
14.18.2.62OSRSetCS	563
14.18.2.63OSRSetEC	563
14.18.2.64OSRSetEckert	563
14.18.2.65OSRSetEckertIV	564
14.18.2.66OSRSetEckertVI	564
14.18.2.67OSRSetEquirectangular	564
14.18.2.68OSRSetEquirectangular2	564
14.18.2.69OSRSetFromUserInput	564
14.18.2.70OSRSetGaussSchreiberTMercator	565
14.18.2.71OSRSetGeogCS	565
14.18.2.72OSRSetGEOS	565
14.18.2.73OSRSetGH	565
14.18.2.74OSRSetGnomonic	565
14.18.2.75OSRSetGS	566
14.18.2.76OSRSetHOM	566
14.18.2.77OSRSetHOM2PNO	566
14.18.2.78OSRSetIWMPolyconic	566
14.18.2.79OSRSetKrovak	566
14.18.2.80OSRSetLAEA	567
14.18.2.81OSRSetLCC	567

14.18.2.82	OSRSetLCC1SP	567
14.18.2.83	OSRSetLCCB	567
14.18.2.84	OSRSetLinearUnits	567
14.18.2.85	OSRSetLinearUnitsAndUpdateParameters	568
14.18.2.86	OSRSetLocalCS	568
14.18.2.87	OSRSetMC	568
14.18.2.88	OSRSetMercator	568
14.18.2.89	OSRSetMollweide	568
14.18.2.90	OSRSetNormProjParm	569
14.18.2.91	OSRSetNZMG	569
14.18.2.92	OSRSetOrthographic	569
14.18.2.93	OSRSetOS	569
14.18.2.94	OSRSetPolyconic	569
14.18.2.95	OSRSetProjCS	569
14.18.2.96	OSRSetProjection	570
14.18.2.97	OSRSetProjParm	570
14.18.2.98	OSRSetPS	570
14.18.2.99	OSRSetRobinson	570
14.18.2.100	OSRSetSinusoidal	570
14.18.2.101	OSRSetSOC	571
14.18.2.102	OSRSetStereographic	571
14.18.2.103	OSRSetTM	571
14.18.2.104	OSRSetTMG	571
14.18.2.105	OSRSetTMSO	571
14.18.2.106	OSRSetTMVariant	571
14.18.2.107	OSRSetTOWGS84	572
14.18.2.108	OSRSetUTM	572
14.18.2.109	OSRSetVDG	572
14.18.2.110	OSRSetWagner	572
14.18.2.111	OSRSetWellKnownGeogCS	572
14.18.2.112	OSRStripCTParms	573
14.18.2.113	OSRValidate	573
14.19	ogrsf_frmts.h File Reference	573
14.19.1	Detailed Description	573

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 `FWTtools` 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.443) which registers all format drivers built into GDAL/OGR.

In C++ :

```
#include "ogr_sfrmts.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, RDBM-Ses, 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. 266) 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. 105) can potentially have many layers associated with it. The number of layers available can be queried with **OGRDataSource::GetLayerCount()** (p. 111) and individual layers fetched by index using **OGRDataSource::GetLayer()** (p. 110). However, we will 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. 200) to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using **OGRLayer::GetNextFeature()** (p. 198). 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. 134). 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. 123) method. In fact we could shorten the above by using **OGRFeature::GetFieldAsString()** (p. 123) 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. 151) 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,%.3f\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,%.3f\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. 125) returns a pointer to the internal geometry owned by the **OGRFeature** (p. 115). There we don't actually delete the return geometry. However, the **OGRLayer::GetNextFeature()** (p. 198) 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. 192) returned by **OGRDataSource::GetLayerByName()** (p. 110) is also a reference to an internal layer owned by the **OGRDataSource** (p. 105) so we don't need to delete it. But we do need to delete the datasource in order to close the input file. Once again we do this with a custom delete method to avoid special win32 heap issues.

In C++ :

```

    OGRDataSource::DestroyDataSource( poDS );
}

```

In C :

```

    OGR_DS_Destroy( hDS );
}

```

All together our program looks like this.

In C++ :

```

#include "ogr_sfrmts.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,%.3f\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()

feat = lyr.GetNextFeature()
while feat is not None:

    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"

        feat = lyr.GetNextFeature()

ds.Destroy()

```

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 separated values from input text will be written to a point shapefile via OGR.

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

In C++ :

```

#include "ogr_sfrmts.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. 139) 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. 139) 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 snippets 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. 115), 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. 134) 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.132) differs from **OGRFeature::SetGeometry()** (p.132) 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.194) 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 "ogr_sfrmts.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.Destroy()
```


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. 151), 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. 268) encapsulates the definition of a projection and datum.
- **Feature** (`ogr_feature.h`): The **OGRFeature** (p. 115) 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. 134) class captures the schema (set of field definitions) for a group of related features (normally a whole layer).
- **Layer** (`ogr_sf_frmts.h`): **OGRLayer** (p. 192) is an abstract base class represent a layer of features in an **OGRDataSource** (p. 105).
- **Data Source** (`ogr_sf_frmts.h`): An **OGRDataSource** (p. 105) is an abstract base class representing a file or database containing one or more **OGRLayer** (p. 192) objects.

- **Drivers** (`ogr_srf_frmts.h`): An **OGRSFDriver** (p. 259) represents a translator for a specific format, opening **OGRDataSource** (p. 105) objects. All available drivers are managed by the **OGRSFDriverRegistrar** (p. 263).

3.2 Geometry

The geometry classes represent various kinds of vector geometry. All the geometry classes derived from **OGRGeometry** (p. 151) which defines the common services of all geometries. Types of geometry include **OGRPoint** (p. 237), **OGRLineString** (p. 210), **OGRPolygon** (p. 246), **OGRGeometryCollection** (p. 172), **OGRMultiPolygon** (p. 233), **OGRMultiPoint** (p. 229), and **OGRMultiLineString** (p. 226).

Additional intermediate abstract base classes contain functionality that could eventually be implemented by other geometry types. These include **OGRCurve** (p. 102) (base class for **OGRLineString** (p. 210)) and **OGRSurface** (p. 339) (base class for **OGRPolygon** (p. 246)). 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. 184) 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. 151) includes a reference to an **OGRSpatialReference** (p. 268) 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. 151) 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. 151).

While it is theoretically possible to derive other or more specific geometry classes from the existing **OGRGeometry** (p. 151) 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. 184) without modifying it.

3.3 Spatial Reference

The **OGRSpatialReference** (p. 268) 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. 268) 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. 99) 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. 268) class.

3.4 Feature / Feature Definition

The **OGRGeometry** (p. 151) captures the geometry of a vector feature ... the spatial position/region of a feature. The **OGRFeature** (p. 115) contains this geometry, and adds feature attributes, feature id, and a feature class identifier.

The set of attributes, their types, names and so forth is represented via the **OGRFeatureDefn** (p. 134) class. One **OGRFeatureDefn** (p. 134) 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::GetGeometryType()** (p. 137)). 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. 134) 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. 192) represents a layer of features within a data source. All features in an **OGRLayer** (p. 192) share a common schema and are of the same **OGRFeatureDefn** (p. 134). An **OGRLayer** (p. 192) class also contains methods for reading features from the data source. The **OGRLayer** (p. 192) 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. 192) represents a spatial table.

The **OGRLayer** (p. 192) includes methods for sequential and random reading and writing. Read access (via the **OGRLayer::GetNextFeature()** (p. 198) 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. 192) (via the **OGRLayer::SetSpatialFilter()** (p. 202) method).

One flaw in the current OGR architecture is that the spatial filter is set directly on the

OGRLayer (p. 192) 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 introduce an **OGRLayerView** class or something similar.

Another question that might arise is why the **OGRLayer** (p. 192) and **OGRFeatureDefn** (p. 134) classes are distinct. An **OGRLayer** (p. 192) always has a one-to-one relationship to an **OGRFeatureDefn** (p. 134), so why not amalgamate the classes. - There are two reasons:

1. As defined now **OGRFeature** (p. 115) and **OGRFeatureDefn** (p. 134) don't depend on **OGRLayer** (p. 192), 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. 192) 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. 105), and aren't instantiated or destroyed directly.

3.6 Data Source

An **OGRDataSource** (p. 105) represents a set of **OGRLayer** (p. 192) objects. This usually represents a single file, set of files, database or gateway. An **OGRDataSource** (p. 105) has a list of **OGRLayer**'s which it owns but can return references to.

OGRDataSource (p. 105) is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. **OGRDataSource** (p. 105) objects are not normally instantiated directly but rather with the assistance of an **OGRSFDriver** (p. 259). Deleting an **OGRDataSource** (p. 105) closes access to the underlying persistent data source, but does not normally result in deletion of that file.

An **OGRDataSource** (p. 105) has a name (usually a filename) that can be used to re-open the data source with an **OGRSFDriver** (p. 259).

The **OGRDataSource** (p. 105) also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the **OGRDataSource::ExecuteSQL()** (p. 109) 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. 259) object is instantiated for each file format supported. The **OGRSFDriver** (p. 259) objects are registered with the **OGRSFDriverRegistrar** (p. 263), a singleton class that is normally used to open new data sources.

It is intended that a new **OGRSFDriver** (p. 259) derived class be implemented for each file format to be supported (along with a file format specific **OGRDataSource** (p. 105), and **OGRLayer** (p. 192) classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate **OGRSFDriver** (p. 259) objects, and register them with the **OGRSFDriverRegistrar** (p. 263). When a data source is to be opened, the registrar will normally try each **OGRSFDriver** (p. 259) in turn, until one succeeds, returning an **OGRDataSource** (p. 105) object.

It is not intended that the **OGRSFDriverRegistrar** (p. 263) 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. 259), **OGRDataSource** (p. 105) and **OGRLayer** (p. 192). The **OGRSFDriver** (p. 259) subclass is registered with the **OGRSFDriverRegistrar** (p. 263) 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. 27)
2. **Basic Read Only Data Source** (p. 29)
3. **Read Only Layer** (p. 31)

4.3 Implementing OGRSFDriver

The format specific driver class is implemented as a subclass of **OGRSFDriver** (p. 259). 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.261) 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.266), or from the C API **OGROpen()** (p.533). The **OGRSFDriver::Open()** (p.262) 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.105) 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.105) 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. 535)) 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. 105) 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 preferable.

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` implements layer semantics for an `.spf` file. It provides access to a set of feature objects in a consistent 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. 134) 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. 134). As **OGRFeature**'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. 134). 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. 192) 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.192) base class support maintain filters in the **OGRLayer** (p.192) 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. 105) supports executing commands against a datasource via the **OGRDataSource::ExecuteSQL()** (p. 109) 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.

5.1 Supported SQL syntax

OGR SQL supports the following pseudo-syntax:

```
SELECT <field-list> FROM <table_def>
    [LEFT JOIN <table_def>
      ON [<table_ref>.<key_field> = [<table_ref>.<key_field>]*
    [WHERE <where-expr>]
    [ORDER BY <sort specification list>]

<field-list> ::= <column-spec> [ { , <column-spec> }... ]

<column-spec> ::= <field-spec> [ <as clause> ]
                | CAST ( <field-spec> AS <data type> ) [ <as clause> ]

<field-spec> ::= [DISTINCT] <field_ref>
                | <field_func> ( [DISTINCT] <field-ref> )
                | Count(*)

<as clause> ::= [ AS ] <column_name>

<data type> ::= character [ ( field_length ) ]
              | float [ ( field_length ) ]
              | numeric [ ( field_length [, field_precision ] ) ]
              | integer [ ( field_length ) ]
              | date [ ( field_length ) ]
              | time [ ( field_length ) ]
              | timestamp [ ( field_length ) ]

<field_func> ::= AVG | MAX | MIN | SUM | COUNT
```

```

<field_ref> ::= [<table_ref>.]field_name

<sort specification list> ::=
    <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::= <sort key> [ <ordering specification> ]

<sort key> ::= <field_ref>

<ordering specification> ::= ASC | DESC

<table_def> ::= ['<datasource name>'.]table_name [table_alias]

<table_ref> ::= table_name | table_alias

```

5.2 SELECT

The SELECT statement is used to fetch layer features (analogous 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 analogous to tables in an RDBMS and feature attributes are analogous to column values. The simplest 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 equivalent 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
```

5.2.1 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 a lot 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 (numerical sum), MIN (lexical or numerical minimum), and MAX (lexical or numerical maximum). This example produces a variety of summarization 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"
```

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 though it would get the same result as giving it any of the column names. 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
```

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.

5.2.1.1 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 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:

```
SELECT COUNT(areacode) AS 'count' FROM polylayer
```

We can optionally omit the AS keyword in the field name aliases, like:

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

5.2.1.2 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. Conversion to the 'integer list', 'double list' and 'string list' OGR data types are not supported, which doesn't conform to the SQL92 specification.

5.2.1.3 Field List Limitations

1. Field arithmetic, and other binary operators are not supported, so you can't do something like:

```
SELECT prop_value / area FROM invoices
```

2. Lots of operators are missing.

5.2.2 WHERE

The argument to the WHERE clause is a fairly simplistic logical expression used select records to be selected 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.

A WHERE clause consists of a set of attribute tests. Each basic test is of the form **fieldname operator value**. The **fieldname** is any of the fields in the source layer. The operator is one of =, !=, <>, <, >, <=, >=, **LIKE** and **ILIKE** and **IN**.

Most of the operators are self explanatory, but it is worth noting 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.

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

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 precedence 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 "NON%")
SELECT * FROM poly WHERE (prop_value IS NOT NULL) AND (prop_value < 100000)
```

5.2.3 WHERE Limitations

1. The left of any comparison operator must be a field name, and the right must be a literal value. Fields cannot currently be compared to fields.
2. Fields must all come from the primary table (the one listed in the FROM clause, and must not have any table prefix ... they must just be the field name.
3. No arithmetic operations are supported. You can't test "WHERE (a+b) < 10" for instance.
4. All string comparisons are case insensitive except for <, >, <= and >=.

5.2.4 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.

5.2.5 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 until 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 tables 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
```

5.2.6 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.

5.3 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 precedence over the other fields with the same names in the data source.

5.3.1 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
```

5.3.2 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. 163) 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'
```

5.3.3 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%'
```

5.3.4 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. 340) method. For **OGRGeometryCollection** (p. 172) and **OGRMultiPolygon** (p. 233) 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'
```

5.3.5 OGR_STYLE

The **OGR_STYLE** special field represents the style string of the feature returned by - **OGRFeature::GetStyleString()** (p. 126). 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%'
```

5.4 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
```

5.4.1 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 they will accelerate is a simple "field = value" query.

5.5 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
```

5.6 ExecuteSQL()

SQL is executed against an **OGRDataSource** (p. 105), 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. 202) method. It may be NULL for no special spatial restriction.

The result of an ExecuteSQL() call is usually a temporary **OGRLayer** (p. 192) 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::ReleaseResultSet() method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

5.7 Non-OGR SQL

All OGR drivers for database systems: MySQL, PostgreSQL and PostGIS (PG), Oracle (OCI), SQLite, ODBC and ESRI Personal Geodatabase (PGeo) override the **OGRDataSource::ExecuteSQL()** (p. 109) 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.268), and **OGRCoordinateTransformation** (p.99) 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.268) class. - There are a number of ways of initializing an **OGRSpatialReference** (p.268) 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.268) 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 serialization, and transmission of projection definitions to other packages, the - OpenGIS Well Known Text format for coordinate systems is used. An **OGRSpatialReference** (p.268) 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. 304) method can be used to set an `OGRSpatialReference` (p. 268) 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.268) 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.306) and **OGRSpatialReference::IsGeographic()** (p.305) methods. The **OGRSpatialReference::GetSemiMajor()** (p.289), **OGRSpatialReference::GetSemiMinor()** (p.290) and **OGRSpatialReference::GetInvFlattening()** (p.287) methods can be used to get information about the spheroid. The **OGRSpatialReference::GetAttrValue()** (p.284) method can be used to get the PROJCS, GEOGCS, DATUM, SPHEROID, and PROJECTION names strings. The **OGRSpatialReference::GetProjParm()** (p.289) method can be used to get the projection parameters. The **OGRSpatialReference::GetLinearUnits()** (p.287) 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'd 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 `ogrsatialreference.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.99) class is used for translating positions between different coordinate systems. New transformation objects are created using **OGRCreateCoordinateTransformation()** (p.543), and then the **OGRCoordinateTransformation::Transform()** (p.101) 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. 543) 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. 543) fails it will return a NULL.

The **OGRCoordinateTransformation::Transform()** (p. 101) 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.544), 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. 99) service is implemented on top of the PROJ. 4 library originally written by Gerald Evenden of the USGS.

Chapter 7

Deprecated List

Member OGRSpatialReference::~OGRSpatialReference (p. 275) ()

Chapter 8

Directory Hierarchy

8.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

ogrsf_frmts	67
generic	67
port	67

Chapter 9

Class Index

9.1 Class Hierarchy

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

_CPLHashSet	71
_CPLList	71
_CPLQuadTree	72
_QuadTreeNode	72
_sPolyExtended	72
CPLErrorContext	72
CPLHTTPResult	72
CPLKeywordParser	74
CPLLocaleC	74
CPLMimePart	74
CPLMutexHolder	75
CPLODBCDriverInstaller	75
CPLODBCSession	77
CPLODBCStatement	78
CPLRectObj	87
CPLSharedFileInfo	87
CPLStdCallThreadInfo	87
CPLString	87
CPLXMLNode	88
ctb	90
DefaultCSVFileNameTLS	90
errHandler	91
file_in_zip_read_info_s	91
FindFileTLS	91
GZipSnapshot	91
OGR_SRSNode	91
ogr_style_param	99
ogr_style_value	99
OGRAttrIndex	99

OGRMIAttrIndex	225
OGRCoordinateTransformation	99
OGRProj4CT	257
OGRDataSource	105
OGREnvelope	115
OGRFeature	115
OGRFeatureDefn	134
OGRFeatureQuery	139
OGRField	139
OGRFieldDefn	140
OGRGeometry	151
OGRCurve	102
OGRLineString	210
OGRLinearRing	206
OGRGeometryCollection	172
OGRMultiLineString	226
OGRMultiPoint	229
OGRMultiPolygon	233
OGRPoint	237
OGRSurface	339
OGRPolygon	246
OGRGeometryFactory	184
OGRLayer	192
OGRGenSQLResultsLayer	145
OGRLayerAttrIndex	206
OGRMILayerAttrIndex	226
OGRProj4Datum	259
OGRRawPoint	259
OGRSFDriver	259
OGRSFDriverRegistrar	263
OGRSpatialReference	268
OGRStyleMgr	330
OGRStyleTable	334
OGRStyleTool	338
OGRStyleBrush	329
OGRStyleLabel	329
OGRStylePen	334
OGRStyleSymbol	334
OZIDatums	341
ParseContext	341
PCIDatums	341
projUV	341
StackContext	341
swq_col_def	341
swq_field_list	341
swq_field_op	342
swq_join_def	342
swq_order_def	342

swq_select	342
swq_summary	342
swq_table_def	342
tm_unz_s	343
unz_file_info_internal_s	343
unz_file_info_s	343
unz_file_pos_s	343
unz_global_info_s	343
unz_s	343
VSIFileManager	343
VSIFilesystemHandler	344
VSIZipFilesystemHandler	344
VSIMemFilesystemHandler	345
VSIStdoutFilesystemHandler	346
VSISubFileFilesystemHandler	346
VSIUnixStdioFilesystemHandler	347
VSIZipFilesystemHandler	348
VSIMemFile	345
VSVirtualHandle	348
VSIZipHandle	344
VSIZipWriteHandle	345
VSIMemHandle	345
VSIStdoutHandle	346
VSISubFileHandle	347
VSIUnixStdioHandle	347
ZIPContent	348
ZIPEntry	349
zlib_filefunc_def_s	349

Chapter 10

Class Index

10.1 Class List

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

_CPLHashSet	71
_CPLList	71
_CPLQuadTree	72
_QuadTreeNode	72
_sPolyExtended	72
CPLErrorContext	72
CPLHTTPResult	72
CPLKeywordParser	74
CPLLocaleC	74
CPLMimePart	74
CPLMutexHolder	75
CPLODBCDriverInstaller	75
CPLODBCSession	77
CPLODBCStatement	78
CPLRectObj	87
CPLSharedFileInfo	87
CPLStdCallThreadInfo	87
CPLString	87
CPLXMLNode	88
ctb	90
DefaultCSVFileNameTLS	90
errHandler	91
file_in_zip_read_info_s	91
FindFileTLS	91
GZipSnapshot	91
OGR_SRSNode	91
ogr_style_param	99
ogr_style_value	99
OGRAttrIndex	99

OGRCoordinateTransformation	99
OGRCurve	102
OGRDataSource	105
OGREnvelope	115
OGRFeature	115
OGRFeatureDefn	134
OGRFeatureQuery	139
OGRField	139
OGRFieldDefn	140
OGRGenSQLResultsLayer	145
OGRGeometry	151
OGRGeometryCollection	172
OGRGeometryFactory	184
OGRLayer	192
OGRLayerAttrIndex	206
OGRLinearRing	206
OGRLineString	210
OGRMIAAttrIndex	225
OGRMILayerAttrIndex	226
OGRMultiLineString	226
OGRMultiPoint	229
OGRMultiPolygon	233
OGRPoint	237
OGRPolygon	246
OGRProj4CT	257
OGRProj4Datum	259
OGRRawPoint	259
OGRSFDriver	259
OGRSFDriverRegistrar	263
OGRSpatialReference	268
OGRStyleBrush	329
OGRStyleLabel	329
OGRStyleMgr	330
OGRStylePen	334
OGRStyleSymbol	334
OGRStyleTable	334
OGRStyleTool	338
OGRSurface	339
OZIDatums	341
ParseContext	341
PCIDatums	341
projUV	341
StackContext	341
swq_col_def	341
swq_field_list	341
swq_field_op	342
swq_join_def	342
swq_order_def	342
swq_select	342
swq_summary	342

swq_table_def	342
tm_unz_s	343
unz_file_info_internal_s	343
unz_file_info_s	343
unz_file_pos_s	343
unz_global_info_s	343
unz_s	343
VSIFileManager	343
VSIFilesystemHandler	344
VSIZipFilesystemHandler	344
VSIZipHandle	344
VSIZipWriteHandle	345
VSIMemFile	345
VSIMemFilesystemHandler	345
VSIMemHandle	345
VSIStdoutFilesystemHandler	346
VSIStdoutHandle	346
VSISubFileFilesystemHandler	346
VSISubFileHandle	347
VSIUnixStdioFilesystemHandler	347
VSIUnixStdioHandle	347
VSVirtualHandle	348
VSIZipFilesystemHandler	348
ZIPContent	348
ZIPEntry	349
zlib_filefunc_def_s	349

Chapter 11

File Index

11.1 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	351
cpl_csv.h	??
cpl_error.h	379
cpl_hash_set.h	383
cpl_http.h	387
cpl_list.h	389
cpl_minixml.h	393
cpl_minizip_ioapi.h	??
cpl_minizip_unzip.h	??
cpl_multiproc.h	??
cpl_odbc.h	403
cpl_port.h	403
cpl_quad_tree.h	404
cpl_string.h	407
cpl_time.h	??
cpl_vsi.h	421
cpl_vsi_virtual.h	??
cpl_win32ce_api.h	??
cpl_wince.h	??
cplkeywordparser.h	??
ogr_api.h	434
ogr_attrib.h	??
ogr_core.h	535
ogr_expat.h	??
ogr_feature.h	541
ogr_featurestyle.h	542

ogr_gensql.h	??
ogr_geometry.h	542
ogr_geos.h	??
ogr_p.h	??
ogr_spatialref.h	543
ogr_srs_api.h	544
ogr_srs_esri_names.h	??
ogrsf_frmts.h	573
swq.h	??

Chapter 12

Directory Documentation

12.1 ogrsf_frmts/generic/ Directory Reference

Files

- file **ogr_attrind.cpp**
- file **ogr_gensql.cpp**
- file **ogr_gensql.h**
- file **ogr_miattribind.cpp**
- file **ogrdatasource.cpp**
- file **ogrlayer.cpp**
- file **ogrregisterall.cpp**
- file **ogrsfdriver.cpp**
- file **ogrsfdriverregistrar.cpp**

12.2 ogrsf_frmts/ Directory Reference

Directories

- directory **generic**

Files

- file **ogr_attrind.h**
- file **ogrsf_frmts.h**

12.3 /builddir/build/BUILD/gdal-1.7.3-fedora/port/ Directory Reference -

Files

- file `cpl_atomic_ops.cpp`
- file `cpl_atomic_ops.h`
- file `cpl_config.h`
- file `cpl_config_extras.h`
- file `cpl_conv.cpp`
- file `cpl_conv.h`
- file `cpl_csv.cpp`
- file `cpl_csv.h`
- file `cpl_error.cpp`
- file `cpl_error.h`
- file `cpl_findfile.cpp`
- file `cpl_getexecpath.cpp`
- file `cpl_hash_set.cpp`
- file `cpl_hash_set.h`
- file `cpl_http.cpp`
- file `cpl_http.h`
- file `cpl_list.cpp`
- file `cpl_list.h`
- file `cpl_minixml.cpp`
- file `cpl_minixml.h`
- file `cpl_minizip_ioapi.cpp`
- file `cpl_minizip_ioapi.h`
- file `cpl_minizip_unzip.cpp`
- file `cpl_minizip_unzip.h`
- file `cpl_multiproc.cpp`
- file `cpl_multiproc.h`
- file `cpl_odbc.cpp`
- file `cpl_odbc.h`
- file `cpl_path.cpp`
- file `cpl_port.h`
- file `cpl_quad_tree.cpp`
- file `cpl_quad_tree.h`
- file `cpl_recode_stub.cpp`
- file `cpl_string.cpp`
- file `cpl_string.h`
- file `cpl_strtod.cpp`
- file `cpl_time.cpp`
- file `cpl_time.h`
- file `cpl_vsi.h`
- file `cpl_vsi_mem.cpp`
- file `cpl_vsi_virtual.h`
- file `cpl_vsil.cpp`
- file `cpl_vsil_gzip.cpp`
- file `cpl_vsil_simple.cpp`
- file `cpl_vsil_stdout.cpp`

- file `cpl_vsil_subfile.cpp`
- file `cpl_vsil_unix_stdio_64.cpp`
- file `cpl_vsil_win32.cpp`
- file `cpl_vsisimple.cpp`
- file `cpl_win32ce_api.cpp`
- file `cpl_win32ce_api.h`
- file `cpl_wince.h`
- file `cplgetsymbol.cpp`
- file `cplkeywordparser.cpp`
- file `cplkeywordparser.h`
- file `cplstring.cpp`
- file `xmlreformat.cpp`

Chapter 13

Class Documentation

13.1 _CPLHashSet Struct Reference

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

- `cpl_hash_set.cpp`

13.2 _CPLList Struct Reference

```
#include <cpl_list.h>
```

Public Attributes

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

13.2.1 Detailed Description

List element structure.

13.2.2 Member Data Documentation

13.2.2.1 `void* _CPLList::pData`

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

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

13.2.2.2 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()`, `CPLListInsert()`, `CPLListGetLast()`, `CPLListGet()`, `CPLListCount()`, `CPLListRemove()`, `CPLListDestroy()`, and `CPLListGetNext()`.

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

- `cpl_list.h`

13.3 `_CPLQuadTree` Struct Reference

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

- `cpl_quad_tree.cpp`

13.4 `_QuadTreeNode` Struct Reference

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

- `cpl_quad_tree.cpp`

13.5 `_sPolyExtended` Struct Reference

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

- `ogrgeometryfactory.cpp`

13.6 `CPLErrorContext` Struct Reference

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

- `cpl_error.cpp`

13.7 `CPLHTTPResult` Struct Reference

```
#include <cpl_http.h>
```

Public Attributes

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

13.7.1 Detailed Description

Describe the result of a **CPLHTTPFetch()** (p. 388) call

13.7.2 Member Data Documentation

13.7.2.1 int **CPLHTTPResult::nDataLen**

Length of the pabyData buffer

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

13.7.2.2 int **CPLHTTPResult::nMimePartCount**

Number of parts in a multipart message

Referenced by **CPLHTTPParseMultipartMime()**.

13.7.2.3 int **CPLHTTPResult::nStatus**

HTTP status code : 200=success, value < 0 if request failed

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

13.7.2.4 GByte* **CPLHTTPResult::pabyData**

Buffer with downloaded data

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

13.7.2.5 **CPLMimePart*** **CPLHTTPResult::pasMimePart**

Array of parts (resolved by **CPLHTTPParseMultipartMime()** (p. 389))

Referenced by **CPLHTTPParseMultipartMime()**.

13.7.2.6 `char* CPLHTTPResult::pszContentType`

Content-Type of the response

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

13.7.2.7 `char* CPLHTTPResult::pszErrMsgBuf`

Error message from curl, or NULL

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

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

- `cpl_http.h`

13.8 CPLKeywordParser Class Reference

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

- `cplkeywordparser.h`
- `cplkeywordparser.cpp`

13.9 CPLLocaleC Class Reference

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

- `cpl_conv.h`
- `cpl_conv.cpp`

13.10 CPLMimePart Struct Reference

```
#include <cpl_http.h>
```

Public Attributes

- `char ** ppszHeaders`
- `GByte * pabyData`
- `int nDataLen`

13.10.1 Detailed Description

Describe a part of a multipart message

13.10.2 Member Data Documentation

13.10.2.1 int CPLMimePart::nDataLen

Buffer length

Referenced by CPLHTTPParseMultipartMime().

13.10.2.2 GByte* CPLMimePart::pabyData

Buffer with data of the part

Referenced by CPLHTTPParseMultipartMime().

13.10.2.3 char** CPLMimePart::papszHeaders

NULL terminated array of headers

Referenced by CPLHTTPParseMultipartMime().

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

- **cpl_http.h**

13.11 CPLMutexHolder Class Reference

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

- **cpl_multiproc.h**
- **cpl_multiproc.cpp**

13.12 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)

13.12.1 Detailed Description

A class providing functions to install or remove ODBC driver.

13.12.2 Member Function Documentation

13.12.2.1 `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

<i>pszDriver</i>	- The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).
<i>pszPathIn</i>	- Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).
<i>fRequest</i>	- 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.

13.12.2.2 `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

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

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`

13.13 CPODBCSession Class Reference

```
#include <cpl_odbc.h>
```

Public Member Functions

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

13.13.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

13.13.2 Member Function Documentation

13.13.2.1 `int CPODBCSession::EstablishSession (const char * pszDSN, const char * pszUserid, const char * pszPassword)`

Connect to database and logon.

Parameters

<i>pszDSN</i>	The name of the DSN being used to connect. This is not optional.
<i>pszUserid</i>	the userid to logon as, may be NULL if not not required, or provided by the DSN.
<i>psz-Password</i>	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. 77) to get details on failure.

References `GetLastError()`.

13.13.2.2 `const char * CPODBCSession::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 CPODBCStatement::Fetch().

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

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

13.14 CPODBCStatement 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 GetType** (int)
- static SQLSMALLINT **GetTypeMapping** (SQLSMALLINT)

13.14.1 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.

13.14.2 Member Function Documentation

13.14.2.1 void CPODBCStatement::Append (const char * *pszText*)

Append text to internal command.

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

Parameters

<i>pszText</i>	text to append.
----------------	-----------------

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

13.14.2.2 void CPODBCStatement::Append (int *nValue*)

Append to internal command.

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

Parameters

<i>nValue</i>	value to append to the command.
---------------	---------------------------------

References Append().

13.14.2.3 void CPODBCStatement::Append (double *dfValue*)

Append to internal command.

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

Parameters

<i>dfValue</i>	value to append to the command.
----------------	---------------------------------

References Append().

13.14.2.4 void CPODBCStatement::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

<i>pszText</i>	text to append.
----------------	-----------------

References Append().

13.14.2.5 int CPODBCStatement::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. 79) methods.

Parameters

<i>pszFormat</i>	printf() style format string.
------------------	-------------------------------

Returns

FALSE if formatting fails due to result being too large.

References Append().

13.14.2.6 void CPODBCStatement::Clear ()

Clear internal command text and result set definitions.

Referenced by ExecuteSQL().

13.14.2.7 void CPODBCStatement::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

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

References GetColCount(), GetColName(), GetColPrecision(), GetColSize(), GetType-Name(), GetColType(), GetColNullable(), Fetch(), and GetColData().

13.14.2.8 int CPODBCStatement::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. 79) or **Appendf()** (p. 80) calls will be used.

Parameters

<i>psz-Statement</i>	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 OGRDBCSession::GetLastError().

References Clear(), and Append().

13.14.2.9 int CPODBCStatement::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 occurred if **Fetch()** (p. 81) fails.

Parameters

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

Returns

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

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

Referenced by DumpResult().

13.14.2.10 int CPODBCStatement::GetColCount ()

Fetch the resultset column count.

Returns

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

Referenced by DumpResult().

13.14.2.11 `const char * CPODBCStatement::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

<i>iCol</i>	the zero based column to fetch.
<i>pszDefault</i>	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 GetColData(), and DumpResult().

13.14.2.12 `const char * CPODBCStatement::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

<i>pszColName</i>	the name of the column requested.
<i>pszDefault</i>	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 GetColId(), and GetColData().

13.14.2.13 int CPODBCStatement::GetColId (const char * *pszColName*)

Fetch column index.

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

Parameters

<i>pszColName</i>	the name to search for.
-------------------	-------------------------

Returns

the column index, or -1 if not found.

Referenced by GetColData().

13.14.2.14 const char * CPODBCStatement::GetColName (int *iCol*)

Fetch a column name.

Parameters

<i>iCol</i>	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().

13.14.2.15 short CPODBCStatement::GetColNullable (int *iCol*)

Fetch the column nullability.

Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

Returns

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

13.14.2.16 short CPODBCStatement::GetColPrecision (int *iCol*)

Fetch the column precision.

Parameters

<i>iCol</i>	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().

13.14.2.17 short CPODBCStatement::GetColSize (int *iCol*)

Fetch the column width.

Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

Returns

column width, zero for unknown width columns.

Referenced by DumpResult().

13.14.2.18 short CPODBCStatement::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_TIMESTAMP.

Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

Returns

type code or -1 if the column is illegal.

Referenced by DumpResult().

13.14.2.19 const char * CPODBCStatement::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

<i>iCol</i>	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.

13.14.2.20 `int CPODBCStatement::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 **CPODBCStatement** (p. 78) column definitions essentially as if a "SELECT * FROM tablename" had been done; however, no resultset will be available.

Parameters

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

Returns

TRUE on success or FALSE on failure.

13.14.2.21 `int CPODBCStatement::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

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

Returns

TRUE on success or FALSE on failure.

13.14.2.22 `int CPODBCStatement::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

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

Returns

TRUE on success or FALSE on failure.

13.14.2.23 `SQLSMALLINT CPODBCStatement::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 **CPODBCStatement::GetColType()** (p. 84)).

Parameters

<i>nTypeCode</i>	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().

13.14.2.24 `CPLString CPODBCStatement::GetTypeName (int nTypeCode)` [static]

Get name for SQL column type.

Returns a string name for the indicated type code (as returned from **CPODBCStatement::GetColType()** (p. 84)).

Parameters

<i>nTypeCode</i>	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

13.15 CPLRectObj Struct Reference

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

- **cpl_quad_tree.h**

13.16 CPLSharedFileInfo Struct Reference

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

- **cpl_conv.h**

13.17 CPLStdCallThreadInfo Struct Reference

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

- cpl_multiproc.cpp

13.18 CPLString Class Reference

Public Member Functions

- **CPLString & FormatC** (double dfValue, const char *pszFormat=0)
- **CPLString & Trim** ()

13.18.1 Member Function Documentation

13.18.1.1 **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 separator) and appended to the target **CPLString** (p. 87).

Parameters

<i>dfValue</i>	the value to format.
<i>pszFormat</i>	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. 87).

13.18.1.2 **CPLString** & **CPLString::Trim** ()

Trim white space.

Trims white space off the left 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. 87).

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

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

13.19 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.

13.19.1 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. 398). 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. 88) structures.

13.19.2 Member Data Documentation

13.19.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

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

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

13.19.2.2 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 CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLSearchXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLAddXMLChild(), CPLRemoveXMLChild(), CPLCloneXMLTree(), CPLSetXMLValue(), and CPLStripXMLNamespace().

13.19.2.3 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 CPLSerializeXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLSearchXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLAddXMLChild(), CPLRemoveXMLChild(), CPLAddXMLSibling(), CPLCloneXMLTree(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::importFromXML().

13.19.2.4 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 CPLParseXMLString(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLSearchXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLCloneXMLTree(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::importFromXML().

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

- `cpl_minixml.h`

13.20 ctb Struct Reference

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

- `cpl_csv.cpp`

13.21 DefaultCSVFileNameTLS Struct Reference

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

- `cpl_csv.cpp`

13.22 errHandler Struct Reference

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

- cpl_error.cpp

13.23 file_in_zip_read_info_s Struct Reference

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

- cpl_minizip_unzip.cpp

13.24 FindFileTLS Struct Reference

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

- cpl_findfile.cpp

13.25 GZipSnapshot Struct Reference

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

- cpl_vsif_gzip.cpp

13.26 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)

13.26.1 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. 268) object, which uses the **OGR_SRSNode** (p.91) to implement it's data structure; however, this class is user accessible for detailed access to components of an SRS definition.

13.26.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

<i>pszValueIn</i>	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().

13.26.3 Member Function Documentation

13.26.3.1 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.93) method if the original is to be preserved. New children are always added at the end of the list.

Parameters

<i>poNew</i>	the node to add as a child.
--------------	-----------------------------

References InsertChild().

Referenced by OGRSpatialReference::morphToESRI(), Clone(), importFromWkt(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetLinearUnits(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetTOWGS84(), OGRSpatialReference::SetExtension(), and OGRSpatialReference::SetAxes().

13.26.3.2 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

<i>pszNode</i>	Restrict remapping to children of this type of node (eg. "PROJECTION")
<i>papszSrcValues</i>	a NULL terminated array of source string. If the node value matches one of these (case insensitive) then replacement occurs.
<i>papszDstValues</i>	an array of destination strings. On a match, the one corresponding to a source value will be used to replace a node.
<i>nStepSize</i>	increment when stepping through source and destination arrays, allowing source and destination arrays to be one interleaved array for instances. Defaults to 1.
<i>bChildOfHit</i>	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 occurred.

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

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

13.26.3.3 OGR_SRSNode * OGR_SRSNode::Clone () const

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

Returns

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

References OGR_SRSNode(), and AddChild().

Referenced by OGRSpatialReference::Clone(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::StripVertical(), and OGRSpatialReference::CloneGeogCS().

13.26.3.4 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

<i>iChild</i>	the index of the child.
---------------	-------------------------

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

13.26.3.5 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

<i>ppszResult</i>	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().

13.26.3.6 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

<i>pszValue</i>	the node value being searched for.
-----------------	------------------------------------

Returns

the child index, or -1 on failure.

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

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

Fetch requested child.

Parameters

<i>iChild</i>	the index of the child to fetch, from 0 to GetChildCount() (p. 96) - 1.
---------------	--

Returns

a pointer to the child **OGR_SRSNode** (p. 91), or NULL if there is no such child.

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

13.26.3.8 `int OGR_SRSNode::GetChildCount () const [inline]`

Get number of children nodes.

Returns

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

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

13.26.3.9 `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. 96) on a subtree to be more specific.

Parameters

<i>pszName</i>	the name of the node to search for.
----------------	-------------------------------------

Returns

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

References `GetNode()`.

Referenced by `OGRSpatialReference::Validate()`, `GetNode()`, and `OGRSpatialReference::GetAttrNode()`.

13.26.3.10 `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::morphToESRI(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::Validate(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::SetNode(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::GetLinearUnits(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::StripCTParms(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsGeographic(), OGRSpatialReference::IsSame(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::GetExtension(), OGRSpatialReference::SetExtension(), and OGRSpatialReference::GetAxis().

13.26.3.11 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

<i>ppszInput</i>	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 SetValue(), OGR_SRSNode(), importFromWkt(), and AddChild().

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

13.26.3.12 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. 93) 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

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

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

13.26.3.13 void OGR_SRSNode::MakeValueSafe ()

Message 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 `GetChildCount()`, `GetChild()`, and `MakeValueSafe()`.

Referenced by `MakeValueSafe()`.

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

Set the node value.

Parameters

<i>pszNewValue</i>	the new value to assign to this node. The passed string is duplicated and remains the responsibility of the caller.
--------------------	---

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

13.26.3.15 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

<i>pszName</i>	the name for nodes that should be removed.
----------------	--

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

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

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

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

13.27 ogr_style_param Struct Reference

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

- **ogr_featurestyle.h**

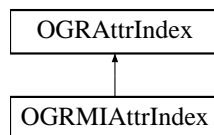
13.28 ogr_style_value Struct Reference

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

- **ogr_featurestyle.h**

13.29 OGRAttrIndex Class Reference

Inheritance diagram for OGRAttrIndex:



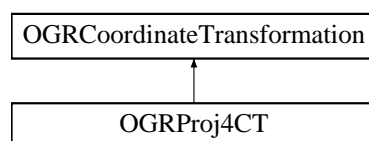
The documentation for this class was generated from the following files:

- **ogr_attrind.h**
- **ogr_attrind.cpp**

13.30 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. 99) destructor.

13.30.1 Detailed Description

Interface for transforming between coordinate systems.

Currently, the only implementation within OGR is **OGRProj4CT** (p. 257), which requires the PROJ.4 library to be available at run-time.

Also, see **OGRCreateCoordinateTransformation()** (p. 543) for creating transformations.

13.30.2 Member Function Documentation

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

OGRCoordinateTransformation (p. 99) destructor.

This function is the same as **OGRCoordinateTransformation::~~OGRCoordinateTransformation()** and **OCTDestroyCoordinateTransformation()** (p. 550)

This static method will destroy a **OGRCoordinateTransformation** (p. 99). 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

<i>poCT</i>	the object to delete
-------------	----------------------

Since

GDAL 1.7.0

13.30.2.2 `virtual OGRSpatialReference* OGRCoordinateTransformation::GetSourceCS ()`
`[pure virtual]`

Fetch internal source coordinate system.

Implemented in **OGRProj4CT** (p. 257).

13.30.2.3 `virtual OGRSpatialReference* OGRCoordinateTransformation::GetTargetCS ()`
`[pure virtual]`

Fetch internal target coordinate system.

Implemented in **OGRProj4CT** (p. 257).

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

13.30.2.4 `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. 102) allows extended success information to be captured indicating which points failed to transform.

Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	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. 258).

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

```
13.30.2.5 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

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.
<i>pabSuccess</i>	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. 258).

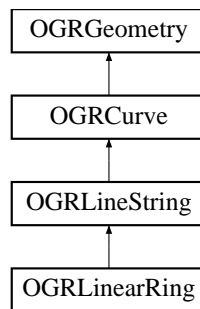
The documentation for this class was generated from the following files:

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

13.31 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.

13.31.1 Detailed Description

Abstract curve base class.

13.31.2 Member Function Documentation

13.31.2.1 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

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

Implemented in **OGRLineString** (p. 214).

Referenced by get_IsClosed().

13.31.2.2 `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 `StartPoint()`, `EndPoint()`, `OGRPoint::getX()`, and `OGRPoint::getY()`.

13.31.2.3 `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.216).

13.31.2.4 `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

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

Implemented in **OGRLineString** (p.224).

Referenced by `get_IsClosed()`.

13.31.2.5 `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

<i>dfDistance</i>	distance along the curve at which to sample position. This distance should be between zero and get_Length() (p. 104) for this curve.
<i>poPoint</i>	the point to be assigned the curve position.

Implemented in **OGRLineString** (p. 224).

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

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

13.32 OGRDataSource Class Reference

```
#include <ogr_sfrmts.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 **papszOptions=NULL)
Duplicate an existing layer.
- **OGRStyleTable** * **GetStyleTable** ()
Returns data source style table.
- void **SetStyleTableDirectly** (**OGRStyleTable** *poStyleTable)
Set data source style table.
- 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** *poResultSet)
*Release results of **ExecuteSQL()** (p. 109).*

- 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**

13.32.1 Detailed Description

This class represents a data source. A data source potentially consists of many layers (**OGRLayer** (p. 192)). 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. 105) is destroyed, all it's associated OGRLayers objects are also destroyed.

13.32.2 Member Function Documentation

13.32.2.1 **OGRLayer * OGRDataSource::CopyLayer** (OGRLayer * poSrcLayer, const char * pszNewName, char ** ppszOptions = 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 ppszOptions 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. 448).

Parameters

<i>poSrcLayer</i>	source layer.
<i>pszNew-Name</i>	the name of the layer to create.
<i>papsz-Options</i>	a StringList of name=value options. Options are driver specific.

Returns

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

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

Referenced by OGRSFDriver::CopyDataSource().

13.32.2.2 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

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

Returns

NULL is returned on failure, or a new **OGRLayer** (p. 192) handle on success.

Example:

```
#include "ogr_srf_frmts.h"
#include "cpl_string.h"

...

OGRLayer *poLayer;
char      *papszOptions;

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

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

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

Referenced by CopyLayer().

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

Delete the indicated layer from the datasource.

If this method is supported the ODS_CDeleteLayer capability will test TRUE on the **OGRDataSource** (p. 105).

This method is the same as the C function **OGR_DS_DeleteLayer()** (p. 450).

Parameters

<i>iLayer</i>	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.

13.32.2.4 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().

13.32.2.5 `void OGRDataSource::DestroyDataSource (OGRDataSource * poDS)`
`[static]`

Closes opened datasource and releases allocated resources.

This static method will close and destroy a datasource. It is equivalent 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. 450).

Parameters

<i>poDS</i>	pointer to allocated datasource object.
-------------	---

13.32.2.6 `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. 192) pointer representing a results set from the query. Note that this **OGRLayer** (p. 192) is in addition to the layers in the data store and must be destroyed with OGRDataSource::ReleaseResultSet() before the data source is closed (destroyed).

This method is the same as the C function **OGR_DS_ExecuteSQL()** (p. 450).

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

<i>pszStatement</i>	the SQL statement to execute.
<i>poSpatialFilter</i>	geometry which represents a spatial filter.
<i>pszDialect</i>	allows control of the statement dialect. By default it is assumed to be "generic" SQL, whatever that is.

Returns

an **OGRLayer** (p. 192) containing the results of the query. Deallocate with ReleaseResultSet().

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

13.32.2.7 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. 451).

Returns

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

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

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

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. 105) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayer()** (p. 451).

Parameters

<i>iLayer</i>	a layer number between 0 and GetLayerCount() (p. 111)-1.
---------------	---

Returns

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

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

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

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. 105) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayerByName()** (p. 452).

Parameters

<i>pszLayer-Name</i>	the layer name of the layer to fetch.
----------------------	---------------------------------------

Returns

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

References `GetLayerCount()`, `GetLayer()`, `OGRLayer::GetLayerDefn()`, and `OGRFeatureDefn::GetName()`.

Referenced by `ExecuteSQL()`.

13.32.2.10 `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. 452).

Returns

layer count.

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

13.32.2.11 `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. 259) 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. 453).

Returns

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

13.32.2.12 `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.

13.32.2.13 void OGRDataSource::GetStyleTable () [inline]

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.

13.32.2.14 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 `GetLayerCount()`, `GetLayer()`, and `OGRLayer::GetRefCount()`.

13.32.2.15 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()`.

13.32.2.16 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. 535).

Returns

`OGRErr_NONE` on success or an error code.

References `OGRSFDriverRegistrar::GetRegistrar()`.

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

Release results of **ExecuteSQL()** (p. 109).

This method should only be used to deallocate OGRLayers resulting from an **ExecuteSQL()** (p. 109) call on the same **OGRDataSource** (p. 105). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 105) may cause errors.

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

Parameters

<i>poResultsSet</i>	the result of a previous ExecuteSQL() (p. 109) call.
---------------------	---

13.32.2.18 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

<i>poDriver</i>	pointer to driver instance associated with the data source.
-----------------	---

Referenced by OGR_Dr_CreateDataSource(), and OGR_Dr_Open().

13.32.2.19 void OGRDataSource::SetStyleTable (OGRStyleTable * *poStyleTable*)
[inline]

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTableDirectly()** (p. 114) except that it does not assume ownership of the passed table.

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

Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

References OGRStyleTable::Clone().

13.32.2.20 `void OGRDataSource::SetStyleTableDirectly (OGRStyleTable * poStyleTable)`
`[inline]`

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTable()** (p. 113) except that it assumes ownership of the passed table.

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

Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

13.32.2.21 `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. 114) method on each of the layers. Conceptionally, calling **SyncToDisk()** (p. 114) on a datasource should include any work that might be accomplished by calling **SyncToDisk()** (p. 114) on layers in that data source.

In any event, you should always close any opened datasource with **OGRDataSource::DestroyDataSource()** (p. 109) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR_DS_SyncToDisk()** (p. 454).

Returns

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

References **GetLayerCount()**, **GetLayer()**, and **OGRLayer::SyncToDisk()**.

13.32.2.22 `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.

- **ODSCreateLayer**: 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 misspelling.

This method is the same as the C function **OGR_DS_TestCapability()** (p. 454).

Parameters

<i>psz-Capability</i>	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**

13.33 OGREnvelope Class Reference

```
#include <ogr_core.h>
```

13.33.1 Detailed Description

Simple container for a bounding region.

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

- **ogr_core.h**

13.34 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. 134).*
- 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. 133) 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. 133) 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.

13.34.1 Detailed Description

A simple feature, including geometry and attributes.

13.34.2 Constructor & Destructor Documentation

13.34.2.1 `OGRFeature::OGRFeature (OGRFeatureDefn * poDefnIn)`

Constructor.

Note that the **OGRFeature** (p. 115) will increment the reference count of it's defining **OGRFeatureDefn** (p. 134). Destruction of the **OGRFeatureDefn** (p. 134) 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. 455).

Parameters

<i>poDefnIn</i>	feature class (layer) definition to which the feature will adhere.
-----------------	--

References `OGRFeatureDefn::Reference()`, and `OGRFeatureDefn::GetFieldCount()`.

Referenced by `CreateFeature()`, and `Clone()`.

13.34.3 Member Function Documentation

13.34.3.1 `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. 134).

This method is the same as the C function **OGR_F_Clone()** (p. 455).

Returns

new feature, exactly matching this feature.

References `OGRFeature()`, `SetGeometry()`, `OGRFeatureDefn::GetFieldCount()`, `SetField()`, `GetStyleString()`, `SetStyleString()`, `SetFID()`, and `GetFID()`.

Referenced by `OGRGenSQLResultsLayer::GetFeature()`.

13.34.3.2 `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. 455).

Parameters

<i>poDefn</i>	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().

13.34.3.3 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. 456).

Parameters

<i>poFeature</i>	the feature to delete.
------------------	------------------------

Referenced by OGRDataSource::CopyLayer().

13.34.3.4 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. 456).

Parameters

<i>fpOut</i>	the stream to write to, such as stdout. If NULL stdout will be used.
<i>papsz-Options</i>	NULL terminated list of options (may be NULL)

References OGRFeatureDefn::GetName(), GetFID(), GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetNameRef(), OGRFieldDefn::GetFieldType-Name(), OGRFieldDefn::GetType(), IsFieldSet(), GetFieldAsString(), GetStyleString(), and OGRGeometry::dumpReadable().

13.34.3.5 OGRBoolean OGRFeature::Equal (OGRFeature * *poFeature*) [virtual]

Test if two features are the same.

Two features are considered equal if they share them (pointer equality) same **OGR-FeatureDefn** (p. 134), 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. 456).

Parameters

<i>poFeature</i>	the other feature to test this one against.
------------------	---

Returns

TRUE if they are equal, otherwise FALSE.

References GetFID(), GetDefnRef(), OGRFeatureDefn::GetFieldCount(), IsFieldSet(), - OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, GetFieldAs-Integer(), OFTReal, GetFieldAsDouble(), OFTString, GetFieldAsString(), OFTInteger-List, GetFieldAsIntegerList(), OFTRealList, GetFieldAsDoubleList(), OFTStringList, - GetFieldAsStringList(), OFTTime, OFTDate, OFTDateTime, GetFieldAsDateTime(), - OFTBinary, GetFieldAsBinary(), GetGeometryRef(), and OGRGeometry::Equals().

13.34.3.6 OGRFeatureDefn * OGRFeature::GetDefnRef () [inline]

Fetch feature definition.

This method is the same as the C function **OGR_F_GetDefnRef()** (p. 457).

Returns

a reference to the feature definition object.

Referenced by Equal().

13.34.3.7 long OGRFeature::GetFID () [inline]

Get feature identifier.

This method is the same as the C function **OGR_F_GetFID()** (p. 457).

Returns

feature id or OGRNullFID if none has been assigned.

Referenced by OGRDataSource::CopyLayer(), Clone(), GetFieldAsInteger(), GetFieldAsDouble(), GetFieldAsString(), DumpReadable(), Equal(), and OGRLayer::GetFeature().

13.34.3.8 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. 457).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>pnBytes</i>	location to put the number of bytes returned.

Returns

the field value. This data is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), and OFTBinary.

Referenced by Equal().

13.34.3.9 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. 458).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns

TRUE on success or FALSE on failure.

References OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), OFTDate, OFTTime, and OFTDateTime.

Referenced by Equal().

13.34.3.10 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. 458).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the field value.

References OGRFeatureDefn::GetFieldCount(), GetFID(), OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), OFTReal, OFTInteger, and OFTString.

Referenced by Equal(), and SetFrom().

13.34.3.11 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. 459).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>pnCount</i>	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. It's lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), and OFTRealList.

Referenced by Equal().

13.34.3.12 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. 459).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the field value.

References OGRFeatureDefn::GetFieldCount(), GetFID(), OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), OFTInteger, OFTReal, and OFTString.

Referenced by Equal(), and SetFrom().

13.34.3.13 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. 460).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>pnCount</i>	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. It's lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), and OFTIntegerList.

Referenced by Equal().

13.34.3.14 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. 460).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the field value. This string is internal, and should not be modified, or freed. It's lifetime may be very brief.

References `OGRFeatureDefn::GetFieldCount()`, `GetFID()`, `OGRGeometry::getGeometryName()`, `GetStyleString()`, `OGRGeometry::exportToWkt()`, `OGRFeatureDefn::GetFieldDefn()`, `IsFieldSet()`, `OGRFieldDefn::GetType()`, `OFTString`, `OFTInteger`, `OFTReal`, `OGRFieldDefn::GetWidth()`, `OGRFieldDefn::GetPrecision()`, `OFTDateTime`, `OFTDate`, `OFTTime`, `OFTIntegerList`, `OFTRealList`, `OFTStringList`, and `OFTBinary`.

Referenced by `DumpReadable()`, `Equal()`, `SetFrom()`, and `GetStyleString()`.

13.34.3.15 `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. 416).

This method is the same as the C function **OGR_F_GetFieldAsStringList()** (p. 461).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References `OGRFeatureDefn::GetFieldDefn()`, `IsFieldSet()`, `OGRFieldDefn::GetType()`, and `OFTStringList`.

Referenced by `Equal()`.

13.34.3.16 `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. 134).

This method is the same as the C function **OGR_F_GetFieldCount()** (p. 461).

Returns

count of fields.

Referenced by DumpReadable(), and SetFrom().

13.34.3.17 OGRFieldDefn * OGRFeature::GetFieldDefnRef (int *iField*) [inline]

Fetch definition for this field.

This method is the same as the C function **OGR_F_GetFieldDefnRef()** (p. 462).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the field definition (from the **OGRFeatureDefn** (p. 134)). This is an internal reference, and should not be deleted or modified.

Referenced by SetFrom().

13.34.3.18 int OGRFeature::GetFieldIndex (const char * *pszName*) [inline]

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. 137) method.

This method is the same as the C function **OGR_F_GetFieldIndex()** (p. 462).

Parameters

<i>pszName</i>	the name of the field to search for.
----------------	--------------------------------------

Returns

the field index, or -1 if no matching field is found.

Referenced by SetFrom(), and GetStyleString().

13.34.3.19 OGRGeometry * OGRFeature::GetGeometryRef () [inline]

Fetch pointer to feature geometry.

This method is the same as the C function **OGR_F_GetGeometryRef()** (p. 463).

Returns

pointer to internal feature geometry. This object should not be modified.

Referenced by Equal(), SetFrom(), and OGRLayer::GetExtent().

13.34.3.20 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. 463).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
---------------	--

Returns

the returned pointer is to an internal data structure, and should not be freed, or modified.

Referenced by SetFrom().

13.34.3.21 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. 542) for services available to parse it.

This method is the same as the C function **OGR_F_GetStyleString()** (p. 463).

Returns

a reference to a representation in string format, or NULL if there isn't one.

References GetFieldIndex(), and GetFieldAsString().

Referenced by Clone(), GetFieldAsString(), DumpReadable(), SetFrom(), and OGRStyleMgr::InitFromFeature().

13.34.3.22 int OGRFeature::IsFieldSet (int *iField*) const [inline]

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. 464).

Parameters

<i>iField</i>	the field to test.
---------------	--------------------

Returns

TRUE if the field has been set, otherwise false.

References OGRFieldDefn::Set().

Referenced by UnsetField(), GetFieldAsInteger(), GetFieldAsDouble(), GetFieldAsString(), GetFieldAsIntegerList(), GetFieldAsDoubleList(), GetFieldAsStringList(), GetFieldAsBinary(), GetFieldAsDateTime(), SetField(), DumpReadable(), Equal(), and SetFrom().

13.34.3.23 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. 464).

Parameters

<i>nFID</i>	the new feature identifier value to assign.
-------------	---

Returns

On success OGRErr_NONE, or on failure some other value.

Referenced by OGRGenSQLResultsLayer::GetFeature(), OGRDataSource::CopyLayer(), Clone(), and SetFrom().

13.34.3.24 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. 466).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>nValue</i>	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTReal, OFTString, and IsFieldSet().

Referenced by OGRGenSQLResultsLayer::GetFeature(), Clone(), SetField(), and SetFrom().

13.34.3.25 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. 465).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>dfValue</i>	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTReal, - OFTInteger, OFTString, and IsFieldSet().

13.34.3.26 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. 467).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>pszValue</i>	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTString, IsFieldSet(), OFTInteger, OFTReal, OFTDate, OFTTime, OFTDateTime, and OGR- ParseDate().

13.34.3.27 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. 467).

Parameters

<i>iField</i>	the field to set, from 0 to GetFieldCount() (p. 124)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger-

List, and SetField().

13.34.3.28 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. 466).

Parameters

<i>iField</i>	the field to set, from 0 to GetFieldCount() (p. 124)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTRealList, and SetField().

13.34.3.29 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. 468).

Parameters

<i>iField</i>	the field to set, from 0 to GetFieldCount() (p. 124)-1.
<i>papszValues</i>	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTStringList, and SetField().

13.34.3.30 void OGRFeature::SetField (int *iField*, OGRField * *puValue*)

Set field.

The passed value **OGRField** (p. 139) 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. 467).

Parameters

<i>iField</i>	the field to fetch, from 0 to GetFieldCount() (p. 124)-1.
<i>puValue</i>	the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger,

OFTReal, OFTString, IsFieldSet(), OFTDate, OFTTime, OFTDateTime, OFTIntegerList, OFTRealList, OFTStringList, and OFTBinary.

13.34.3.31 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. 465).

Parameters

<i>iField</i>	the field to set, from 0 to GetFieldCount() (p. 124)-1.
<i>nBytes</i>	bytes of data being set.
<i>pabyData</i>	the raw data being applied.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTBinary, and SetField().

13.34.3.32 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. 465).

Parameters

<i>iField</i>	the field to set, from 0 to GetFieldCount() (p. 124)-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>nSecond</i>	(0-59)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTDate, - OFTTime, and OFTDateTime.

13.34.3.33 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. 134). Field values are copied by corresponding field names. Field types do not have to exactly match. **SetField()** (p. 127) method conversion rules will be applied as needed.

This method is the same as the C function **OGR_F_SetFrom()** (p. 468).

Parameters

<i>poSrc-Feature</i>	the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	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()`, `GetFieldIndex()`, `GetFieldDefnRef()`, and `OGRFieldDefn::GetNameRef()`.

Referenced by `OGRDataSource::CopyLayer()`.

13.34.3.34 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. 134). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. 127) method conversion rules will be applied as needed. This is more efficient than **OGR_F_SetFrom()** (p. 468) 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. 469).

Parameters

<i>poSrc-Feature</i>	the feature from which geometry, and field values will be copied.
<i>panMap</i>	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.
<i>bForgiving</i>	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 SetFID(), SetGeometry(), GetGeometryRef(), SetStyleString(), GetStyleString(), GetFieldCount(), IsFieldSet(), UnsetField(), GetFieldDefnRef(), OGRFieldDefn::GetType(), OFTInteger, SetField(), GetFieldAsInteger(), OFTReal, GetFieldAsDouble(), OFTString, GetFieldAsString(), OFTDate, OFTDateTime, OFTTime, and GetRawFieldRef().

13.34.3.35 OGRErr OGRFeature::SetGeometry (OGRGeometry * poGeomIn)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometry-Directly()** (p. 132), 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. 469).

Parameters

<i>poGeomIn</i>	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. 134) (checking not yet implemented).

References OGRGeometry::clone().

Referenced by Clone(), and SetFrom().

13.34.3.36 OGRErr OGRFeature::SetGeometryDirectly (OGRGeometry * poGeomIn)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometry()** (p. 132), except that this method assumes ownership of the passed geometry.

This method is the same as the C function **OGR_F_SetGeometryDirectly()** (p. 470).

Parameters

<i>poGeomIn</i>	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. 134) (checking not yet implemented).

13.34.3.37 void OGRFeature::SetStyleString (const char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleString-Directly()** (p. 133) 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. 470).

Parameters

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--

Referenced by Clone(), SetFrom(), and OGRStyleMgr::SetFeatureStyleString().

13.34.3.38 void OGRFeature::SetStyleStringDirectly (char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. 133) except that it assumes ownership of the passed string.

This method is the same as the C function **OGR_F_SetStyleStringDirectly()** (p. 471).

Parameters

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--

13.34.3.39 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. 132).

After this call the **OGRFeature** (p. 115) will have a NULL geometry.

Returns

the pointer to the geometry.

13.34.3.40 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. 471).

Parameters

<i>iField</i>	the field to unset.
---------------	---------------------

References OGRFeatureDefn::GetFieldDefn(), IsFieldSet(), OGRFieldDefn::GetType(), OFTRealList, OFTIntegerList, OFTStringList, OFTString, and OFTBinary.

Referenced by SetFrom().

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

- **ogr_feature.h**
- ogrfeature.cpp

13.35 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. 134).
- 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.
- 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.

13.35.1 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. 134) 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. 134) with additional translator specific information.

13.35.2 Constructor & Destructor Documentation

13.35.2.1 OGRFeatureDefn::OGRFeatureDefn (const char * pszName = NULL)

Constructor.

The **OGRFeatureDefn** (p. 134) maintains a reference count, but this starts at zero. It is mainly intended to represent a count of OGRFeature's based on this definition.

This method is the same as the C function **OGR_FD_Create()** (p. 472).

Parameters

<i>pszName</i>	the name to be assigned to this layer/class. It does not need to be unique.
----------------	---

References wkbUnknown.

Referenced by Clone().

13.35.3 Member Function Documentation

13.35.3.1 void OGRFeatureDefn::AddFieldDefn (OGRFieldDefn * poNewDefn)

Add a new field definition.

This method should only be called while there are no **OGRFeature** (p. 115) objects in existence based on this **OGRFeatureDefn** (p. 134). The **OGRFieldDefn** (p. 140) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR_FD_AddFieldDefn()** (p. 471).

Parameters

<i>poNewDefn</i>	the definition of the new field.
------------------	----------------------------------

Referenced by Clone().

13.35.3.2 OGRFeatureDefn * OGRFeatureDefn::Clone ()

Create a copy of this feature definition.

Creates a deep copy of the feature definition.

Returns

the copy.

References OGRFeatureDefn(), GetName(), SetGeomType(), GetGeomType(), GetFieldCount(), AddFieldDefn(), and GetFieldDefn().

13.35.3.3 int OGRFeatureDefn::Dereference () [inline]

Decrements the reference count by one.

This method is the same as the C function **OGR_FD_Dereference()** (p. 472).

Returns

the updated reference count.

Referenced by Release().

13.35.3.4 int OGRFeatureDefn::GetFieldCount () [inline]

Fetch number of fields on this feature.

This method is the same as the C function **OGR_FD_GetFieldCount()** (p. 473).

Returns

count of fields.

Referenced by OGRDataSource::CopyLayer(), OGRDataSource::ExecuteSQL(), OGRFeature::OGRFeature(), OGRFeature::Clone(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsString(), OGRFeature::Equal(), and Clone().

13.35.3.5 OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (int iField)

Fetch field definition.

This method is the same as the C function **OGR_FD_GetFieldDefn()** (p. 473).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

Parameters

<i>iField</i>	the field to fetch, between 0 and GetFieldCount() (p. 136)-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 OGRDataSource::CopyLayer(), OGRDataSource::ExecuteSQL(), OGRFeature::UnsetField(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsStringList(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::SetField(), - OGRFeature::DumpReadable(), OGRFeature::Equal(), and Clone().

13.35.3.6 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. 474).

Parameters

<i>pszFieldName</i>	the field name to search for.
---------------------	-------------------------------

Returns

the field index, or -1 if no match found.

13.35.3.7 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. 474).

Returns

the base type for all geometry related to this definition.

Referenced by OGRDataSource::CopyLayer(), and Clone().

13.35.3.8 `const char * OGRFeatureDefn::GetName () [inline]`

Get name of this **OGRFeatureDefn** (p. 134).

This method is the same as the C function **OGR_FD_GetName()** (p. 474).

Returns

the name. This name is internal and should not be modified, or freed.

Referenced by **OGRDataSource::CopyLayer()**, **OGRDataSource::GetLayerByName()**, **OGRFeature::DumpReadable()**, **Clone()**, and **OGRSFDriver::CopyDataSource()**.

13.35.3.9 `int OGRFeatureDefn::GetReferenceCount () [inline]`

Fetch current reference count.

This method is the same as the C function **OGR_FD_GetReferenceCount()** (p. 475).

Returns

the current reference count.

13.35.3.10 `int OGRFeatureDefn::Reference () [inline]`

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. 115) objects referencing this definition.

This method is the same as the C function **OGR_FD_Reference()** (p. 475).

Returns

the updated reference count.

Referenced by **OGRFeature::OGRFeature()**.

13.35.3.11 `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. 476).

Parameters

<i>eNewType</i>	the new type to assign.
-----------------	-------------------------

Referenced by Clone().

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

- **ogr_feature.h**
- ogrfeaturedefn.cpp

13.36 OGRFeatureQuery Class Reference

Public Member Functions

- char ** **GetUsedFields** ()

13.36.1 Member Function Documentation

13.36.1.1 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. 416) when no longer required.

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

- **ogr_feature.h**
- ogrfeaturequery.cpp

13.37 OGRField Union Reference

```
#include <ogr_core.h>
```

13.37.1 Detailed Description

OGRFeature (p. 115) field attribute value union.

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

- **ogr_core.h**

13.38 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. 140) that is already part of an **OGRFeatureDefn** (p. 134).*
- **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**=OJ-Undefined)
Set defining parameters for a field in one call.
- void **SetDefault** (const **OGRField** *)
Set default field value.

Static Public Member Functions

- static const char * **GetFieldTypeName** (OGRFieldType)

Fetch human readable name for a field type.

13.38.1 Detailed Description

Definition of an attribute of an **OGRFeatureDefn** (p. 134).

13.38.2 Constructor & Destructor Documentation

13.38.2.1 OGRFieldDefn::OGRFieldDefn (const char * *pszNameIn*, OGRFieldType *eTypeIn*)

Constructor.

Parameters

<i>pszNameIn</i>	the name of the new field.
<i>eTypeIn</i>	the type of the new field.

13.38.2.2 OGRFieldDefn::OGRFieldDefn (OGRFieldDefn * *poPrototype*)

Constructor.

Create by cloning an existing field definition.

Parameters

<i>poPrototype</i>	the field definition to clone.
--------------------	--------------------------------

References `GetNameRef()`, `GetType()`, `SetJustify()`, `GetJustify()`, `SetWidth()`, `GetWidth()`, `SetPrecision()`, and `GetPrecision()`.

13.38.3 Member Function Documentation

13.38.3.1 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. 507).

Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

Returns

pointer to an internal static name string. It should not be modified or freed.

References OFTInteger, OFTReal, OFTString, OFTIntegerList, OFTRealList, OFTStringList, OFTBinary, OFTDate, OFTTime, and OFTDateTime.

Referenced by OGRFeature::DumpReadable(), and OGR_GetFieldName().

13.38.3.2 OGRJustification OGRFieldDefn::GetJustify () [inline]

Get the justification for this field.

This method is the same as the C function **OGR_Fld_GetJustify()** (p. 477).

Returns

the justification.

Referenced by OGRFieldDefn().

13.38.3.3 const char * OGRFieldDefn::GetNameRef () [inline]

Fetch name of this field.

This method is the same as the C function **OGR_Fld_GetNameRef()** (p. 477).

Returns

pointer to an internal name string that should not be freed or modified.

Referenced by OGRDataSource::ExecuteSQL(), OGRFeature::DumpReadable(), OGRFeature::SetFrom(), and OGRFieldDefn().

13.38.3.4 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. 478).

Returns

the precision.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

13.38.3.5 OGRFieldType OGRFieldDefn::GetType () [inline]

Fetch type of this field.

This method is the same as the C function **OGR_Fld_GetType()** (p. 478).

Returns

field type.

Referenced by OGRDataSource::ExecuteSQL(), OGRFeature::UnsetField(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsStringList(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::SetField(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::SetFrom(), and OGRFieldDefn().

13.38.3.6 int OGRFieldDefn::GetWidth () [inline]

Get the formatting width for this field.

This method is the same as the C function **OGR_Fld_GetWidth()** (p. 478).

Returns

the width, zero means no specified width.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

13.38.3.7 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. 479).

Parameters

<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>nPrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetName(), SetType(), SetWidth(), SetPrecision(), and SetJustify().

Referenced by OGRFeature::IsFieldSet().

13.38.3.8 void OGRFieldDefn::SetDefault (const OGRField * puDefaultIn)

Set default field value.

Currently use of **OGRFieldDefn** (p. 140) "defaults" is discouraged. This feature may be

fleshed out in the future.

References OFTInteger, OFTReal, and OFTString.

13.38.3.9 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. 479).

Parameters

<i>eJustify</i>	the new justification.
-----------------	------------------------

Referenced by OGRFieldDefn(), and Set().

13.38.3.10 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. 480).

Parameters

<i>pszNameIn</i>	the new name to apply.
------------------	------------------------

Referenced by Set().

13.38.3.11 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. 480).

Parameters

<i>nPrecision</i>	the new precision.
-------------------	--------------------

Referenced by OGRFieldDefn(), and Set().

13.38.3.12 void OGRFieldDefn::SetType (OGRFieldType *eType*) [inline]

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 140) that is already part of an **OGRFeatureDefn** (p. 134).

This method is the same as the C function **OGR_Fld_SetType()** (p. 480).

Parameters

<i>eType</i>	the new field type.
--------------	---------------------

Referenced by Set().

13.38.3.13 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. 480).

Parameters

<i>nWidth</i>	the new width.
---------------	----------------

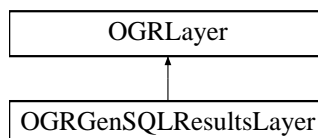
Referenced by OGRFieldDefn(), and Set().

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

- **ogr_feature.h**
- **ogrfielddefn.cpp**

13.39 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.

13.39.1 Member Function Documentation

13.39.1.1 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. 146) without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

This method is the same as the C function **OGR_L_GetExtent()** (p. 509).

Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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. 195).

References OGRLayer::GetExtent().

13.39.1.2 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. 120)) will be the same as nFID.

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. 146); 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. 146) call.

The returned feature should be free with **OGRFeature::DestroyFeature()** (p. 119).

This method is the same as the C function **OGR_L_GetFeature()** (p. 510).

Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

Returns

a feature now owned by the caller, or NULL on failure.

Reimplemented from **OGRLayer** (p. 196).

References **OGRFeature::Clone()**, **OGRFeature::SetField()**, **OGRFeature::SetFID()**, and **OGRLayer::GetFeature()**.

Referenced by **GetNextFeature()**.

13.39.1.3 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.

This method is the same as the C function **OGR_L_GetFeatureCount()** (p. 511).

Parameters

<i>bForce</i>	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. 197).

References **OGRLayer::GetFeatureCount()**.

13.39.1.4 OGRFeatureDefn * OGRGenSQLResultsLayer::GetLayerDefn () [virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p.134) is owned by the **OGRLayer** (p.192), 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. 512).

Returns

feature definition.

Implements **OGRLayer** (p. 198).

13.39.1.5 OGRFeature * OGRGenSQLResultsLayer::GetNextFeature () [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsibility of the caller to delete with **OGRFeature::DestroyFeature()** (p. 119).

Only features matching the current spatial filter (set with **SetSpatialFilter()** (p. 202)) will be returned.

This method implements sequential access to the features of a layer. The **Reset-Reading()** (p. 149) method can be used to start at the beginning again.

This method is the same as the C function **OGR_L_GetNextFeature()** (p. 512).

Returns

a feature, or NULL if no more features are available.

Implements **OGRLayer** (p. 198).

References **GetFeature()**, and **OGRLayer::GetNextFeature()**.

13.39.1.6 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. 513).

Returns

spatial filter geometry.

Reimplemented from **OGRLayer** (p. 199).

13.39.1.7 OGRSpatialReference * OGRGenSQLResultsLayer::GetSpatialRef ()
[virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 192) and should not be modified or freed by the application.

This method is the same as the C function **OGR_L_GetSpatialRef()** (p. 513).

Returns

spatial reference, or NULL if there isn't one.

Reimplemented from **OGRLayer** (p. 199).

References **OGRLayer::GetSpatialRef()**.

13.39.1.8 void OGRGenSQLResultsLayer::ResetReading () [virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. 148).

This method is the same as the C function **OGR_L_ResetReading()** (p. 514).

Implements **OGRLayer** (p. 200).

References **OGRLayer::SetAttributeFilter()**, **OGRLayer::SetSpatialFilter()**, and **OGRLayer::ResetReading()**.

13.39.1.9 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. 148) 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. 148) 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. 149) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. 149) and then calls **GetNextFeature()** (p. 148) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. 150) method with a value of **OLCFastSetNextByIndex**.

This method is the same as the C function **OGR_L_SetNextByIndex()** (p. 516).

Parameters

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

Returns

OGRErrorNone on success or an error code.

Reimplemented from **OGRLayer** (p. 202).

References **OGRLayer::SetNextByIndex()**.

13.39.1.10 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 exist 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. 146) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. 149) and **GetNextFeature()** (p. 148) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. 194) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 192) class may return FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. 201) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 192) class may return 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. 115) 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. 147)) 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. 146)) 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. 149) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. 194), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. 195) 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 method is the same as the C function **OGR_L_TestCapability()** (p. 518).

Parameters

<i>pszCap</i>	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. 204).

References **OGRLayer::TestCapability()**.

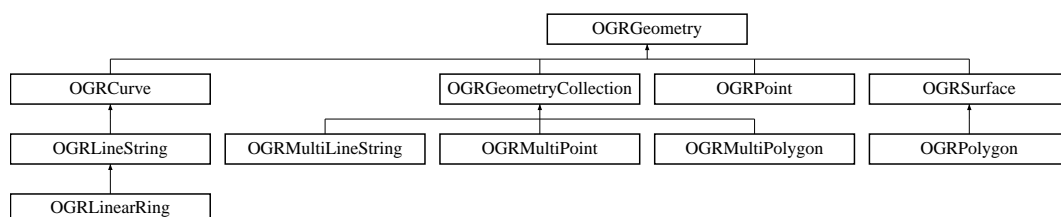
The documentation for this class was generated from the following files:

- ogr_gensql.h
- ogr_gensql.cpp

13.40 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 ps-Envelope 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
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 * **getBoundary** () 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 * Difference** (const **OGRGeometry ***) const
Compute difference.
- virtual **OGRGeometry * SymmetricDifference** (const **OGRGeometry ***) const
Compute symmetric difference.

13.40.1 Detailed Description

Abstract base class for all geometry classes.

Note that the family of spatial analysis methods (**Equal()**, **Disjoint()** (p. 157), ..., **Convex-Hull()** (p. 156), **Buffer()** (p. 155), ...) are not implemented at this time. Some other required and optional geometry methods have also been omitted at this time.

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)

13.40.2 Member Function Documentation

13.40.2.1 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. 268), 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. 483).

Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

References **OGRSpatialReference::Release()**, and **OGRSpatialReference::Reference()**.

Referenced by **OGRGeometryCollection::clone()**, **OGRGeometryCollection::transform()**, **OGRGeometryFactory::createFromWkb()**, **OGRGeometryFactory::createFromWkt()**, **OGRGeometryFactory::createFromFgf()**, **OGRLinearRing::clone()**, **OGRLineString::clone()**, **OGRLineString::transform()**, **OGRMultiLineString::clone()**, **OGRMultiPoint::clone()**, **OGRMultiPolygon::clone()**, **OGRPoint::clone()**, **OGRPoint::transform()**, **OGRPolygon::clone()**, and **OGRPolygon::transform()**.

13.40.2.2 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 segments 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. 483).

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

<i>dfDist</i>	the buffer distance to be applied.
<i>nQuadSegs</i>	the number of segments used to approximate a 90 degree (quadrant) of curvature.

Returns

the newly created geometry, or NULL if an error occurs.

13.40.2.3 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implemented in **OGRMultiLineString** (p. 227), **OGRMultiPoint** (p. 231), **OGRMultiPolygon** (p. 234), **OGRGeometryCollection** (p. 175), **OGRPolygon** (p. 249), **OGRLinearRing** (p. 207), **OGRLineString** (p. 214), and **OGRPoint** (p. 238).

Referenced by **OGRFeature::SetGeometry()**, and **OGRGeometryCollection::addGeometry()**.

13.40.2.4 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. 175), **OGRPolygon** (p. 249), and **OGRLinearRing** (p. 208).

13.40.2.5 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. 484).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

13.40.2.6 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. 485).

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.

13.40.2.7 `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. 487).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if they are crossing, otherwise FALSE.

13.40.2.8 `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. 488).

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

<i>poOther-Geom</i>	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.

13.40.2.9 `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. 488).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if they are disjoint, otherwise FALSE.

13.40.2.10 `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. 489).

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

<i>poOther-Geom</i>	the other geometry to compare against.
---------------------	--

Returns

the distance between the geometries or -1 if an error occurs.

13.40.2.11 `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. 489).

Parameters

<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.
<i>papsz-Options</i>	NULL terminated list of options (may be NULL)

References `getGeometryName()`, `getGeometryType()`, `wkbUnknown`, `wkbNone`, `wkbPoint`, `wkbPoint25D`, `wkbLineString`, `wkbLineString25D`, `OGRLineString::getNumPoints()`, `wkbPolygon`, `wkbPolygon25D`, `OGRPolygon::getExteriorRing()`, `OGRPolygon::getNumInteriorRings()`, `OGRPolygon::getInteriorRing()`, `wkbMultiPoint`, `wkbMultiPoint25D`, `wkbMultiLineString`, `wkbMultiLineString25D`, `wkbMultiPolygon`, `wkbMultiPolygon25D`, `wkbGeometryCollection`, `wkbGeometryCollection25D`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::getGeometryRef()`, `dumpReadable()`, `wkbLinearRing`, and `exportToWkt()`.

Referenced by `OGRFeature::DumpReadable()`, and `dumpReadable()`.

13.40.2.12 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. 490).

Implemented in **OGRGeometryCollection** (p. 176), **OGRPolygon** (p. 249), **OGRLineString** (p. 214), and **OGRPoint** (p. 239).

13.40.2.13 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. 490).

Returns

TRUE if equivalent or FALSE otherwise.

Implemented in **OGRGeometryCollection** (p. 176), **OGRPolygon** (p. 249), **OGRLineString** (p. 215), and **OGRPoint** (p. 239).

Referenced by `OGRFeature::Equal()`, and `OGRGeometryCollection::Equals()`.

13.40.2.14 char * OGRGeometry::exportToGML () 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.

This method is the same as the C function `OGR_G_ExportToGML()`.

Returns

A GML fragment or NULL in case of error.

13.40.2.15 `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()`.

Returns

A GeoJSON fragment or NULL in case of error.

13.40.2.16 `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()`.

Returns

A KML fragment or NULL in case of error.

13.40.2.17 `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. 490).

Parameters

<i>eByteOrder</i>	One of <code>wkbXDR</code> or <code>wkbNDR</code> indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <code>OGRGeometry::WkbSize()</code> (p. 172) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implemented in **OGRGeometryCollection** (p. 176), **OGRPolygon** (p. 250), **OGRLinearRing** (p. 208), **OGRLineString** (p. 215), and **OGRPoint** (p. 239).

Referenced by OGRGeometryCollection::exportToWkb().

13.40.2.18 `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. 491).

Parameters

<i>ppszDstText</i>	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. 228), **OGRMultiPoint** (p. 231), **OGRMultiPolygon** (p. 235), **OGRGeometryCollection** (p. 177), **OGRPolygon** (p. 250), **OGRLineString** (p. 215), and **OGRPoint** (p. 240).

Referenced by OGRFeature::GetFieldAsString(), dumpReadable(), OGRGeometryCollection::exportToWkt(), OGRMultiLineString::exportToWkt(), and OGRMultiPolygon::exportToWkt().

13.40.2.19 `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. 491).

Implemented in **OGRGeometryCollection** (p. 177), **OGRPolygon** (p. 250), **OGRLineString** (p. 216), and **OGRPoint** (p. 240).

13.40.2.20 `OGRGeometry * OGRGeometry::getBoundary () 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_GetBoundary()** (p. 492).

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.

13.40.2.21 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. 492).

Returns

in practice this always returns 2 indicating that coordinates are specified within a two dimensional space.

Referenced by OGRGeometryCollection::getGeometryType(), OGRGeometryCollection::addGeometryDirectly(), OGRGeometryCollection::importFromWkb(), OGRLinearRing::closeRings(), OGRLineString::getGeometryType(), OGRLineString::clone(), OGRLineString::WkbSize(), OGRLineString::getPoint(), OGRLineString::setNumPoints(), OGRLineString::setPoint(), OGRLineString::setPoints(), OGRLineString::exportToWkb(), OGRLineString::exportToWkt(), OGRLineString::Value(), OGRLineString::segmentize(), OGRMultiLineString::getGeometryType(), OGRMultiPoint::getGeometryType(), OGRMultiPoint::exportToWkt(), OGRMultiPolygon::getGeometryType(), OGRPolygon::getGeometryType(), OGRPolygon::addRing(), OGRPolygon::addRingDirectly(), OGRPolygon::WkbSize(), OGRPolygon::exportToWkb(), and OGRPolygon::exportToWkt().

13.40.2.22 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. 162)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 493).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implemented in **OGRGeometryCollection** (p. 178), **OGRPolygon** (p. 251), **OGRLineString** (p. 216), and **OGRPoint** (p. 240).

13.40.2.23 `void OGRGeometry::getEnvelope (OGREnvelope * psEnvelope) const`
[pure virtual]

Computes and returns the bounding envelope for this geometry in the passed ps-Envelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 493).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implemented in **OGRGeometryCollection** (p. 178), **OGRPolygon** (p. 251), **OGRLineString** (p. 217), and **OGRPoint** (p. 241).

Referenced by **Intersects()**, **OGRGeometryCollection::getEnvelope()**, **OGRGeometryFactory::organizePolygons()**, and **OGRLayer::GetExtent()**.

13.40.2.24 `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. 494).

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. 228), **OGRMultiPoint** (p. 232), **OGRMultiPolygon** (p. 235), **OGRGeometryCollection** (p. 178), **OGRPolygon** (p. 252), **OGRLinearRing** (p. 209), **OGRLineString** (p. 217), and **OGRPoint** (p. 241).

Referenced by **OGRFeature::GetFieldAsString()**, and **dumpReadable()**.

13.40.2.25 `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. 495).

Returns

the geometry type code.

Implemented in **OGRMultiLineString** (p. 228), **OGRMultiPoint** (p. 232), **OGRMultiPolygon** (p. 236), **OGRGeometryCollection** (p. 179), **OGRPolygon** (p. 252), **OGRLineString** (p. 217), and **OGRPoint** (p. 241).

Referenced by OGRBuildPolygonFromEdges(), dumpReadable(), OGRGeometryCollection::Equals(), OGRGeometryCollection::get_Area(), OGRGeometryFactory::forceToPolygon(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiLineString(), OGRLineString::Equals(), OGRMultiLineString::addGeometryDirectly(), OGRMultiPoint::addGeometryDirectly(), OGRMultiPolygon::addGeometryDirectly(), OGRPoint::Equals(), and OGRPolygon::Equals().

13.40.2.26 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. 496).

Returns

a reference to the spatial reference object. The object may be shared with many geometry objects, and should not be modified.

Referenced by transformTo(), OGRGeometryCollection::clone(), OGRLinearRing::clone(), OGRLineString::clone(), OGRMultiLineString::clone(), OGRMultiPoint::clone(), OGRMultiPolygon::clone(), OGRPoint::clone(), and OGRPolygon::clone().

13.40.2.27 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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of <i>pabyData</i> 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. 180), **OGRPolygon** (p. 253), **OGRLinearRing** (p. 209), **OGRLineString** (p. 220), and **OGRPoint** (p. 242).

Referenced by OGRGeometryFactory::createFromWkb().

13.40.2.28 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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 229), **OGRMultiPoint** (p. 232), **OGRMultiPolygon** (p. 236), **OGRGeometryCollection** (p. 181), **OGRPolygon** (p. 254), **OGRLineString** (p. 220), and **OGRPoint** (p. 243).

Referenced by OGRGeometryFactory::createFromWkt().

13.40.2.29 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. 166) method can be used to test if two geometries intersect.

This method is the same as the C function **OGR_G_Intersection()** (p. 498).

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

<i>poOtherGeom</i>	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.

13.40.2.30 `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 rigorous 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. 499).

Parameters

<i>poOtherGeom</i>	the other geometry to test against.
--------------------	-------------------------------------

Returns

TRUE if the geometries intersect, otherwise FALSE.

References getEnvelope().

13.40.2.31 `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. 181), **OGRPolygon** (p. 254), **OGRLineString** (p. 221), and **OGRPoint** (p. 243).

13.40.2.32 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. 500).

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.

13.40.2.33 OGRBoolean OGRGeometry::IsSimple () const [virtual]

Test if the geometry is simple.

This method is the same as the C function **OGR_G_IsSimple()** (p. 500).

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.

13.40.2.34 OGRBoolean OGRGeometry::IsValid () const [virtual]

Test if the geometry is valid.

This method is the same as the C function **OGR_G_IsValid()** (p. 501).

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.

13.40.2.35 `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. 501).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if they are overlapping, otherwise FALSE.

Referenced by OGRGeometryFactory::organizePolygons().

13.40.2.36 `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. 502)

Parameters

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented in **OGRGeometryCollection** (p. 182), **OGRPolygon** (p. 255), and **OGRLineString** (p. 221).

13.40.2.37 `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

<i>nNew-Dimension</i>	New coordinate dimension value, either 2 or 3.
-----------------------	--

Reimplemented in **OGRGeometryCollection** (p. 183), **OGRPolygon** (p. 255), **OGR-
LineString** (p. 221), and **OGRPoint** (p. 244).

Referenced by OGRGeometryCollection::setCoordinateDimension().

13.40.2.38 OGRGeometry * OGRGeometry::SymmetricDifference (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_SymmetricDifference()** (p. 503).

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

<i>poOther-Geom</i>	the other geometry.
---------------------	---------------------

Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

13.40.2.39 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. 504).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if they are touching, otherwise FALSE.

13.40.2.40 `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. 99) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 504).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR_NONE on success or an error code.

Implemented in **OGRGeometryCollection** (p. 183), **OGRPolygon** (p. 256), **OGRLineString** (p. 224), and **OGRPoint** (p. 245).

Referenced by `transformTo()`, and `OGRGeometryCollection::transform()`.

13.40.2.41 `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. 99) object it is significantly more expensive to use this method to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 99) in advance, and call **transform()** (p. 170) 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. 505).

Parameters

<i>poSR</i>	spatial reference system to transform to.
-------------	---

Returns

OGRERR_NONE on success, or an error code.

References `getSpatialReference()`, `OGRCreateCoordinateTransformation()`, and `transform()`.

13.40.2.42 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. 506).

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

<i>poOther-Geom</i>	the other geometry unioned with "this" geometry.
---------------------	--

Returns

a new geometry representing the union or NULL if an error occurs.

13.40.2.43 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. 506).

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

<i>poOther-Geom</i>	the geometry to compare to this geometry.
---------------------	---

Returns

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

13.40.2.44 `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. 507).

Returns

size of binary representation in bytes.

Implemented in **OGRGeometryCollection** (p. 183), **OGRPolygon** (p. 256), **OGRLinearRing** (p. 210), **OGRLineString** (p. 225), and **OGRPoint** (p. 245).

Referenced by OGRGeometryCollection::WkbSize(), OGRGeometryCollection::importFromWkb(), and OGRGeometryCollection::exportToWkb().

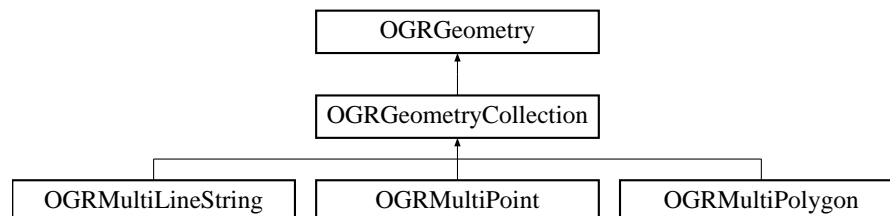
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometry.cpp

13.41 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_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 ps-Envelope 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.

13.41.1 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.

13.41.2 Member Function Documentation

13.41.2.1 OGRErr OGRGeometryCollection::addGeometry (const OGRGeometry * poNewGeom) [virtual]

Add a geometry to the container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 481).

Parameters

<i>poNew-Geom</i>	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 OGRGeometry::clone(), and addGeometryDirectly().

Referenced by clone(), OGRMultiLineString::clone(), OGRMultiPoint::clone(), and OGRMultiPolygon::clone().

13.41.2.2 OGRErr OGRGeometryCollection::addGeometryDirectly (OGRGeometry * poNewGeom) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 481).

There is no SFCOM analog to this method.

Parameters

<i>poNew-Geom</i>	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. 227), **OGRMultiPoint** (p. 230), and **OGRMultiPolygon** (p. 234).

References **OGRGeometry::getCoordinateDimension()**.

Referenced by **addGeometry()**, **importFromWkt()**, and **OGRGeometryFactory::createFromFgf()**.

13.41.2.3 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 155).

Reimplemented in **OGRMultiLineString** (p. 227), **OGRMultiPoint** (p. 231), and **OGRMultiPolygon** (p. 234).

References **OGRGeometryCollection()**, **OGRGeometry::assignSpatialReference()**, **OGRGeometry::getSpatialReference()**, and **addGeometry()**.

13.41.2.4 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. 156).

References `getGeometryType()`, and `wkbPolygon`.

13.41.2.5 `void OGRGeometryCollection::empty ()` [virtual]

Clear geometry information. This restores the geometry to its 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. 490).

Implements **OGRGeometry** (p. 159).

Referenced by `importFromWkb()`, `importFromWkt()`, `OGRMultiLineString::importFromWkt()`, `OGRMultiPoint::importFromWkt()`, and `OGRMultiPolygon::importFromWkt()`.

13.41.2.6 `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. 490).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 159).

References `OGRGeometry::getGeometryType()`, `getGeometryType()`, `getNumGeometries()`, `getGeometryRef()`, and `OGRGeometry::Equals()`.

13.41.2.7 `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. 490).

Parameters

<i>eByteOrder</i>	One of <code>wkbXDR</code> or <code>wkbNDR</code> indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least OGRGeometry::WkbSize() (p. 172) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 160).

References `getGeometryType()`, `OGRGeometry::exportToWkb()`, and `OGRGeometry::WkbSize()`.

13.41.2.8 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. 491).

Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer.
--------------------	--

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 161).

Reimplemented in **OGRMultiLineString** (p. 228), **OGRMultiPoint** (p. 231), and **OGR-MultiPolygon** (p. 235).

References `getNumGeometries()`, `OGRGeometry::exportToWkt()`, and `getGeometry-Name()`.

13.41.2.9 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. 491).

Implements **OGRGeometry** (p. 161).

13.41.2.10 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. 235).

References **OGRGeometry::getGeometryType()**, **wkbPolygon**, **wkbMultiPolygon**, **wkbLinearRing**, **wkbLineString**, **getGeometryName()**, and **wkbGeometryCollection**.

13.41.2.11 `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. 162)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 493).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 162).

13.41.2.12 `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. 493).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. 163).

References **OGRGeometry::getEnvelope()**.

13.41.2.13 `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. 494).

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. 163).

Reimplemented in **OGRMultiLineString** (p. 228), **OGRMultiPoint** (p. 232), and **OGRMultiPolygon** (p. 235).

Referenced by `importFromWkt()`, `exportToWkt()`, and `get_Area()`.

13.41.2.14 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>i</i>	the index of the geometry to fetch, between 0 and getNumGeometries() (p. 180) - 1.
----------	---

Returns

pointer to requested geometry.

Referenced by `OGRBuildPolygonFromEdges()`, `OGRGeometry::dumpReadable()`, `-Equals()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToMultiPoint()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRMultiLineString::clone()`, `OGRMultiLineString::exportToWkt()`, `OGRMultiPoint::clone()`, `OGRMultiPoint::exportToWkt()`, `OGRMultiPolygon::clone()`, `OGRMultiPolygon::exportToWkt()`, and `OGRMultiPolygon::get_Area()`.

13.41.2.15 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. 495).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 163).

Reimplemented in **OGRMultiLineString** (p. 228), **OGRMultiPoint** (p. 232), and **OGRMultiPolygon** (p. 236).

References `OGRGeometry::getCoordinateDimension()`, `wkbGeometryCollection25D`, and `wkbGeometryCollection`.

Referenced by `exportToWkb()`, `Equals()`, and `closeRings()`.

13.41.2.16 `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 `OGRBuildPolygonFromEdges()`, `OGRGeometry::dumpReadable()`, `exportToWkt()`, `Equals()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToMultiPoint()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRMultiLineString::clone()`, `OGRMultiLineString::exportToWkt()`, `OGRMultiPoint::clone()`, `OGRMultiPoint::exportToWkt()`, `OGRMultiPolygon::clone()`, `OGRMultiPolygon::exportToWkt()`, and `OGRMultiPolygon::get_Area()`.

13.41.2.17 `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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of <i>pabyData</i> 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. 164).

References `wkbGeometryCollection`, `wkbMultiPolygon`, `wkbMultiLineString`, `wkbMultiPoint`, `empty()`, `VSIMalloc2()`, `OGRGeometryFactory::createFromWkb()`, `OGRGeometry::getCoordinateDimension()`, and `OGRGeometry::WkbSize()`.

13.41.2.18 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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 165).

Reimplemented in **OGRMultiLineString** (p. 229), **OGRMultiPoint** (p. 232), and **OGRMultiPolygon** (p. 236).

References `empty()`, `getGeometryName()`, `OGRGeometryFactory::createFromWkt()`, and `addGeometryDirectly()`.

13.41.2.19 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. 166).

Referenced by OGRMultiPoint::exportToWkt().

13.41.2.20 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. 502).

Parameters

<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	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::forceToMultiPolygon(), OGRGeometryFactory::forceToMultiPoint(), and OGRGeometryFactory::forceToMultiLineString().

13.41.2.21 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. 502)

Parameters

<i>dfMax-Length</i>	the maximum distance between 2 points after segmentization
---------------------	--

Reimplemented from **OGRGeometry** (p. 168).

13.41.2.22 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

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented from **OGRGeometry** (p. 168).

References OGRGeometry::setCoordinateDimension().

13.41.2.23 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. 99) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 504).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 170).

References OGRGeometry::transform(), OGRGeometry::assignSpatialReference(), and OGRCoordinateTransformation::GetTargetCS().

13.41.2.24 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. 507).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 172).

References **OGRGeometry::WkbSize()**.

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

- **ogr_geometry.h**
- **ogrgeometrycollection.cpp**

13.42 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)**

13.42.1 Detailed Description

Create geometry objects from well known text/binary.

13.42.2 Member Function Documentation

13.42.2.1 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. 376)

Parameters

<i>dfCenterX</i>	center X
<i>dfCenterY</i>	center Y
<i>dfZ</i>	center Z
<i>dfPrimary-Radius</i>	X radius of ellipse.

<i>dfSecondaryRadius</i>	Y radius of ellipse.
<i>dfRotation</i>	rotation of the ellipse clockwise.
<i>dfStartAngle</i>	angle to first point on arc (clockwise of X-positive)
<i>dfEndAngle</i>	angle to last point on arc (clockwise of X-positive)
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.

Returns

OGRLineString (p. 210) geometry representing an approximation of the arc.

References **OGRLineString::setPoint()**.

**13.42.2.2 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. 184) object.

The C function **OGR_G_CreateFromFgf()** is the same as this method.

Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
<i>nBytes</i>	the number of bytes available in <i>pabyData</i> .
<i>pnBytesConsumed</i>	if not NULL, it will be set to the number of bytes consumed (at most <i>nBytes</i>).

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 **OGRLineString::setNumPoints()**, **OGRLineString::setPoint()**, **OGRPolygon::addRingDirectly()**, **OGRGeometryCollection::addGeometryDirectly()**, and **OGRGeometry::assignSpatialReference()**.

13.42.2.3 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. 151). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The C function `OGR_G_CreateFromGML()` is the same as this method.

Parameters

<i>pszData</i>	The GML fragment for the geometry.
----------------	------------------------------------

Returns

a geometry on succes, or NULL on error.

13.42.2.4 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. 172) 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. 184) object.

The C function `OGR_G_CreateFromWkb()` (p. 485) is the same as this method.

Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
<i>nBytes</i>	the number of bytes available in <i>pabyData</i> , 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 `createGeometry()`, `OGRGeometry::importFromWkb()`, and `OGRGeometry::assignSpatialReference()`.

Referenced by `OGRGeometryCollection::importFromWkb()`, and `OGR_G_CreateFromWkb()`.

13.42.2.5 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. 486) is the same as this method.

Parameters

<i>ppszData</i>	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.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	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.c_str();
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::importFromWkt()`, and `OGRGeometry::assignSpatialReference()`.

Referenced by `OGRGeometryCollection::importFromWkt()`, and `OGR_G_CreateFromWkt()`.

13.42.2.6 OGRGeometry * OGRGeometryFactory::createGeometry (OGRwkbGeometryType *eGeometryType*) [static]

Create an empty geometry of desired type.

This is equivalent 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. 486).

Parameters

<i>eGeometry-Type</i>	the type code of the geometry class to be instantiated.
-----------------------	---

Returns

the newly create geometry or NULL on failure.

References `wkbPoint`, `wkbLineString`, `wkbPolygon`, `wkbGeometryCollection`, `wkbMultiPolygon`, `wkbMultiPoint`, `wkbMultiLineString`, and `wkbLinearRing`.

Referenced by `createFromWkb()`, and `OGR_G_CreateGeometry()`.

13.42.2.7 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. 487).

Parameters

<i>poGeom</i>	the geometry to deallocate.
---------------	-----------------------------

Referenced by `OGR_G_DestroyGeometry()`.

13.42.2.8 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 `OGRGeometry::getGeometryType()`, `wkbGeometryCollection`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::getGeometryRef()`, `wkbLineString`, `OGRMultiLineString::addGeometryDirectly()`, `OGRGeometryCollection::removeGeometry()`, `wkbPolygon`, `OGRPolygon::getNumInteriorRings()`, `OGRPolygon::getExteriorRing()`, `OGRPolygon::getInteriorRing()`, `OGRLineString::getNumPoints()`, `OGRLineString::addSubLineString()`, and `wkbMultiPolygon`.

13.42.2.9 `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 `OGRGeometry::getGeometryType()`, `wkbGeometryCollection`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::getGeometryRef()`, `wkbPoint`, `OGRMultiPoint::addGeometryDirectly()`, and `OGRGeometryCollection::removeGeometry()`.

13.42.2.10 `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 `OGRGeometry::getGeometryType()`, `wkbGeometryCollection`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::getGeometryRef()`, `wkbPolygon`, `OGRMultiPolygon::addGeometryDirectly()`, and `OGRGeometryCollection::removeGeometry()`.

13.42.2.11 `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).

Returns

new geometry.

References OGRGeometry::getGeometryType(), wkbGeometryCollection, wkbMultiPolygon, OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::getGeometryRef(), wkbPolygon, OGRPolygon::getExteriorRing(), OGRPolygon::addRing(), OGRPolygon::getNumInteriorRings(), and OGRPolygon::getInteriorRing().

13.42.2.12 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().

13.42.2.13 OGRGeometry * OGRGeometryFactory::organizePolygons (OGRGeometry ** *papoPolygons*, int *nPolygonCount*, int * *pblsValidGeometry*, 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

<i>papo-Polygons</i>	array of geometry pointers - should all be <code>OGRPolygons</code> . Ownership of the geometries is passed, but not of the array itself.
<i>nPolygon-Count</i>	number of items in <code>papoPolygons</code>
<i>pblsValid-Geometry</i>	value will be set <code>TRUE</code> if result is valid or <code>FALSE</code> otherwise.
<i>papsz-Options</i>	a list of strings for passing options

Returns

a single resulting geometry (either **OGRPolygon** (p. 246) or **OGRMultiPolygon** (p. 233)).

References `haveGEOS()`, `OGRGeometry::getEnvelope()`, `wkbPolygon`, `OGRPolygon::get_Area()`, `OGRPolygon::getExteriorRing()`, `OGRLineString::getPoint()`, `OGRLinearRing::isClockwise()`, `OGRPolygon::addRing()`, `OGRLineString::getNumPoints()`, `OGRGeometry::Overlaps()`, and `OGRPolygon::exportToWkt()`.

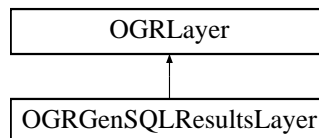
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- `ogrgeometryfactory.cpp`

13.43 OGRLayer Class Reference

```
#include <ogrsgf_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 **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 **SyncToDisk** ()
- Flush pending changes to disk.*
- **OGRStyleTable** * **GetStyleTable** ()
- Returns layer style table.*
- void **SetStyleTableDirectly** (**OGRStyleTable** *poStyleTable)
- Set layer style table.*
- 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.

- int **Reference** ()

Increment layer reference count.

- int **Dereference** ()

Decrement layer reference count.

- int **GetRefCount** () const

Fetch reference count.

13.43.1 Detailed Description

This class represents a layer of simple features, with access methods.

13.43.2 Member Function Documentation

13.43.2.1 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. 508).

Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

Returns

OGRERR_NONE on success.

Referenced by OGRDataSource::CopyLayer().

13.43.2.2 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. 134) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 134) used by a layer directly.

This function is the same as the C function **OGR_L_CreateField()** (p. 508).

Parameters

<i>poField</i>	field definition to write to disk.
<i>bApproxOK</i>	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().

13.43.2.3 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. 204) 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. 509).

Parameters

<i>nFID</i>	the feature id to be deleted from the layer
-------------	---

Returns

OGRERR_NONE on success.

13.43.2.4 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.

13.43.2.5 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 known. 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. 195) without setting a spatial filter.

Layers without any geometry may return `OGRERR_FAILURE` just indicating that no meaningful extents could be collected.

This method is the same as the C function **OGR_L_GetExtent()** (p. 509).

Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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. 146).

References `GetLayerDefn()`, `wkbNone`, `ResetReading()`, `GetNextFeature()`, `OGRFeature::GetGeometryRef()`, and `OGRGeometry::getEnvelope()`.

Referenced by `OGRGenSQLResultsLayer::GetExtent()`.

13.43.2.6 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. 120)) will be the same as `nFID`.

Use `OGRLayer::TestCapability(OLCRandomRead)` to establish if this layer supports efficient random access reading via **GetFeature()** (p. 196); 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. 196) call.

The returned feature should be free with **OGRFeature::DestroyFeature()** (p. 119).

This method is the same as the C function **OGR_L_GetFeature()** (p. 510).

Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

Returns

a feature now owned by the caller, or NULL on failure.

Reimplemented in **OGRGenSQLResultsLayer** (p. 146).

References `ResetReading()`, `GetNextFeature()`, and `OGRFeature::GetFID()`.

Referenced by `OGRGenSQLResultsLayer::GetFeature()`.

13.43.2.7 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.

This method is the same as the C function **OGR_L_GetFeatureCount()** (p. 511).

Parameters

<i>bForce</i>	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. 147).

References `ResetReading()`, and `GetNextFeature()`.

Referenced by `OGRGenSQLResultsLayer::GetFeatureCount()`.

13.43.2.8 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. 511).

Returns

fid column name.

13.43.2.9 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. 511).

Returns

geometry column name.

13.43.2.10 `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. 198) call.

Parameters

<i>pszTag</i>	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.

13.43.2.11 `OGRFeatureDefn * OGRLayer::GetLayerDefn ()` [pure virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. 134) is owned by the **OGRLayer** (p. 192), 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. 512).

Returns

feature definition.

Implemented in **OGRGenSQLResultsLayer** (p. 148).

Referenced by **OGRDataSource::CopyLayer()**, **OGRDataSource::GetLayerByName()**, **OGRDataSource::ExecuteSQL()**, **GetExtent()**, **SetAttributeFilter()**, and **OGRSFDriver::CopyDataSource()**.

13.43.2.12 `OGRFeature * OGRLayer::GetNextFeature ()` [pure virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsibility of the caller to delete with **OGRFeature::DestroyFeature()** (p. 119).

Only features matching the current spatial filter (set with **SetSpatialFilter()** (p. 202)) will be returned.

This method implements sequential access to the features of a layer. The **ResetReading()** (p. 200) method can be used to start at the beginning again.

This method is the same as the C function **OGR_L_GetNextFeature()** (p. 512).

Returns

a feature, or NULL if no more features are available.

Implemented in **OGRGenSQLResultsLayer** (p. 148).

Referenced by OGRGenSQLResultsLayer::GetNextFeature(), OGRDataSource::CopyLayer(), GetFeatureCount(), GetExtent(), GetFeature(), and SetNextByIndex().

13.43.2.13 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().

13.43.2.14 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. 513).

Returns

spatial filter geometry.

Reimplemented in **OGRGenSQLResultsLayer** (p. 148).

13.43.2.15 OGRSpatialReference * OGRLayer::GetSpatialRef () [inline, virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 192) and should not be modified or freed by the application.

This method is the same as the C function **OGR_L_GetSpatialRef()** (p. 513).

Returns

spatial reference, or NULL if there isn't one.

Reimplemented in **OGRGenSQLResultsLayer** (p. 149).

Referenced by **OGRGenSQLResultsLayer::GetSpatialRef()**, and **OGRDataSource::CopyLayer()**.

13.43.2.16 void OGRLayer::GetStyleTable () [inline]

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.

13.43.2.17 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.

13.43.2.18 void OGRLayer::ResetReading () [pure virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. 198).

This method is the same as the C function **OGR_L_ResetReading()** (p. 514).

Implemented in **OGRGenSQLResultsLayer** (p. 149).

Referenced by **OGRGenSQLResultsLayer::ResetReading()**, **OGRDataSource::CopyLayer()**, **GetFeatureCount()**, **GetExtent()**, **SetAttributeFilter()**, **GetFeature()**, **SetNextByIndex()**, and **SetSpatialFilter()**.

13.43.2.19 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. 198) 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 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 **ResetReading()** (p. 200)).

This method is the same as the C function **OGR_L_SetAttributeFilter()** (p. 514).

Parameters

<i>pszQuery</i>	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 **ResetReading()**, and **GetLayerDefn()**.

Referenced by **OGRGenSQLResultsLayer::ResetReading()**.

13.43.2.20 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. 115).

Use **OGRLayer::TestCapability(OLCRandomWrite)** to establish if this layer supports random access writing via **SetFeature()** (p. 201).

This method is the same as the C function **OGR_L_SetFeature()** (p. 515).

Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

Returns

OGRERR_NONE if the operation works, otherwise an appropriate error code.

13.43.2.21 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. 198) 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. 198) 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. 202) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. 200) and then calls **GetNextFeature()** (p. 198) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. 204) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR_L_SetNextByIndex()** (p. 516).

Parameters

<i>nIndex</i>	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. 149).

References **ResetReading()**, and **GetNextFeature()**.

Referenced by **OGRGenSQLResultsLayer::SetNextByIndex()**.

13.43.2.22 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. 198) 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. 163)) 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. 199)). In the future this may be generalized.

This method is the same as the C function **OGR_L_SetSpatialFilter()** (p. 516).

Parameters

<i>poFilter</i>	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 **OGREnSQLResultsLayer::ResetReading()**, and **SetSpatialFilterRect()**.

13.43.2.23 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. 198) 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. 199)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. 202). 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. 517).

Parameters

<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

References **OGRLineString::addPoint()**, **OGRPolygon::addRing()**, and **SetSpatialFilter()**.

13.43.2.24 void OGRLayer::SetStyleTable (OGRStyleTable * poStyleTable) [inline]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTableDirectly()** (p. 204) except that it does not assume ownership of the passed table.

This method is the same as the C function **OGR_L_SetStyleTable()**.

Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

References OGRStyleTable::Clone().

13.43.2.25 void OGRLayer::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*)
[inline]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTable()** (p. 203) except that it assumes ownership of the passed table.

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

Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

13.43.2.26 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. 109) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR_L_SyncToDisk()** (p. 518).

Returns

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

Referenced by OGRDataSource::SyncToDisk().

13.43.2.27 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. 196) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. 200) and **GetNextFeature()** (p. 198) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. 194) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 192) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. 201) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 192) 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. 115) 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. 197)) 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. 195)) 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. 202) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. 194), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. 195) 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 method is the same as the C function **OGR_L_TestCapability()** (p. 518).

Parameters

<i>pszCap</i>	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. 150).

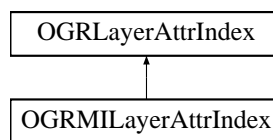
Referenced by OGRGenSQLResultsLayer::TestCapability(), and OGRDataSource::CopyLayer().

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

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrlayer.cpp

13.44 OGRLayerAttrIndex Class Reference

Inheritance diagram for OGRLayerAttrIndex:



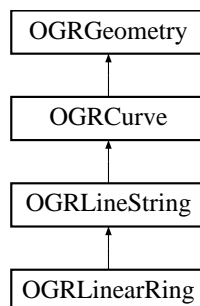
The documentation for this class was generated from the following files:

- ogr_attrind.h
- ogr_attrind.cpp

13.45 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

13.45.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivalent to an **OGRLineString** (p. 210), 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. 246).

The **OGRLinearRing** (p. 206) has no corresponding free standing well known binary representation, so **importFromWkb()** (p. 209) and **exportToWkb()** (p. 208) will not actually work. There is a non-standard GDAL WKT representation though.

Because **OGRLinearRing** (p. 206) is not a "proper" free standing simple features object, it cannot be directly used on a feature via **SetGeometry()**, and cannot generally be used with GEOS for operations like **Intersects()** (p. 166). Instead the polygon should be used, or the **OGRLinearRing** (p. 206) should be converted to an **OGRLineString** (p. 210) for such operations.

13.45.2 Member Function Documentation

13.45.2.1 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRLineString** (p. 214).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRLineString::setPoints().

13.45.2.2 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. 156).

References OGRLineString::getX(), OGRLineString::getY(), OGRLineString::getZ(), OGRGeometry::getCoordinateDimension(), and OGRLineString::addPoint().

13.45.2.3 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. 490).

Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least OGRGeometry::WkbSize() (p. 172) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRLineString** (p. 215).

13.45.2.4 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 $\text{pointCount}-1$, assuming the last point is a duplicate of the first.

Returns

computed area.

Referenced by OGRPolygon::get_Area().

13.45.2.5 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. 494).

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. 217).

13.45.2.6 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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	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. 220).

13.45.2.7 `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()`.

13.45.2.8 `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. 507).

Returns

size of binary representation in bytes.

Reimplemented from **OGRLineString** (p. 225).

Referenced by `OGR_G_AddGeometry()`, and `OGR_G_AddGeometryDirectly()`.

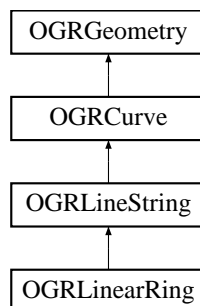
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- **ogrlinearring.cpp**

13.46 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 **) const
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 ps-Envelope 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 **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.

13.46.1 Detailed Description

Concrete representation of a multi-vertex line.

13.46.2 Member Function Documentation

13.46.2.1 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

<i>poPoint</i>	the point to assign to the new vertex.
----------------	--

References setPoint(), OGRPoint::getX(), OGRPoint::getY(), and OGRPoint::getZ().

Referenced by OGRBuildPolygonFromEdges(), OGRLayer::SetSpatialFilterRect(), and OGRLinearRing::closeRings().

13.46.2.2 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

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.
<i>z</i>	the Z coordinate to assign to the new point (defaults to zero).

References setPoint().

13.46.2.3 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

<i>poOtherLine</i>	the other OGRLineString (p. 210).
<i>nStartVertex</i>	the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.
<i>nEndVertex</i>	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()`.

13.46.2.4 **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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 155).

Reimplemented in **OGRLinearRing** (p. 207).

References `OGRLineString()`, `OGRGeometry::assignSpatialReference()`, `OGRGeometry::getSpatialReference()`, `setPoints()`, `setCoordinateDimension()`, and `OGRGeometry::getCoordinateDimension()`.

13.46.2.5 **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. 490).

Implements **OGRGeometry** (p. 159).

References `setNumPoints()`.

13.46.2.6 **void OGRLineString::EndPoint (OGRPoint * poPoint) const** [virtual]

Return the curve end point.

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

Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

Implements **OGRCurve** (p. 103).

References `getPoint()`.

Referenced by `Value()`.

13.46.2.7 `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. 490).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 159).

References **OGRGeometry::getGeometryType()**, **getGeometryType()**, **getNumPoints()**, **getX()**, **getY()**, and **getZ()**.

Referenced by **OGRPolygon::Equals()**.

13.46.2.8 `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. 490).

Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least OGRGeometry::WkbSize() (p. 172) byte in size.

Returns

Currently **OGRERR_NONE** is always returned.

Implements **OGRGeometry** (p. 160).

Reimplemented in **OGRLinearRing** (p. 208).

References **getGeometryType()**, and **OGRGeometry::getCoordinateDimension()**.

13.46.2.9 `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. 491).

Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer.
--------------------	--

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 161).

References `getGeometryName()`, and `OGRGeometry::getCoordinateDimension()`.

Referenced by `OGRPolygon::exportToWkt()`.

13.46.2.10 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. 491).

Implements **OGRGeometry** (p. 161).

13.46.2.11 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. 104).

13.46.2.12 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. 162)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 493).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 162).

13.46.2.13 void OGRLineStyle::getEnvelope (OGREnvelope * *psEnvelope*) const
[virtual]

Computes and returns the bounding envelope for this geometry in the passed ps-Envelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 493).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. 163).

Referenced by OGRPolygon::getEnvelope().

13.46.2.14 const char * OGRLineStyle::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. 494).

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. 163).

Reimplemented in **OGRLinearRing** (p. 209).

Referenced by importFromWkt(), and exportToWkt().

13.46.2.15 OGRwkbGeometryType OGRLineStyle::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. 495).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 163).

References OGRGeometry::getCoordinateDimension(), wkbLineString25D, and wkb-LineString.

Referenced by OGR_G_AddGeometry(), OGR_G_AddGeometryDirectly(), exportToWkb(), and Equals().

13.46.2.16 `int OGRLineString::getNumPoints () const [inline]`

Fetch vertex count.

Returns the number of vertices in the line string.

Returns

vertex count.

Referenced by `OGR_G_GetPointCount()`, `OGRBuildPolygonFromEdges()`, `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::organizePolygons()`, `addSubLineString()`, and `Equals()`.

13.46.2.17 `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>i</i>	the vertex to fetch, from 0 to getNumPoints() (p. 218)-1.
<i>poPoint</i>	a point to initialize with the fetched point.

References `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRGeometry::getCoordinateDimension()`, and `OGRPoint::setZ()`.

Referenced by `OGRGeometryFactory::organizePolygons()`, `StartPoint()`, and `EndPoint()`.

13.46.2.18 `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

<i>paoPoints-Out</i>	a buffer into which the points is written.
<i>padfZ</i>	the Z values that go with the points (optional, may be NULL).

13.46.2.19 `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

<i>iVertex</i>	the vertex to return, between 0 and getNumPoints() (p. 218)-1.
----------------	---

Returns

X value.

Referenced by `OGRBuildPolygonFromEdges()`, `OGRLinearRing::closeRings()`, and `-Equals()`.

13.46.2.20 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

<i>iVertex</i>	the vertex to return, between 0 and getNumPoints() (p. 218)-1.
----------------	---

Returns

X value.

Referenced by `OGRBuildPolygonFromEdges()`, `OGRLinearRing::closeRings()`, and `-Equals()`.

13.46.2.21 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

<i>iVertex</i>	the vertex to return, between 0 and getNumPoints() (p. 218)-1.
----------------	---

Returns

Z value.

Referenced by `OGRBuildPolygonFromEdges()`, `OGRLinearRing::closeRings()`, and `-Equals()`.

13.46.2.22 `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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	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. 164).

Reimplemented in **OGRLinearRing** (p. 209).

References wkbLineString, and setNumPoints().

13.46.2.23 `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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 165).

References `getGeometryName()`.

13.46.2.24 OGRBoolean OGRLineStyle::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. 166).

13.46.2.25 void OGRLineStyle::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. 502)

Parameters

<i>dfMax- Length</i>	the maximum distance between 2 points after segmentization
--------------------------	--

Reimplemented from **OGRGeometry** (p. 168).

References `OGRGeometry::getCoordinateDimension()`.

13.46.2.26 void OGRLineStyle::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

<i>nNew- Dimension</i>	New coordinate dimension value, either 2 or 3.
----------------------------	--

Reimplemented from **OGRGeometry** (p. 168).

Referenced by `clone()`, and `OGRPolygon::exportToWkt()`.

13.46.2.27 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. 222) is used to assign them to avoid reallocating the array larger with each call to **addPoint()** (p. 213).

This method has no SFCOM analog.

Parameters

<i>nNewPointCount</i>	the new number of points for geometry.
-----------------------	--

References `OGRGeometry::getCoordinateDimension()`.

Referenced by `OGRGeometryFactory::createFromFgf()`, `empty()`, `setPoint()`, `setPoints()`, `addSubLineString()`, and `importFromWkb()`.

13.46.2.28 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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>poPoint</i>	the value to assign to the vertex.

References `OGRPoint::getX()`, `OGRPoint::getY()`, and `OGRPoint::getZ()`.

Referenced by `OGRGeometryFactory::createFromFgf()`, `OGRGeometryFactory::approximateArcAngles()`, and `addPoint()`.

13.46.2.29 void OGRLineString::setPoint (int *iPoint*, double *xln*, double *yn*, double *zln*)

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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.
<i>zIn</i>	input Z coordinate to assign (defaults to zero).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

13.46.2.30 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

<i>nPointsIn</i>	number of points being passed in paoPointsIn
<i>paoPointsIn</i>	list of points being assigned.
<i>padfZ</i>	the Z values that go with the points (optional, may be NULL).

References setNumPoints(), and OGRGeometry::getCoordinateDimension().

Referenced by OGRLinearRing::clone(), clone(), transform(), OGRMultiLineString::importFromWkt(), OGRMultiPolygon::importFromWkt(), and OGRPolygon::importFromWkt().

13.46.2.31 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

<i>nPointsIn</i>	number of points being passed in padfX and padfY.
<i>padfX</i>	list of X coordinates of points being assigned.
<i>padfY</i>	list of Y coordinates of points being assigned.
<i>padfZ</i>	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints().

13.46.2.32 void OGRLineString::StartPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve start point.

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

Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

Implements **OGRCurve** (p. 104).

References getPoint().

Referenced by Value().

13.46.2.33 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. 99) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 504).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 170).

References OGRCoordinateTransformation::Transform(), setPoints(), OGRGeometry::assignSpatialReference(), and OGRCoordinateTransformation::GetTargetCS().

Referenced by OGRPolygon::transform().

13.46.2.34 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

<i>dfDistance</i>	distance along the curve at which to sample position. This distance should be between zero and get_Length() (p. 216) for this curve.
<i>poPoint</i>	the point to be assigned the curve position.

Implements **OGRCurve** (p. 104).

References `StartPoint()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRGeometry::getCoordinateDimension()`, `OGRPoint::setZ()`, and `EndPoint()`.

13.46.2.35 `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. 507).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 172).

Reimplemented in **OGRLinearRing** (p. 210).

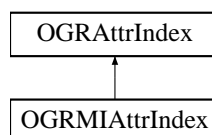
References `OGRGeometry::getCoordinateDimension()`.

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

- **ogr_geometry.h**
- **ogrlinestring.cpp**

13.47 OGRMIArrIndex Class Reference

Inheritance diagram for OGRMIArrIndex:

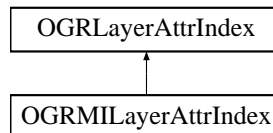


The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

13.48 OGRMILayerAttrIndex Class Reference

Inheritance diagram for OGRMILayerAttrIndex:



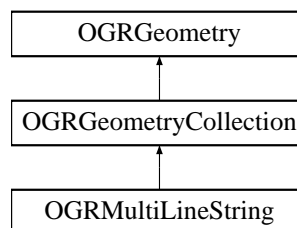
The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

13.49 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.

13.49.1 Detailed Description

A collection of OGRLineStrings.

13.49.2 Member Function Documentation

13.49.2.1 OGRErr OGRMultiLineString::addGeometryDirectly (OGRGeometry * poNewGeom) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 481).

There is no SFCOM analog to this method.

Parameters

<i>poNew-Geom</i>	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. 174).

References OGRGeometry::getGeometryType(), wkbLineString, and wkbLineString25-D.

Referenced by OGRGeometryFactory::forceToMultiLineString(), and importFromWkt().

13.49.2.2 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 175).

References `OGRGeometry::assignSpatialReference()`, `OGRGeometry::getSpatialReference()`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::addGeometry()`, and `OGRGeometryCollection::getGeometryRef()`.

13.49.2.3 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. 491).

Parameters

<i>ppszDstText</i>	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. 177).

References `OGRGeometryCollection::getNumGeometries()`, `OGRGeometryCollection::getGeometryRef()`, and `OGRGeometry::exportToWkt()`.

13.49.2.4 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. 494).

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. 178).

Referenced by `importFromWkt()`.

13.49.2.5 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. 495).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 179).

References `OGRGeometry::getCoordinateDimension()`, `wkbMultiLineString25D`, and `wkbMultiLineString`.

13.49.2.6 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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 181).

References `OGRGeometryCollection::empty()`, `getGeometryName()`, `OGRLineString::setPoints()`, and `addGeometryDirectly()`.

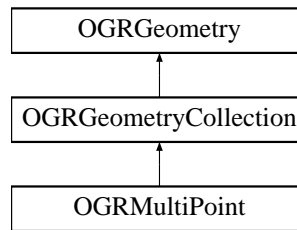
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- **ogrmultilinestring.cpp**

13.50 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.

13.50.1 Detailed Description

A collection of OGRPoints.

13.50.2 Member Function Documentation

13.50.2.1 OGRErr OGRMultiPoint::addGeometryDirectly (OGRGeometry * poNewGeom) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 481).

There is no SFCOM analog to this method.

Parameters

<i>poNew-Geom</i>	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. 174).

References OGRGeometry::getGeometryType(), wkbPoint, and wkbPoint25D.

Referenced by OGRGeometryFactory::forceToMultiPoint(), and importFromWkt().

13.50.2.2 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 175).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::addGeometry(), and OGRGeometryCollection::getGeometryRef().

13.50.2.3 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. 491).

Parameters

<i>ppszDstText</i>	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. 177).

References OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::IsEmpty(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), OGRPoint::IsEmpty(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), and OGRGeometry::getCoordinateDimension().

13.50.2.4 `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. 494).

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. 178).

Referenced by `exportToWkt()`, and `importFromWkt()`.

13.50.2.5 `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. 495).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 179).

References `OGRGeometry::getCoordinateDimension()`, `wkbMultiPoint25D`, and `wkbMultiPoint`.

13.50.2.6 `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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 181).

References OGRGeometryCollection::empty(), getGeometryName(), and addGeometryDirectly().

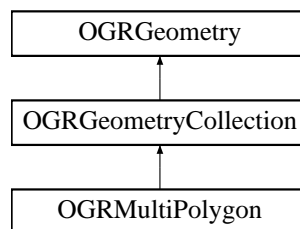
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipoint.cpp

13.51 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

13.51.1 Detailed Description

A collection of non-overlapping OGRPolygons.

Note that the IMultiSurface class hasn't been modelled, nor have any of its methods.

13.51.2 Member Function Documentation

13.51.2.1 OGRErr OGRMultiPolygon::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 481).

There is no SFCOM analog to this method.

Parameters

<i>poNew-Geom</i>	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. 174).

References OGRGeometry::getGeometryType(), wkbPolygon, and wkbPolygon25D.

Referenced by OGRGeometryFactory::forceToMultiPolygon(), and importFromWkt().

13.51.2.2 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 175).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::addGeometry(), and OGRGeometryCollection::getGeometryRef().

13.51.2.3 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. 491).

Parameters

<i>ppszDstText</i>	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. 177).

References OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::getGeometryRef(), and OGRGeometry::exportToWkt().

13.51.2.4 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. 177).

References OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::getGeometryRef(), and OGRPolygon::get_Area().

13.51.2.5 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. 494).

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. 178).

Referenced by `importFromWkt()`.

13.51.2.6 **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. 495).

Returns

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 179).

References `OGRGeometry::getCoordinateDimension()`, `wkbMultiPolygon25D`, and `wkbMultiPolygon`.

13.51.2.7 **OGRwkbGeometryType** **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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 181).

References OGRGeometryCollection::empty(), getGeometryName(), OGRLineString::setPoints(), OGRPolygon::addRingDirectly(), and addGeometryDirectly().

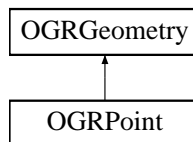
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipolygon.cpp

13.52 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 ps-Envelope 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.

13.52.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

13.52.2 Member Function Documentation

13.52.2.1 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 155).

References `OGRPoint()`, `OGRGeometry::assignSpatialReference()`, `OGRGeometry::getSpatialReference()`, and `setCoordinateDimension()`.

13.52.2.2 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. 490).

Implements **OGRGeometry** (p. 159).

Referenced by `OGRPoint()`, and `importFromWkt()`.

13.52.2.3 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. 490).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 159).

References `OGRGeometry::getGeometryType()`, `getGeometryType()`, `getX()`, `getY()`, and `getZ()`.

13.52.2.4 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. 490).

Parameters

<i>eByteOrder</i>	One of <code>wkbXDR</code> or <code>wkbNDR</code> indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least OGRGeometry::WkbSize() (p. 172) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 160).

References getGeometryType().

13.52.2.5 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. 491).

Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer.
--------------------	--

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 161).

13.52.2.6 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. 491).

Implements **OGRGeometry** (p. 161).

13.52.2.7 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. 162)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 493).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 162).

13.52.2.8 `void OGRPoint::getEnvelope (OGREnvelope * psEnvelope) const`
`[virtual]`

Computes and returns the bounding envelope for this geometry in the passed ps-Envelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 493).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. 163).

References `getX()`, and `getY()`.

13.52.2.9 `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. 494).

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. 163).

13.52.2.10 `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. 495).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 163).

References `wkbPoint`, and `wkbPoint25D`.

Referenced by `exportToWkb()`, `Equals()`, `OGRPolygon::Centroid()`, and `OGRPolygon::PointOnSurface()`.

13.52.2.11 `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 OGRCurve::get_IsClosed(), OGRLineString::setPoint(), OGRLineString::addPoint(), OGRMultiPoint::exportToWkt(), getEnvelope(), Equals(), OGRPolygon::Centroid(), and OGRPolygon::PointOnSurface().

13.52.2.12 `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 OGRCurve::get_IsClosed(), OGRLineString::setPoint(), OGRLineString::addPoint(), OGRMultiPoint::exportToWkt(), getEnvelope(), Equals(), OGRPolygon::Centroid(), and OGRPolygon::PointOnSurface().

13.52.2.13 `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::setPoint(), OGRLineString::addPoint(), OGRMultiPoint::exportToWkt(), and Equals().

13.52.2.14 `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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	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. 164).

References wkbPoint.

13.52.2.15 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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 165).

References empty().

13.52.2.16 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. 166).

Referenced by OGRMultiPoint::exportToWkt().

13.52.2.17 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

<i>nNew-Dimension</i>	New coordinate dimension value, either 2 or 3.
-----------------------	--

Reimplemented from **OGRGeometry** (p. 168).

Referenced by clone().

13.52.2.18 void OGRPoint::setX (double *x/n*) [inline]

Assign point X coordinate.

There is no corresponding SFCOM method.

Referenced by OGRLineString::getPoint(), OGRLineString::Value(), OGRPolygon::Centroid(), and OGRPolygon::PointOnSurface().

13.52.2.19 void OGRPoint::setY (double *y/n*) [inline]

Assign point Y coordinate.

There is no corresponding SFCOM method.

Referenced by OGRLineString::getPoint(), OGRLineString::Value(), OGRPolygon::Centroid(), and OGRPolygon::PointOnSurface().

13.52.2.20 void OGRPoint::setZ (double *z/n*) [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().

13.52.2.21 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. 99) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 504).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 170).

References **OGRCoordinateTransformation::Transform()**, **OGRGeometry::assignSpatialReference()**, and **OGRCoordinateTransformation::GetTargetCS()**.

13.52.2.22 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. 507).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 172).

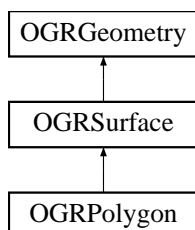
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- **ogrpoint.cpp**

13.53 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 **Centroid (OGRPoint *poPoint)** const
Compute the polygon centroid.
- 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 ps-Envelope 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.

13.53.1 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. 233) must be used for this.

13.53.2 Member Function Documentation

13.53.2.1 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. 206) remains the responsibility of the caller (an internal copy is made).

This method has no SFCOM analog.

Parameters

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

References OGRGeometry::getCoordinateDimension().

Referenced by OGRGeometryFactory::forceToPolygon(), OGRGeometryFactory::organizePolygons(), OGRLayer::SetSpatialFilterRect(), and clone().

13.53.2.2 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. 246), but otherwise this method operates the same as OGRPolygon::AddRing().

This method has no SFCOM analog.

Parameters

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

References OGRGeometry::getCoordinateDimension().

Referenced by OGRBuildPolygonFromEdges(), OGRGeometryFactory::createFromFgf(), and OGRMultiPolygon::importFromWkt().

13.53.2.3 int OGRPolygon::Centroid (OGRPoint * poPoint) const [virtual]

Compute the polygon centroid.

The centroid location is applied to the passed in **OGRPoint** (p. 237) object.

Returns

OGRERR_NONE on success or OGRERR_FAILURE on error.

Implements **OGRSurface** (p. 339).

References OGRPoint::getGeometryType(), wkbPoint, OGRPoint::setX(), OGRPoint::getX(), OGRPoint::setY(), and OGRPoint::getY().

13.53.2.4 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. 484).

Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 155).

References OGRPolygon(), OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and addRing().

13.53.2.5 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. 156).

13.53.2.6 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. 490).

Implements **OGRGeometry** (p. 159).

13.53.2.7 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. 490).

Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 159).

References OGRGeometry::getGeometryType(), getGeometryType(), getNumInteriorRings(), getExteriorRing(), OGRLineString::Equals(), and getInteriorRing().

13.53.2.8 `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. 490).

Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least OGRGeometry::WkbSize() (p. 172) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 160).

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

13.53.2.9 `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. 491).

Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer.
--------------------	--

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 161).

References getExteriorRing(), IsEmpty(), OGRLineString::setCoordinateDimension(), - OGRGeometry::getCoordinateDimension(), and OGRLineString::exportToWkt().

Referenced by OGRGeometryFactory::organizePolygons().

13.53.2.10 `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. 491).

Implements **OGRGeometry** (p. 161).

13.53.2.11 `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. 340).

References `getExteriorRing()`, `OGRLinearRing::get_Area()`, `getNumInteriorRings()`, and `getInteriorRing()`.

Referenced by `OGRGeometryFactory::organizePolygons()`, and `OGRMultiPolygon::get_Area()`.

13.53.2.12 `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. 162)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 493).

Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 162).

13.53.2.13 `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. 493).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. 163).

References `OGRLineString::getEnvelope()`.

13.53.2.14 `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. 246). 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. 155) 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. 246) is empty.

Referenced by `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::organizePolygons()`, `exportToWkt()`, `Equals()`, and `get_Area()`.

13.53.2.15 `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. 494).

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. 163).

13.53.2.16 `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. 495).

Returns

the geometry type code.

Implements **OGRGeometry** (p. 163).

References `OGRGeometry::getCoordinateDimension()`, `wkbPolygon25D`, and `wkbPolygon`.

Referenced by `exportToWkb()`, and `Equals()`.

13.53.2.17 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. 246). 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. 155) method to make a separate copy within the application.

Relates to the SFCOM `IPolygon::get_InternalRing()` method.

Parameters

<i>iRing</i>	internal ring index from 0 to <code>getNumInternalRings() - 1</code> .
--------------	--

Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. 246) is empty.

Referenced by `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryFactory::forceToMultiLineString()`, `Equals()`, and `get_Area()`.

13.53.2.18 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 `OGRBuildPolygonFromEdges()`, `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryFactory::forceToMultiLineString()`, `Equals()`, and `get_Area()`.

**13.53.2.19 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. 184) 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. 497).

Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	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. 164).

References wkbPolygon, and VSIMalloc2().

13.53.2.20 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. 184) 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. 498).

Parameters

<i>ppszInput</i>	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. 165).

References OGRLineString::setPoints().

13.53.2.21 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. 166).

Referenced by exportToWkt().

13.53.2.22 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

<i>poPoint</i>	point to be set with an internal point.
----------------	---

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implements **OGRSurface** (p. 340).

References OGRPoint::getGeometryType(), wkbPoint, OGRPoint::setX(), OGRPoint::getX(), OGRPoint::setY(), and OGRPoint::getY().

13.53.2.23 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. 502)

Parameters

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented from **OGRGeometry** (p. 168).

13.53.2.24 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

<i>nNew-Dimension</i>	New coordinate dimension value, either 2 or 3.
-----------------------	--

Reimplemented from **OGRGeometry** (p. 168).

13.53.2.25 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. 99) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 504).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 170).

References **OGRLineString::transform()**, **OGRGeometry::assignSpatialReference()**, and **OGRCoordinateTransformation::GetTargetCS()**.

13.53.2.26 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. 507).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. 172).

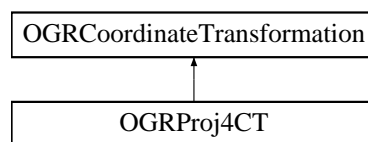
References `OGRGeometry::getCoordinateDimension()`.

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

- **ogr_geometry.h**
- `ogrpolygon.cpp`

13.54 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)

13.54.1 Member Function Documentation

13.54.1.1 **OGRSpatialReference *** **OGRProj4CT::GetSourceCS** () [virtual]

Fetch internal source coordinate system.

Implements **OGRCoordinateTransformation** (p. 101).

13.54.1.2 **OGRSpatialReference *** **OGRProj4CT::GetTargetCS** () [virtual]

Fetch internal target coordinate system.

Implements **OGRCoordinateTransformation** (p. 101).

13.54.1.3 `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. 258) allows extended success information to be captured indicating which points failed to transform.

Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	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. 101).

References TransformEx().

13.54.1.4 `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

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.
<i>pabSuccess</i>	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. 102).

Referenced by Transform().

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

- ogrct.cpp

13.55 OGRProj4Datum Struct Reference

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

- ogr_srs_proj4.cpp

13.56 OGRRawPoint Class Reference

```
#include <ogr_geometry.h>
```

13.56.1 Detailed Description

Simple container for a position.

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

- ogr_geometry.h

13.57 OGRSFDriver Class Reference

```
#include <ogr_ssf_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.

13.57.1 Detailed Description

Represents an operational format driver.

One **OGRSFDriver** (p. 259) 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. 263).

13.57.2 Member Function Documentation

13.57.2.1 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. 109) 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. 445).

Parameters

<i>poSrcDS</i>	source datasource
<i>pszNewName</i>	the name for the new data source.
<i>papszOptions</i>	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. 105) handle on success.

References **TestCapability()**, **GetName()**, **CreateDataSource()**, **OGRDataSource::GetLayerCount()**, **OGRDataSource::GetLayer()**, **OGRDataSource::CopyLayer()**, **OGRLayer::GetLayerDefn()**, and **OGRFeatureDefn::GetName()**.

13.57.2.2 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. 109) 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. 445).

Note

This method does **NOT** attach driver instance to the returned data source, so caller should expect that **OGRDataSource::GetDriver()** (p. 110) 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

<i>pszName</i>	the name for the new data source.
<i>papsz-Options</i>	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. 105) on success.

Referenced by **OGR_Dr_CreateDataSource()**, and **CopyDataSource()**.

13.57.2.3 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 **Test-Capability()** (p. 262) on **ODrCDeleteDataSource**.

This method is the same as the C function **OGR_Dr_DeleteDataSource()** (p. 446).

Parameters

<i>pszDataSource</i>	the name of the datasource to delete.
----------------------	---------------------------------------

Returns

OGRERR_NONE on success, and **OGRERR_UNSUPPORTED_OPERATION** if this is not supported by this driver.

13.57.2.4 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. 447).

Returns

driver name. This is an internal string and should not be modified or freed.

Referenced by CopyDataSource(), OGRSFDriverRegistrar::Open(), and OGRSFDriverRegistrar::RegisterDriver().

13.57.2.5 OGRDataSource * OGRSFDriver::Open (const char * pszName, int bUpdate = FALSE) [pure virtual]

Attempt to open file with this driver.

This method is what **OGRSFDriverRegistrar** (p. 263) uses to implement its **Open()** (p. 262) 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. 263)) 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.

This method is the same as the C function **OGR_Dr_Open()** (p. 447).

Parameters

<i>pszName</i>	the name of the file, or data source to try and open.
<i>bUpdate</i>	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. 105). This **OGRDataSource** (p. 105) should be closed by deleting the object when it is no longer needed.

Referenced by OGRSFDriverRegistrar::Open().

13.57.2.6 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 misspelling.

This method is the same as the C function **OGR_Dr_TestCapability()** (p. 447).

Parameters

<i>psz-Capability</i>	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

13.58 OGRSFDriverRegistrar Class Reference

```
#include <ogrsf_frmts.h>
```

Public Member Functions

- void **RegisterDriver** (**OGRSFDriver** *poDriver)
Add a driver to 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.

13.58.1 Detailed Description

Singleton manager for **OGRSFDriver** (p. 259) instances that will be used to try and open datasources. Normally the registrar is populated with standard drivers using the **OGRRegisterAll()** (p. 443) 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. 531).

13.58.2 Member Function Documentation

13.58.2.1 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 is 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 CPLGetExecPath(), CPLGetDirname(), CPLGetExtension(), CPLGetBasename(), CPLFormFilename(), and CPLGetSymbol().

Referenced by OGRRegisterAll().

13.58.2.2 OGRSFDriver * OGRSFDriverRegistrar::GetDriver (int iDriver)

Fetch the indicated driver.

This method is the same as the C function **OGRGetDriver()** (p. 532).

Parameters

<i>iDriver</i>	the driver index, from 0 to GetDriverCount() (p. 265)-1.
----------------	---

Returns

the driver, or NULL if iDriver is out of range.

Referenced by OGRGetDriver().

13.58.2.3 OGRSFDriver * OGRSFDriverRegistrar::GetDriverByName (const char * *pszName*)

Fetch the indicated driver.

This method is the same as the C function OGRGetDriverByName

Parameters

<i>pszName</i>	the driver name
----------------	-----------------

Returns

the driver, or NULL if no driver with that name is found

Referenced by OGRGetDriverByName().

13.58.2.4 int OGRSFDriverRegistrar::GetDriverCount (void)

Fetch the number of registered drivers.

This method is the same as the C function **OGRGetDriverCount()** (p. 533).

Returns

the drivers count.

Referenced by OGRGetDriverCount().

13.58.2.5 OGRDataSource * OGRSFDriverRegistrar::GetOpenDS (int *iDS*)

Return the iDS th datasource opened.

This method is the same as the C function **OGRGetOpenDS()** (p. 533).

Parameters

<i>iDS</i>	the index of the dataset to return (between 0 and GetOpenDSCount() (p. 266) - 1)
------------	---

Referenced by OGRGetOpenDS().

13.58.2.6 `int OGRSFDriverRegistrar::GetOpenDSCount ()` `[inline]`

Return the number of opened datasources.

This method is the same as the C function **OGRGetOpenDSCount()** (p. 533)

Returns

the number of opened datasources.

Referenced by **OGRGetOpenDSCount()**.

13.58.2.7 `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 existence.

Returns

the current driver registrar.

Referenced by **OGRDataSource::Release()**, **OGRDataSource::ExecuteSQL()**, **OGRRegisterAll()**, **Open()**, **OGRReleaseDataSource()**, **OGRGetOpenDSCount()**, **OGRGetOpenDS()**, **OGRRegisterDriver()**, and **OGRGetDriverByName()**.

13.58.2.8 `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. 263) methods directly, nor do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 263) instance.

If this method fails, **CPLGetLastErrorMsg()** (p. 381) can be used to check if there is an error message explaining why.

This method is the same as the C function **OGROpen()** (p. 533).

Parameters

<i>pszName</i>	the name of the file, or data source to open.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>ppoDriver</i>	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. 105). This **OGRDataSource** (p. 105) should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSource (p.105) *poDS;

poDS = OGRSFDriverRegistrar::Open (p.266) ( "polygon.shp" );
if( poDS == NULL )
{
    return;
}

... use the data source ...

OGRDataSource::DestroyDataSource(poDS);
```

References GetRegistrar(), OGRSFDriver::Open(), OGRDataSource::Reference(), OGRDataSource::GetDriver(), and OGRSFDriver::GetName().

Referenced by OGROpen().

13.58.2.9 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. 534).

Parameters

<i>poDriver</i>	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`

13.59 OGRSpatialReference Class Reference

```
#include <ogr_spatialref.h>
```

Public Member Functions

- **OGRSpatialReference** (const char *=NULL)
Constructor.
- virtual **~OGRSpatialReference** ()
OGRSpatialReference (p. 268) 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. 268).
- **OGRSpatialReference * CloneGeogCS** () const
Make a duplicate of the GEOGCS node of this OGRSpatialReference (p. 268) 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 *pdfPrjParams, 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 **importFromMlCoordSys** (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)
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.
- double **GetLinearUnits** (char **=NULL) const
Fetch linear projection units.
- 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 **IsLocal** () const
Check if local coordinate system.
- int **IsSameGeogCS** (const OGRSpatialReference *) const
Do the GeogCS'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 **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. 268).
- 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 *pRet=NULL) const
Get spheroid semi major axis.
- double **GetSemiMinor** (OGRErr *pRet=NULL) const
Get spheroid semi minor axis.
- double **GetInvFlattening** (OGRErr *pRet=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 **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. 268) destructor.

13.59.1 Detailed Description

This class represents 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 **OGREGeometry** (p. 151) objects.

Normally application code can fetch needed parameter values for this SRS using **GetAttrValue()** (p. 284), but in special cases the underlying parse tree (or **OGR_SRSNode** (p. 91) objects) can be accessed more directly.

See the `tutorial` for more information on how to use this class.

13.59.2 Constructor & Destructor Documentation

13.59.2.1 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. 304) with the WKT string.

Note that newly created objects are given a reference count of one.

The C function **OSRNewSpatialReference()** (p. 561) does the same thing as this constructor.

Parameters

<i>pszWKT</i>	well known text definition to which the object should be initialized, or NULL (the default).
---------------	--

References `importFromWkt()`.

13.59.2.2 OGRSpatialReference::~~OGRSpatialReference () [virtual]

OGRSpatialReference (p. 268) destructor.

The C function **OSRDestroySpatialReference()** (p. 553) does the same thing as this method. Preferred C++ method : **OGRSpatialReference::DestroySpatialReference()** (p. 277)

Deprecated

13.59.3 Member Function Documentation

13.59.3.1 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. 268) 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 **CPL-Error()** (p. 380).

This method is the same as the C function `OSRAutoidentifyEPSG()`.

Returns

`OGRERR_NONE` or `OGRERR_UNSUPPORTED_SRS`.

References `IsProjected()`, `IsGeographic()`, `GetAuthorityCode()`, `SetAuthority()`, `GetUTMZone()`, and `GetAuthorityName()`.

13.59.3.2 void OGRSpatialReference::Clear ()

Wipe current definition.

Returns **OGRSpatialReference** (p. 268) to a state with no definition, as it exists when first created. It does not affect reference counts.

Referenced by `SetStatePlane()`, `importFromERM()`, `importFromOzi()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, `importFromXML()`, `importFromWkt()`, `SetGeogCS()`, `CopyGeogCSFrom()`, `SetFromUserInput()`, and `importFromWMS-AUTO()`.

13.59.3.3 **OGRSpatialReference** * **OGRSpatialReference::Clone** () const

Make a duplicate of this **OGRSpatialReference** (p. 268).

This method is the same as the C function **OSRClone()** (p. 552).

Returns

a new SRS, which becomes the responsibility of the caller.

References **OGR_SRSNode::Clone()**.

Referenced by **exportToPrettyWkt()**.

13.59.3.4 **OGRSpatialReference** * **OGRSpatialReference::CloneGeogCS** () const

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. 268) object.

Returns

a new SRS, which becomes the responsibility of the caller.

References **GetAttrNode()**, **SetRoot()**, and **OGR_SRSNode::Clone()**.

13.59.3.5 **OGRERR** **OGRSpatialReference::CopyGeogCSFrom** (const **OGRSpatialReference** * *poSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. 268).

The GEOGCS information is copied into this **OGRSpatialReference** (p. 268) from another. If this object has a PROJCS root already, the GEOGCS is installed within it, otherwise it is installed as the root.

Parameters

<i>poSrcSRS</i>	the spatial reference to copy the GEOGCS information from.
-----------------	--

Returns

OGRERR_NONE on success or an error code.

References **GetAttrNode()**, **Clear()**, **OGR_SRSNode::FindChild()**, **OGR_SRSNode::DestroyChild()**, **OGR_SRSNode::InsertChild()**, **OGR_SRSNode::Clone()**, and **SetRoot()**.

Referenced by **importFromERM()**, **importFromESRI()**, **importFromOzi()**, **importFrom-Panorama()**, **importFromPCI()**, **importFromProj4()**, and **SetWellKnownGeogCS()**.

13.59.3.6 **int** **OGRSpatialReference::Dereference** ()

Decrements the reference count by one.

The method does the same thing as the C function **OSRDereference()** (p. 553).

Returns

the updated reference count.

Referenced by `Release()`.

13.59.3.7 `void OGRSpatialReference::DestroySpatialReference (OGRSpatialReference *
poSRS) [static]`

OGRSpatialReference (p. 268) destructor.

This static method will destroy a **OGRSpatialReference** (p. 268). 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. 553)

Parameters

<i>poSRS</i>	the object to delete
--------------	----------------------

Since

GDAL 1.7.0

13.59.3.8 `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.

Returns

TRUE or FALSE.

References `IsGeographic()`, `GetAuthorityName()`, `GetAttrNode()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetChild()`, and `OGR_SRSNode::GetValue()`.

13.59.3.9 `OGRERR OGRSpatialReference::exportToERM (char * pszProj, char * pszDatum, char
* pszUnits)`

Convert coordinate system to ERMapper format.

Parameters

<i>pszProj</i>	32 character buffer to receive projection name.
<i>pszDatum</i>	32 character buffer to receive datum name.
<i>pszUnits</i>	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 IsProjected(), IsGeographic(), GetAuthorityName(), GetAuthorityCode(), - GetAttrValue(), importFromDict(), GetUTMZone(), and GetLinearUnits().

13.59.3.10 OGRErr OGRSpatialReference::exportToMCoordSys (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 **OSRExportToMCoordSys()** (p. 553).

Parameters

<i>ppszResult</i>	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.

13.59.3.11 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

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piDatum</i>	Pointer to variable, where the coordinate system code will be returned.
<i>piEllips</i>	Pointer to variable, where the spheroid code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM projection system will be returned.
<i>padfPrjParams</i>	an existing 7 double buffer into which the projection parameters will be placed. See importFromPanorama() (p. 295) for the list of parameters.

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), IsLocal(), GetNormProjParm(), GetUTMZone(), GetSemiMajor(), and GetInvFlattening().

13.59.3.12 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 equivalent of the C function **OSRExportToPCI()** (p. 553).

Parameters

<i>ppszProj</i>	pointer to which dynamically allocated PCI projection definition will be assigned.
<i>ppszUnits</i>	pointer to which dynamically allocated units definition will be assigned.
<i>ppadfPrjParams</i>	pointer to which dynamically allocated array of 17 projection parameters will be assigned. See importFromPCI() (p. 296) for the list of parameters.

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetAttrNode(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), IsLocal(), GetLinearUnits(), GetNormProjParm(), GetUTMZone(), GetSemiMajor(), and GetInvFlattening().

13.59.3.13 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. 554).

Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
<i>bSimplify</i>	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().

13.59.3.14 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 CPL-Free() 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 equivalent of the C function **OSRExportToProj4()** (p. 554).

Parameters

<i>ppszProj4</i>	pointer to which dynamically allocated PROJ.4 definition will be assigned.
------------------	--

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetExtension(), GetAttrNode(), OGR_SRSNode::GetChildCount(), CPLAtof(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), IsGeographic(), GetNormProjParm(), GetUTMZone(), GetSemiMajor(), GetInvFlattening(), GetSemiMinor(), GetAuthorityName(), GetAuthorityCode(), and GetLinearUnits().

13.59.3.15 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. 554).

Parameters

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.
<i>ppadfPrjParams</i>	Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See importFromUSGS() (p. 299) for the list of parameters. Caller responsible to free this array.
<i>piDatum</i>	Pointer to variable, where the datum code will be returned.

Returns

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), IsLocal(), GetNormProjParm(), GetUTMZone(), GetSemi-Major(), and GetInvFlattening().

13.59.3.16 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. 554).

Parameters

<i>ppszResult</i>	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().

13.59.3.17 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 equivalent of the C function **OSRExportToXML()** (p. 555).

Parameters

<i>ppszRawXML</i>	pointer to which dynamically allocated XML definition will be assigned.
<i>pszDialect</i>	currently ignored. The dialect used is GML based.

Returns

OGRERR_NONE on success or an error code on failure.

References IsGeographic(), and IsProjected().

13.59.3.18 `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

<i>psz-Parameter</i>	projection parameter to look for
<i>poPROJCS</i>	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::GetChildCount()`, `OGR_SRSNode::GetChild()`, and `OGR_SRSNode::GetValue()`.

Referenced by `morphToESRI()`, and `GetProjParm()`.

13.59.3.19 `OGRERR OGRSpatialReference::Fixup ()`

Fixup as needed.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 268), 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 **OS-RFixupOrdering()** (p. 555)) so that the resulting WKT is valid.

This method should be expected to evolve over time to as problems are discovered. The following are among the fixup actions this method will take:

- Fixup the ordering of nodes to match the BNF WKT ordering, using the **Fixup-Ordering()** (p. 283) method.
- Add missing linear or angular units nodes.

This method is the same as the C function **OSRFixup()** (p. 555).

Returns

OGRERR_NONE on success or an error code if something goes wrong.

References `GetAttrNode()`, `OGR_SRSNode::FindChild()`, `SetLinearUnits()`, `SetAngularUnits()`, `CPLAtof()`, and `FixupOrdering()`.

Referenced by `morphToESRI()`.

13.59.3.20 OGRErr OGRSpatialReference::FixupOrdering ()

Correct parameter ordering to match CT Specification.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 268), 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. 555).

Returns

OGRErr_NONE on success or an error code if something goes wrong.

Referenced by importFromEPSGA(), morphFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromUSGS(), and Fixup().

13.59.3.21 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. 555).

Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the OGRSpatialReference (p. 268) and shouldn't be freed, or modified. It may be invalidated on the next OGRSpatialReference (p. 268) call.
-----------------	---

Returns

the value to multiply by angular distances to transform them to radians.

References GetAttrNode(), CPLAtof(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by morphToESRI().

13.59.3.22 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

<i>pszNode-Path</i>	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 `importFromEPSG()`, `SetStatePlane()`, `EPSGTreatsAsLatLong()`, `importFromESRI()`, `morphToESRI()`, `morphFromESRI()`, `exportToPCI()`, `importFromProj4()`, `exportToProj4()`, `GetAttrValue()`, `SetAngularUnits()`, `GetAngularUnits()`, `SetLinearUnitsAndUpdateParameters()`, `SetLinearUnits()`, `GetLinearUnits()`, `GetPrimeMeridian()`, `SetGeogCS()`, `CopyGeogCSFrom()`, `GetSemiMajor()`, `GetInvFlattening()`, `SetLocalCS()`, `SetProjCS()`, `SetProjection()`, `SetProjParm()`, `FindProjParm()`, `GetProjParm()`, `SetAuthority()`, `IsProjected()`, `IsGeographic()`, `CloneGeogCS()`, `IsSame()`, `SetTOWGS84()`, `GetTOWGS84()`, and `Fixup()`.

13.59.3.23 `const char * OGRSpatialReference::GetAttrValue (const char * pszNodeName, int iAttr = 0) const`

Fetch indicated attribute of named node.

This method uses **GetAttrNode()** (p. 283) 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. 555).

Parameters

<i>pszNode-Name</i>	the tree node to look for (case insensitive).
<i>iAttr</i>	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::GetChildCount()`, `OGR_SRSNode::GetChild()`, and `OGR_SRSNode::GetValue()`.

Referenced by `exportToERM()`, `morphToESRI()`, `morphFromESRI()`, `exportToPanorama()`, `exportToPCI()`, `exportToProj4()`, `exportToUSGS()`, `SetUTM()`, `GetUTMZone()`, `IsSameGeogCS()`, and `IsSame()`.

13.59.3.24 `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. 556).

Parameters

<i>pszTargetKey</i>	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(), morphToESRI(), and exportToProj4().

13.59.3.25 `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. 556).

Parameters

<i>pszTargetKey</i>	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 importFromEPSGA(), AutoidentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), morphToESRI(), and exportToProj4().

13.59.3.26 `const char * OGRSpatialReference::GetAxis (const char * pszTargetKey, int iAxis, OGRAxisOrientation * peOrientation)`

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. 556).

Parameters

<i>pszTarget-Key</i>	the coordinate system part to query ("PROJCS" or "GEOGCS").
<i>iAxis</i>	the axis to query (0 for first, 1 for second).
<i>pe-Orientation</i>	location into which to place the fetch orientation, may be NULL.

Returns

the name of the axis or NULL on failure.

References OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

13.59.3.27 `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

<i>pszTarget-Key</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszDefault</i>	the value to return if the extension is not found.

Returns

node value if successful or pszDefault on failure.

References OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by exportToProj4().

13.59.3.28 double OGRSpatialReference::GetInvFlattening (OGRErr * *pnErr* = NULL) const

Get spheroid inverse flattening.

This method does the same thing as the C function **OSRGetInvFlattening()** (p. 556).

Parameters

<i>pnErr</i>	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 GetAttrNode(), OGR_SRSNode::GetChildCount(), CPLAtof(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

13.59.3.29 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 or LOCAL_CS node for units.

This method does the same thing as the C function **OSRGetLinearUnits()** (p. 556)/

Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the OGRSpatialReference (p. 268) and shouldn't be freed, or modified. It may be invalidated on the next OGRSpatialReference (p. 268) call.
-----------------	---

Returns

the value to multiply by linear distances to transform them to meters.

References GetAttrNode(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), and CPLAtof().

Referenced by `SetStatePlane()`, `exportToERM()`, `importFromESRI()`, `morphToESRI()`, `exportToPCI()`, `importFromProj4()`, `exportToProj4()`, `SetLinearUnitsAndUpdateParameters()`, and `IsSame()`.

13.59.3.30 `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. 289) 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. 557).

Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in ogr_srs_api.h (p. 544).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References `GetProjParm()`.

Referenced by `SetStatePlane()`, `morphToESRI()`, `exportToPanorama()`, `exportToPCI()`, `exportToProj4()`, `exportToUSGS()`, and `GetUTMZone()`.

13.59.3.31 `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. 557).

Parameters

<i>ppszName</i>	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 `GetAttrNode()`, `OGR_SRSNode::GetChildCount()`, `CPLAtof()`, `OGR_SRSNode::GetChild()`, and `OGR_SRSNode::GetValue()`.

13.59.3.32 `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. 557).

Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in ogr_srs_api.h (p. 544).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References `GetAttrNode()`, `FindProjParm()`, `OGR_SRSNode::GetChild()`, `CPLAtof()`, and `OGR_SRSNode::GetValue()`.

Referenced by `morphToESRI()`, `morphFromESRI()`, `importFromProj4()`, `SetLinearUnitsAndUpdateParameters()`, `GetNormProjParm()`, `GetUTMZone()`, and `IsSame()`.

13.59.3.33 `int OGRSpatialReference::GetReferenceCount () const [inline]`

Fetch current reference count.

Returns

the current reference count.

13.59.3.34 `double OGRSpatialReference::GetSemiMajor (OGRErr * pnErr = NULL) const`

Get spheroid semi major axis.

This method does the same thing as the C function `OSRGetSemiMajor()` (p. 557).

Parameters

<i>pnErr</i>	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 GetAttrNode(), OGR_SRSNode::GetChildCount(), CPLAtof(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

13.59.3.35 double OGRSpatialReference::GetSemiMinor (OGRErr * *pnErr* = NULL) const

Get spheroid semi minor axis.

This method does the same thing as the C function **OSRGetSemiMinor()** (p. 557).

Parameters

<i>pnErr</i>	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 GetSemiMajor(), and GetInvFlattening().

Referenced by exportToProj4().

13.59.3.36 OGRErr OGRSpatialReference::GetTOWGS84 (double * *padfCoeff*, int *nCoeffCount* = 7) const

Fetch TOWGS84 parameters, if available.

Parameters

<i>padfCoeff</i>	array into which up to 7 coefficients are placed.
<i>nCoeffCount</i>	size of <i>padfCoeff</i> - defaults to 7.

Returns

OGRERR_NONE on success, or OGRERR_FAILURE if there is no TOWGS84 node available.

References GetAttrNode(), OGR_SRSNode::GetChildCount(), CPLAtof(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

13.59.3.37 int OGRSpatialReference::GetUTMZone (int * *pbNorth* = NULL) const

Get utm zone information.

This is the same as the C function **OSRGetUTMZone()** (p. 558).

Parameters

<i>pbNorth</i>	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()**, **morphToESRI()**, **exportToPanorama()**, **exportToPCI()**, **exportToProj4()**, and **exportToUSGS()**.

13.59.3.38 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. 268).

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

<i>pszDictFile</i>	the name of the dictionary file to load.
<i>pszCode</i>	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 **importFromEPSGA()**, **importFromERM()**, **exportToERM()**, **SetFromUserInput()**, and **importFromURN()**.

13.59.3.39 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. 292) 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. 558).

Parameters

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

Returns

OGRERR_NONE on success, or an error code on failure.

References importFromEPSGA(), GetAttrNode(), and OGR_SRSNode::StripNodes().

Referenced by SetStatePlane(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), SetWellKnownGeogCS(), and SetFromUserInput().

13.59.3.40 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. 292) 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. 292) for more details on operation of this method.

This method is the same as the C function **OSRImportFromEPSGA()** (p. 558).

Parameters

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

Returns

OGRERR_NONE on success, or an error code on failure.

References `importFromDict()`, `importFromProj4()`, `IsProjected()`, `GetAuthorityName()`, `SetAuthority()`, `IsGeographic()`, and `FixupOrdering()`.

Referenced by `importFromEPSG()`, `SetWellKnownGeogCS()`, `SetFromUserInput()`, and `importFromURN()`.

13.59.3.41 OGRErr OGRSpatialReference::importFromERM (const char * *pszProj*, const char * *pszDatum*, const char * *pszUnits*)

OGR WKT from ERMapper projection definitions.

Generates an **OGRSpatialReference** (p. 268) definition from an ERMapper datum and projection name. Based on the `ecw_cs.wkt` dictionary file from `gdal/data`.

Parameters

<i>pszProj</i>	the projection name, such as "NUTM11" or "GEOGRAPHIC".
<i>pszDatum</i>	the datum name, such as "NAD83".
<i>pszUnits</i>	the linear units "FEET" or "METERS".

Returns

OGRERR_NONE on success or OGRERR_UNSUPPORTED_SRS if not found.

References `Clear()`, `importFromDict()`, `SetLinearUnits()`, `IsLocal()`, and `CopyGeogCSFrom()`.

13.59.3.42 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. 268) 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. 293) by an automatical call to `morphFromESRI()` (p. 307).

Currently only GEOGRAPHIC, UTM, STATEPLANE, GREATBRITIAN_GRID, ALBERS, EQUIDISTANT_CONIC, and TRANSVERSE (mercator) projections are supported from old style files.

At this time there is no equivalent `exportToESRI()` method. Writing old style .prj files is not supported by **OGRSpatialReference** (p. 268). However the `morphToESRI()` (p. 307) and `exportToWkt()` (p. 281) methods can be used to generate output suitable to write to new style (Arc 8) .prj files.

This function is the equivalent of the C function **OSRImportFromESRI()** (p. 558).

Parameters

<i>papszPrj</i>	NULL terminated list of strings containing the definition.
-----------------	--

Returns

OGRERR_NONE on success or an error code in case of failure.

References `importFromWkt()`, `morphFromESRI()`, `SetUTM()`, `SetStatePlane()`, `SetAC-EA()`, `SetLCC()`, `SetEC()`, `SetTM()`, `SetPS()`, `SetLocalCS()`, `IsLocal()`, `GetAttrNode()`, `SetWellKnownGeogCS()`, `importFromEPSG()`, `CopyGeogCSFrom()`, `IsProjected()`, `GetLinearUnits()`, `SetLinearUnitsAndUpdateParameters()`, and `OGR_SRSNode::DestroyChild()`.

13.59.3.43 OGRErr OGRSpatialReference::importFromMlCoordSys (const char * *pszCoordSys*)

Import Mapinfo style CoordSys definition.

The **OGRSpatialReference** (p. 268) is initialized from the passed Mapinfo style CoordSys definition string.

This method is the equivalent of the C function **OSRImportFromMlCoordSys()** (p. 558).

Parameters

<i>pszCoordSys</i>	Mapinfo style CoordSys definition string.
--------------------	---

Returns

OGRERR_NONE on success, OGRERR_FAILURE on failure, OGRERR_UNSUPPORTED_OPERATION if MITAB library was not linked in.

13.59.3.44 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

<i>pszDatum</i>	Datum string. This is a fifth string in the OziExplorer .MAP file.
<i>pszProj</i>	Projection string. Search for line starting with "Map Projection" name in the OziExplorer .MAP file and supply it as a whole in this parameter.
<i>pszProjParms</i>	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(), SetMercator(), CPLAtof(), SetTM(), SetLCC(), SetSinusoidal(), SetACEA(), SetLocalCS(), IsLocal(), importFromEPSG(), CopyGeogCSFrom(), SetWellKnownGeogCS(), IsProjected(), SetLinearUnits(), and FixupOrdering().

13.59.3.45 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

<i>iProjSys</i>	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

```

Parameters

<i>iDatum</i>	Input coordinate system.
---------------	--------------------------

Supported Datums

```

1: Pulkovo, 1942
2: WGS, 1984
3: OSGB 1936 (British National Grid)
9: Pulkovo, 1995

```

Parameters

<i>iEllips</i>	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)

```

Parameters

<i>padfPrjParams</i>	Array of 7 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

```

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(), SetUTM(), SetWagner(), SetMercator(), SetPS(), SetPolyconic(), SetEC(), SetLCC(), SetTM(), SetStereographic(), SetAE(), SetGnomonic(), SetMollweide(), SetLAEA(), SetEquirectangular(), SetCEA(), SetIWMPPolyconic(), SetLocalCS(), IsLocal(), importFromEPSG(), CopyGeogCSFrom(), SetGeogCS(), SetAuthority(), SetWellKnownGeogCS(), IsProjected(), SetLinearUnits(), and FixupOrdering().

13.59.3.46 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. 296) function.

This function is the equivalent of the C function **OSRImportFromPCI()** (p. 559).

Parameters

<i>pszProj</i>	NULL terminated string containing the definition. Looks like "pppppppppppp Ennn" or "pppppppppppp Dnnn", where "pppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" --- a datum code.
<i>pszUnits</i>	Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.
<i>padfPrj-Params</i>	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 supliet 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(), SetLocalCS(), SetLinearUnits(), SetACEA(), SetAE(), SetEC(), - SetEquirectangular2(), SetGnomonic(), SetLAEA(), SetLCC(), SetMC(), SetMercator(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetStereographic(), SetSinusoidal(), SetStatePlane(), SetLinearUnitsAndUpdateParameters(), SetTM(), SetUTM(), SetVDG(), IsProjected(), IsGeographic(), importFromEPSG(), CopyGeogCS-From(), SetGeogCS(), SetAuthority(), SetWellKnownGeogCS(), IsLocal(), SetAngularUnits(), and FixupOrdering().

13.59.3.47 OGRErr OGRSpatialReference::importFromProj4 (const char * *pszProj4*)

Import PROJ.4 coordinate string.

The **OGRSpatialReference** (p. 268) is initialized from the passed PROJ.4 style coordinate system string. In addition to many +proj formulations which have OGC equivalents, it is also possible to import "+init=epsg:n" style definitions. These are passed to **importFromEPSG()** (p. 292). 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. 268) 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. 559).

Parameters

<i>pszProj4</i>	the PROJ.4 style string.
-----------------	--------------------------

Returns

OGRERR_NONE on success or OGRERR_CORRUPT_DATA on failure.

References `Clear()`, `importFromEPSG()`, `SetBonne()`, `SetCS()`, `SetNZMG()`, `SetCEA()`, `SetTM()`, `SetUTM()`, `SetMercator()`, `SetPS()`, `SetOS()`, `SetStereographic()`, `SetEquirectangular2()`, `SetEquirectangular()`, `SetGaussSchreiberTMercator()`, `SetGnomonic()`, `SetOrthographic()`, `SetLAEA()`, `SetAE()`, `SetEC()`, `SetMC()`, `SetMollweide()`, `SetEckert()`, `SetPolyconic()`, `SetACEA()`, `SetRobinson()`, `SetVDG()`, `SetSinusoidal()`, `SetGS()`, `SetGH()`, `SetGEOS()`, `SetLCC1SP()`, `SetLCC()`, `SetHOM()`, `SetKrovak()`, `SetIWMPolyconic()`, `SetWagner()`, `SetTPED()`, `SetWellKnownGeogCS()`, `CopyGeogCSFrom()`, `CPLAtof()`, `SetGeogCS()`, `SetTOWGS84()`, `IsProjected()`, `IsLocal()`, `CPLAtofM()`, `SetLinearUnits()`, `GetLinearUnits()`, `GetAttrNode()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetValue()`, `SetNormProjParm()`, `GetProjParm()`, and `SetExtension()`.

Referenced by `importFromEPSGA()`, and `SetFromUserInput()`.

13.59.3.48 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. 559) function.

Parameters

<i>pszUrl</i>	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 download data.

References `CPLHTTPResult::nDataLen`, `CPLHTTPResult::pabyData`, `CPLHTTPResult::nStatus`, `CPLHTTPResult::pszErrBuf`, and `SetFromUserInput()`.

Referenced by `SetFromUserInput()`.

13.59.3.49 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. 312) which can normally be used for URNs.

Parameters

<i>pszURN</i>	the urn string.
---------------	-----------------

Returns

OGRERR_NONE on success or an error code.

References `importFromEPSGA()`, `importFromDict()`, `SetWellKnownGeogCS()`, and `importFromWMSAUTO()`.

Referenced by `SetFromUserInput()`.

13.59.3.50 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. G-CTP operates on angles in packed DMS format (see **CPLDecToPackedDMS()** (p. 357) 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. 559). Note that the **bAnglesInPackedDMSFormat** parameter is only present in the C++ method. The C function assumes **bAnglesInPackedFormat** = TRUE.

Parameters

<i>iProjSys</i>	Input projection system code, used in GCTP.
<i>iZone</i>	Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. <i>iZone</i> ignored for all other projections.
<i>padfPrj-Params</i>	Array of 15 coordinate system parameters. These parameters differs for different projections.

Projection Transformation Package Projection Parameters

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
```

Code & Projection Id	Array Element				
	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

<i>iDatum</i>	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]`:

$$\text{semiminor} = \sqrt{1.0 - \text{ES}} * \text{semimajor}$$
 where

$$\text{ES} = \text{eccentricity squared}$$
- 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:

Supported Datums

```

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

<i>bAnglesInPackedDMSFormat</i>	TRUE if the angle values specified in the <code>padfPrjParams</code> array should be in the packed DMS format
---------------------------------	---

Returns

OGRERR_NONE on success or an error code in case of failure.

References `SetUTM()`, `SetStatePlane()`, `SetACEA()`, `SetLCC()`, `SetMercator()`, `SetP-S()`, `SetPolyconic()`, `SetEC()`, `SetTM()`, `SetStereographic()`, `SetLAEA()`, `SetAE()`, `SetGnomonic()`, `SetOrthographic()`, `SetSinusoidal()`, `SetEquirectangular2()`, `SetMC()`, `SetVDG()`, `SetHOM()`, `SetHOM2PNO()`, `SetRobinson()`, `SetMollweide()`, `SetWagner()`, `-SetLocalCS()`, `IsLocal()`, `SetGeogCS()`, `SetAuthority()`, `SetWellKnownGeogCS()`, `IsProjected()`, `SetLinearUnits()`, and `FixupOrdering()`.

13.59.3.51 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. 559).

Parameters

<i>ppszInput</i>	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 `Clear()`, and `OGR_SRSNode::importFromWkt()`.

Referenced by `importFromDict()`, `importFromESRI()`, `OGRSpatialReference()`, `OSR-NewSpatialReference()`, `SetWellKnownGeogCS()`, and `SetFromUserInput()`.

13.59.3.52 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

<i>pszDefinition</i>	the WMSAUTO string
----------------------	--------------------

Returns

OGRERR_NONE on success or an error code.

References CPLAtof(), Clear(), SetUTM(), SetTM(), SetOrthographic(), Set-Equirectangular(), SetMollweide(), SetLinearUnits(), SetAuthority(), and SetWellKnown-GeogCS().

Referenced by SetFromUserInput(), and importFromURN().

13.59.3.53 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. 560)

Parameters

<i>pszXML</i>	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().

13.59.3.54 int OGRSpatialReference::IsGeographic () const

Check if geographic coordinate system.

This method is the same as the C function **OSRIsGeographic()** (p. 560).

Returns

TRUE if this spatial reference is geographic ... that is the root is a GEOGCS node.

References OGR_SRSNode::GetValue(), and GetAttrNode().

Referenced by importFromEPSGA(), AutoIdentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), importFromPCI(), exportToProj4(), exportToXML(), and SetWellKnown-GeogCS().

13.59.3.55 int OGRSpatialReference::IsLocal () const

Check if local coordinate system.

This method is the same as the C function **OSRIsLocal()** (p. 560).

Returns

TRUE if this spatial reference is local ... that is the root is a LOCAL_CS node.

Referenced by `importFromERM()`, `importFromESRI()`, `importFromOzi()`, `importFromPanorama()`, `exportToPanorama()`, `importFromPCI()`, `exportToPCI()`, `importFromProj4()`, `importFromUSGS()`, `exportToUSGS()`, and `IsSame()`.

13.59.3.56 `int OGRSpatialReference::IsProjected () const`

Check if projected coordinate system.

This method is the same as the C function **OSRIsProjected()** (p. 560).

Returns

TRUE if this contains a PROJCS node indicating a it is a projected coordinate system.

References `OGR_SRSNode::GetValue()`, and `GetAttrNode()`.

Referenced by `importFromEPSGA()`, `AutoidentifyEPSG()`, `exportToERM()`, `importFromESRI()`, `importFromOzi()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, `importFromUSGS()`, `exportToXML()`, and `IsSame()`.

13.59.3.57 `int OGRSpatialReference::IsSame (const OGRSpatialReference * poOtherSRS) const`

Do these two spatial references describe the same system ?

Parameters

<i>poOtherSRS</i>	the SRS being compared to.
-------------------	----------------------------

Returns

TRUE if equivalent or FALSE otherwise.

References `IsSameGeogCS()`, `OGR_SRSNode::GetValue()`, `IsProjected()`, `GetAttrNode()`, `GetAttrValue()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetChild()`, `GetProjParm()`, `IsLocal()`, and `GetLinearUnits()`.

13.59.3.58 `int OGRSpatialReference::IsSameGeogCS (const OGRSpatialReference * poOther) const`

Do the GeogCS'es match?

This method is the same as the C function **OSRIsSameGeogCS()** (p. 561).

Parameters

<i>poOther</i>	the SRS being compared against.
----------------	---------------------------------

Returns

TRUE if they are the same or FALSE otherwise.

References `GetAttrValue()`, and `CPLAtof()`.

Referenced by `IsSame()`.

13.59.3.59 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.

This does the same as the C function **OSRMorphFromESRI()** (p. 561).

Returns

OGRErr_NONE unless something goes badly wrong.

References `OGR_SRSNode::applyRemapper()`, `GetAttrNode()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetValue()`, `OGR_SRSNode::SetValue()`, `GetAttrValue()`, `GetProjParm()`, `SetNode()`, `SetProjParm()`, and `FixupOrdering()`.

Referenced by `importFromESRI()`, and `SetFromUserInput()`.

13.59.3.60 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 not recognised by ESRI (like AUTHORITY and AXIS).

This does the same as the C function **OSRMorphToESRI()** (p. 561).

Returns

OGRErr_NONE unless something goes badly wrong.

References `Fixup()`, `StripCTParms()`, `GetAttrValue()`, `GetProjParm()`, `SetNode()`, `GetAttrNode()`, `FindProjParm()`, `OGR_SRSNode::DestroyChild()`, `OGR_SRSNode::applyRemapper()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetValue()`, `GetAuthorityName()`, `GetAuthorityCode()`, `OGR_SRSNode::SetValue()`, `GetUTMZone()`, `OGR_SRSNode::GetChildCount()`, `GetAngularUnits()`, `GetLinearUnits()`, `GetNormProjParm()`, and `OGR_SRSNode::AddChild()`.

13.59.3.61 int OGRSpatialReference::Reference ()

Increments the reference count by one.

The reference count is used keep track of the number of **OGRGeometry** (p. 151) objects referencing this SRS.

The method does the same thing as the C function **OSRReference()** (p. 561).

Returns

the updated reference count.

Referenced by OGRGeometry::assignSpatialReference().

13.59.3.62 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. 562).

References Dereference().

Referenced by OGRGeometry::assignSpatialReference().

13.59.3.63 OGRErr OGRSpatialReference::SetACEA (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Albers Conic Equal Area

References SetProjection(), and SetNormProjParm().

Referenced by importFromESRI(), importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.64 OGRErr OGRSpatialReference::SetAE (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Azimuthal Equidistant

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.65 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 GEO-GCS node.

This method does the same as the C function **OSRSetAngularUnits()** (p. 562).

Parameters

<i>pszUnits-Name</i>	the units name to be used. Some preferred units names can be found in ogr_srs_api.h (p. 544) such as SRS_UA_DEGREE.
<i>dfInRadians</i>	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. 544).

Returns

OGRERR_NONE on success.

References `GetAttrNode()`, `OGR_SRSNode::FindChild()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::SetValue()`, and `OGR_SRSNode::AddChild()`.

Referenced by `importFromPCI()`, and `Fixup()`.

13.59.3.66 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. 563).

Parameters

<i>pszTarget-Key</i>	the partial or complete path to the node to set an authority on. ie. "PROJCS", "GEOGCS" or "GEOGCS UNIT".
<i>pszAuthority</i>	authority name, such as "EPSG".
<i>nCode</i>	code for value with this authority.

Returns

OGRERR_NONE on success.

References `GetAttrNode()`, `OGR_SRSNode::FindChild()`, `OGR_SRSNode::DestroyChild()`, and `OGR_SRSNode::AddChild()`.

Referenced by `importFromEPSGA()`, `AutoidentifyEPSG()`, `importFromPanorama()`, `importFromPCI()`, `importFromUSGS()`, and `importFromWMSAUTO()`.

13.59.3.67 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

<i>pszTarget-Key</i>	either "PROJCS" or "GEOGCS", must already exist in SRS.
<i>pszXAxis-Name</i>	name of first axis, normally "Long" or "Easting".
<i>eXAxis-Orientation</i>	normally OAO_East.
<i>pszYAxis-Name</i>	name of second axis, normally "Lat" or "Northing".
<i>eYAxis-Orientation</i>	normally OAO_North.

Returns

OGRErr_NONE on success or an error code.

References `OGR_SRSNode::FindChild()`, `OGR_SRSNode::DestroyChild()`, `OGR_SRSNode::AddChild()`, and `OSRAxisEnumToName()`.

13.59.3.68 `OGRErr OGRSpatialReference::SetBonne (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)`

Bonne

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromProj4()`.

13.59.3.69 `OGRErr OGRSpatialReference::SetCEA (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)`

Cylindrical Equal Area

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromPanorama()`, and `importFromProj4()`.

13.59.3.70 `OGRErr OGRSpatialReference::SetCS (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)`

Cassini-Soldner

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromProj4()`.

13.59.3.71 OGRErr OGRSpatialReference::SetEC (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equidistant Conic

References SetProjection(), and SetNormProjParm().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.72 OGRErr OGRSpatialReference::SetEckert (int *nVariation*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert I-VI

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.73 OGRErr OGRSpatialReference::SetEquirectangular (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromProj4(), and importFromWMSAUT-O().

13.59.3.74 OGRErr OGRSpatialReference::SetEquirectangular2 (double *dfCenterLat*, double *dfCenterLong*, double *dfPseudoStdParallel1*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular generalized form :

References SetProjection(), and SetNormProjParm().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.75 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

<i>pszTargetKey</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszValue</i>	the value to set

Returns

OGRERR_NONE on success

References OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), OGR_SRSNode::SetValue(), and OGR_SRSNode::AddChild().

Referenced by importFromProj4().

13.59.3.76 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. 304).
2. "EPSG:n" - number passed on to **importFromEPSG()** (p. 292).
3. "EPSGA:n" - number passed on to **importFromEPSGA()** (p. 292).
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. 297).
7. filename - file read for WKT, XML or PROJ.4 definition.
8. well known name accepted by **SetWellKnownGeogCS()** (p. 327), 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. 307).

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. 564) function.

Parameters

<i>pszDefinition</i>	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 `importFromWkt()`, `morphFromESRI()`, `importFromEPSG()`, `importFromEPSGA()`, `importFromURN()`, `importFromWMSAUTO()`, `SetWellKnownGeogCS()`, `importFromDict()`, `Clear()`, `importFromProj4()`, `importFromUrl()`, and `importFromXML()`.

Referenced by `importFromUrl()`.

13.59.3.77 OGRErr OGRSpatialReference::SetGaussSchreiberTMercator (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromProj4()`.

13.59.3.78 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. 565).

Parameters

<i>pszGeog-Name</i>	user visible name for the geographic coordinate system (not to serve as a key).
<i>pszDatum-Name</i>	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.
<i>psz-Spheroid-Name</i>	user visible spheroid name (not to serve as a key)
<i>dfSemiMajor</i>	the semi major axis of the spheroid.
<i>dfInv-Flattening</i>	the inverse flattening for the spheroid. This can be computed from the semi minor axis as $1/f = 1.0 / (1.0 - \text{semiminor}/\text{semimajor})$.
<i>pszPMName</i>	the name of the prime meridian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.
<i>dfPMOffset</i>	the longitude of greenwich relative to this prime meridian.

<i>pszAngularUnits</i>	the angular units name (see ogr_srs_api.h (p. 544) for some standard names). If NULL a value of "degrees" will be assumed.
<i>dfConvertToRadians</i>	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 GetAttrNode(), Clear(), OGR_SRSNode::FindChild(), OGR_SRSNode::DestroyChild(), CPLAtof(), OGR_SRSNode::AddChild(), OGR_SRSNode::InsertChild(), and SetRoot().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.79 OGRErr OGRSpatialReference::SetGEOS (double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing*)

Geostationary Satellite

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.80 OGRErr OGRSpatialReference::SetGH (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Goode Homolosine

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.81 OGRErr OGRSpatialReference::SetGnomonic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gnomonic

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.82 OGRErr OGRSpatialReference::SetGS (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gall Stereographic

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.83 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.

This method does the same thing as the C function **OSRSetHOM()** (p. 566).

Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfCenter-Long</i>	Longitude of the projection origin.
<i>dfAzimuth</i>	Azimuth, measured clockwise from North, of the projection centerline.
<i>dfRectTo-Skew</i>	?
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalse-Easting</i>	False easting.
<i>dfFalse-Northing</i>	False northing.

Returns

OGRERR_NONE on success.

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4(), and importFromUSGS().

13.59.3.84 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. 566).

Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfLat1</i>	Latitude of the first point on center line.
<i>dfLong1</i>	Longitude of the first point on center line.
<i>dfLat2</i>	Latitude of the second point on center line.
<i>dfLong2</i>	Longitude of the second point on center line.
<i>dfScale</i>	Scale factor applies to the projection origin.

<i>dfFalse-Easting</i>	False easting.
<i>dfFalse-Northing</i>	False northing.

Returns

OGRERR_NONE on success.

References SetProjection(), and SetNormProjParm().

Referenced by importFromUSGS().

13.59.3.85 OGRErr OGRSpatialReference::SetWMPolyconic (double *dfLat1*, double *dfLat2*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

International Map of the World Polyconic

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), and importFromProj4().

13.59.3.86 OGRErr OGRSpatialReference::SetKrovak (double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfPseudoStdParallelLat*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.87 OGRErr OGRSpatialReference::SetLAEA (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.88 OGRErr OGRSpatialReference::SetLCC (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic

References SetProjection(), and SetNormProjParm().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.89 OGRErr OGRSpatialReference::SetLCC1SP (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.90 OGRErr OGRSpatialReference::SetLCCB (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References SetProjection(), and SetNormProjParm().

13.59.3.91 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 PROJ-CS or LOCAL_CS node.

This method does the same as the C function **OSRSetLinearUnits()** (p. 567).

Parameters

<i>pszUnits-Name</i>	the units name to be used. Some preferred units names can be found in ogr_srs_api.h (p. 544) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	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. 544).

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::FindChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::SetValue(), OGR_SRSNode::DestroyChild(), and OGR_SRSNode::AddChild().

Referenced by SetStatePlane(), importFromERM(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), SetLinearUnitsAndUpdateParameters(), importFromWMSAUTO(), SetUTM(), and Fixup().

13.59.3.92 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. 317) method, but it also updates all existing linear projection parameter values from the old units to the new units.

Parameters

<i>pszName</i>	the units name to be used. Some preferred units names can be found in ogr_srs_api.h (p. 544) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiply by a length in the indicated units to transform to meters. Some standard conversion factors can be found in ogr_srs_api.h (p. 544).

Returns

OGRERR_NONE on success.

References `GetLinearUnits()`, `GetAttrNode()`, `SetLinearUnits()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetValue()`, `GetProjParm()`, and `SetProjParm()`.

Referenced by `importFromESRI()`, and `importFromPCI()`.

13.59.3.93 OGRErr OGRSpatialReference::SetLocalCS (const char * *pszName*)

Set the user visible LOCAL_CS name.

This method is the same as the C function **OSRSetLocalCS()** (p. 568).

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. 317).

Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

Returns

OGRERR_NONE on success.

References `GetAttrNode()`, and `SetNode()`.

Referenced by `SetStatePlane()`, `importFromESRI()`, `importFromOzi()`, `importFromPanorama()`, `importFromPCI()`, and `importFromUSGS()`.

13.59.3.94 OGRErr OGRSpatialReference::SetMC (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Miller Cylindrical

References SetProjection(), and SetNormProjParm().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.95 OGRErr OGRSpatialReference::SetMercator (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mercator

References SetProjection(), and SetNormProjParm().

Referenced by importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.96 OGRErr OGRSpatialReference::SetMollweide (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mollweide

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

13.59.3.97 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 existence. 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. 562).

Parameters

<i>pszNode-Path</i>	full path to attribute to be set. For instance "PROJCS GEOGCS UNIT".
<i>pszNew-NodeValue</i>	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 SetRoot(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), OGR_SRSNode::AddChild(), and OGR_SRSNode::SetValue().

Referenced by morphToESRI(), morphFromESRI(), SetLocalCS(), SetProjCS(), SetProjection(), and SetUTM().

13.59.3.98 OGRErr OGRSpatialReference::SetNormProjParm (const char * *pszName*, double *dfValue*)

Set a projection parameter with a normalized value.

This method is the same as **SetProjParm()** (p. 322) 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. 569).

Parameters

<i>pszName</i>	the parameter name, which should be selected from the macros in ogr_srs_api.h (p. 544), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

Returns

OGRERR_NONE on success.

References SetProjParm().

Referenced by SetStatePlane(), importFromProj4(), SetTM(), SetTMVariant(), SetTM-SO(), SetTPED(), SetTMG(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), -SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGS(), SetGH(), SetGEOS(), SetGaussSchreiberTMercator(), SetGnomonic(), SetHOM(), SetHOM2P-NO(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCC-B(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOS(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetSO-C(), SetVDG(), SetUTM(), and SetWagner().

13.59.3.99 OGRErr OGRSpatialReference::SetNZMG (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

New Zealand Map Grid

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.100 OGRErr OGRSpatialReference::SetOrthographic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Orthographic

References SetProjection(), and SetNormProjParm().

Referenced by importFromPCI(), importFromProj4(), importFromUSGS(), and import-FromWMSAUTO().

13.59.3.101 OGRErr OGRSpatialReference::SetOS (double *dfOriginLat*, double *dfCMeridian*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Oblique Stereographic

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.102 OGRErr OGRSpatialReference::SetPolyconic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polyconic

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.103 OGRErr OGRSpatialReference::SetProjCS (const char * *pszName*)

Set the user visible PROJCS name.

This method is the same as the C function **OSRSetProjCS()** (p. 569).

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

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetValue(), SetNode(), and OGR_SRSNode::InsertChild().

13.59.3.104 OGRErr OGRSpatialReference::SetProjection (const char * *pszProjection*)

Set a projection name.

This method is the same as the C function **OSRSetProjection()** (p. 570).

Parameters

<i>psz-Projection</i>	the projection name, which should be selected from the macros in ogr_srs_api.h (p. 544), such as SRS_PT_TRANSVERSE_MERCATOR.
-----------------------	---

Returns

OGRERR_NONE on success.

References OGR_SRSNode::GetValue(), GetAttrNode(), SetNode(), and OGR_SRSNode::InsertChild().

Referenced by SetTM(), SetTMVariant(), SetTMSO(), SetTPED(), SetTMG(), - SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), - SetEquirectangular(), SetEquirectangular2(), SetGS(), SetGH(), SetGEOS(), SetGaussSchreiberTMercator(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetIWM-Polyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetM-C(), SetMercator(), SetMollweide(), SetNZMG(), SetOS(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetSOC(), SetVDG(), SetUTM(), and SetWagner().

13.59.3.105 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. 570).

Please check http://www.remotesensing.org/geotiff/proj_list pages for legal parameter names for specific projections.

Parameters

<i>pszParmName</i>	the parameter name, which should be selected from the macros in ogr_srs_api.h (p. 544), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetValue(), OGR_SRSNode::SetValue(), and OGR_SRSNode::AddChild().

Referenced by morphFromESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

13.59.3.106 OGRErr OGRSpatialReference::SetPS (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polar Stereographic

References SetProjection(), and SetNormProjParm().

Referenced by `importFromESRI()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

13.59.3.107 `OGRERR OGRSpatialReference::SetRobinson (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)`

Robinson

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

13.59.3.108 `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. 91) is assumed by the **OGRSpatialReference** (p. 268).

Parameters

<i>poNewRoot</i>	object to assign as root.
------------------	---------------------------

Referenced by `SetNode()`, `SetGeogCS()`, `CopyGeogCSFrom()`, `StripVertical()`, and `CloneGeogCS()`.

13.59.3.109 `OGRERR OGRSpatialReference::SetSinusoidal (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)`

Sinusoidal

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromOzi()`, `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

13.59.3.110 `OGRERR OGRSpatialReference::SetSOC (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)`

Swiss Oblique Cylindrical

References `SetProjection()`, and `SetNormProjParm()`.

13.59.3.111 `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().

Parameters

<i>nZone</i>	State plane zone number, in the USGS numbering scheme (as distinct from the Arc/Info and Erdas numbering scheme).
<i>bNAD83</i>	TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used.
<i>pszOverride-UnitName</i>	Linear unit name to apply overriding the legal definition for this zone.
<i>dfOverride-Unit</i>	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 accessible.

References Clear(), SetLocalCS(), SetLinearUnits(), CPLAtof(), importFromEPSG(), - GetLinearUnits(), GetNormProjParm(), SetNormProjParm(), GetAttrNode(), OGR_SRSNode::FindChild(), and OGR_SRSNode::DestroyChild().

Referenced by importFromESRI(), importFromPCI(), and importFromUSGS().

13.59.3.112 OGRErr OGRSpatialReference::SetStereographic (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Stereographic

References SetProjection(), and SetNormProjParm().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

13.59.3.113 OGRErr OGRSpatialReference::SetTM (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator

References SetProjection(), and SetNormProjParm().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

13.59.3.114 OGRErr OGRSpatialReference::SetTMG (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Tunesia Mining Grid

References SetProjection(), and SetNormProjParm().

13.59.3.115 OGRErr OGRSpatialReference::SetTMSO (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References SetProjection(), and SetNormProjParm().

13.59.3.116 OGRErr OGRSpatialReference::SetTMVariant (const char * *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator variants.

References SetProjection(), and SetNormProjParm().

13.59.3.117 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.268) 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.572).

Parameters

<i>dfDX</i>	X child in meters.
<i>dfDY</i>	Y child in meters.
<i>dfDZ</i>	Z child in meters.
<i>dfEX</i>	X rotation in arc seconds (optional, defaults to zero).
<i>dfEY</i>	Y rotation in arc seconds (optional, defaults to zero).
<i>dfEZ</i>	Z rotation in arc seconds (optional, defaults to zero).
<i>dfPPM</i>	scaling factor (parts per million).

Returns

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::FindChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::AddChild(), and OGR_SRSNode::InsertChild().

Referenced by importFromProj4().

13.59.3.118 OGRErr OGRSpatialReference::SetTPED (double *dfLat1*, double *dfLong1*, double *dfLat2*, double *dfLong2*, double *dfFalseEasting*, double *dfFalseNorthing*)

Two Point Equidistant

References SetProjection(), and SetNormProjParm().

Referenced by importFromProj4().

13.59.3.119 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. 572).

Parameters

<i>nZone</i>	UTM zone.
<i>bNorth</i>	TRUE for northern hemisphere, or FALSE for southern hemisphere.

Returns

OGRERR_NONE on success.

References SetProjection(), SetNormProjParm(), GetAttrValue(), SetNode(), and SetLinearUnits().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

13.59.3.120 OGRErr OGRSpatialReference::SetVDG (double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

VanDerGrinten

References SetProjection(), and SetNormProjParm().

Referenced by `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

13.59.3.121 `OGRERR OGRSpatialReference::SetWagner (int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)`

Wagner I -- VII

References `SetProjection()`, and `SetNormProjParm()`.

Referenced by `importFromPanorama()`, `importFromProj4()`, and `importFromUSGS()`.

13.59.3.122 `OGRERR OGRSpatialReference::SetWellKnownGeogCS (const char * pszName)`

Set a GeogCS based on well known name.

This may be called on an empty **OGRSpatialReference** (p. 268) 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

<i>pszName</i>	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 `importFromEPSG()`, `IsGeographic()`, `CopyGeogCSFrom()`, `importFromEPSGA()`, and `importFromWkt()`.

Referenced by `importFromESRI()`, `importFromOzi()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, `importFromUSGS()`, `SetFromUserInput()`, `importFromURN()`, and `importFromWMSAUTO()`.

13.59.3.123 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. 573).

Parameters

<i>poCurrent</i>	node to operate on. NULL to operate on whole tree.
------------------	--

Returns

OGRERR_NONE on success or an error code.

References StripVertical(), OGR_SRSNode::GetValue(), and OGR_SRSNode::StripNodes().

Referenced by morphToESRI().

13.59.3.124 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.

References OGR_SRSNode::GetChild(), OGR_SRSNode::Clone(), and SetRoot().

Referenced by StripCTParms().

13.59.3.125 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. 573).

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.

References OGR_SRSNode::GetValue(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetChild(), CPLAtof(), and OGR_SRSNode::GetNode().

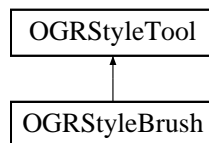
The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogr_fromepsg.cpp
- ogr_srs_dict.cpp
- ogr_srs_erm.cpp
- ogr_srs_esri.cpp
- ogr_srs_ozl.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

13.60 OGRStyleBrush Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleBrush:



13.60.1 Detailed Description

This class represents a style brush

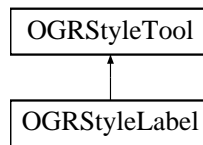
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

13.61 OGRStyleLabel Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleLabel:



13.61.1 Detailed Description

This class represents a style label

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

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

13.62 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.
- GBool **AddPart** (**OGRStyleTool** *)
Add a part (style tool) 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.

13.62.1 Detailed Description

This class represents a style manager

13.62.2 Constructor & Destructor Documentation

13.62.2.1 OGRStyleMgr::OGRStyleMgr (OGRStyleTable * *poDataSetStyleTable* = NULL)

Constructor.

This method is the same as the C function **OGR_SM_Create()** (p. 521)

Parameters

<i>poDataSetStyleTable</i>	(currently unused, reserved for future use), pointer to OGRStyleTable (p. 334). Pass NULL for now.
----------------------------	---

13.62.2.2 OGRStyleMgr::~OGRStyleMgr ()

Destructor.

This method is the same as the C function **OGR_SM_Destroy()** (p. 521)

13.62.3 Member Function Documentation

13.62.3.1 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. 520).

Parameters

<i>poStyleTool</i>	the style tool defining the part to add.
--------------------	--

Returns

TRUE on success, FALSE on errors.

13.62.3.2 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. 521).

Parameters

<i>nPartId</i>	the part number (0-based index).
<i>pszStyle-String</i>	(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. 338) of the requested part (style tools) or NULL on error.

13.62.3.3 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. 522).

Parameters

<i>pszStyle-String</i>	(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.

13.62.3.4 const char * OGRStyleMgr::GetStyleName (const char * *pszStyleString* = NULL)

Get the name of a style from the style table.

Parameters

<i>pszStyle-String</i>	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().

13.62.3.5 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. 522).

Parameters

<i>poFeature</i>	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 InitStyleString(), and OGRFeature::GetStyleString().

13.62.3.6 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. 522).

Parameters

<i>pszStyle-String</i>	the style string to use (can be NULL).
------------------------	--

Returns

TRUE on success, FALSE on errors.

Referenced by InitFromFeature().

13.62.3.7 GBool OGRStyleMgr::SetFeatureStyleString (OGRFeature * *poFeature*, const char * *pszStyleString* = NULL, GBool *bNoMatching* = FALSE)

Set a style in a feature.

Parameters

<i>poFeature</i>	the feature object to store the style in
<i>pszStyle-String</i>	the style to store
<i>bNo-Matching</i>	TRUE to lookup the style in the style table and add the name to the feature

Returns

TRUE on success, FALSE on error.

References OGRFeature::SetStyleString(), and GetStyleName().

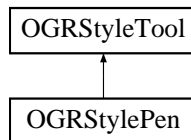
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- **ogrfeaturestyle.cpp**

13.63 OGRStylePen Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStylePen:



13.63.1 Detailed Description

This class represents a style pen

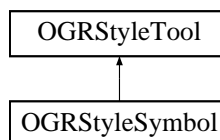
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

13.64 OGRStyleSymbol Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleSymbol:



13.64.1 Detailed Description

This class represents a style symbol

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

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

13.65 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.

13.65.1 Detailed Description

This class represents a style table

13.65.2 Member Function Documentation

13.65.2.1 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

<i>pszName</i>	the name the style to add.
<i>pszStyle-String</i>	the style string to add.

Returns

TRUE on success, FALSE on error

References `IsExist()`.

Referenced by `ModifyStyle()`.

13.65.2.2 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. 334).

Returns

new style table, exactly matching this style table.

Referenced by `OGRLayer::SetStyleTable()`, and `OGRDataSource::SetStyleTable()`.

13.65.2.3 const char * OGRStyleTable::Find (const char * pszName)

Get a style string by name.

Parameters

<i>pszName</i>	the name of the style string to find.
----------------	---------------------------------------

Returns

the style string matching the name, NULL if not found or error.

References `IsExist()`.

13.65.2.4 const char * OGRStyleTable::GetStyleName (const char * pszStyleString)

Get style name by style string.

Parameters

<i>pszStyle-String</i>	the style string to look up.
------------------------	------------------------------

Returns

the Name of the matching style string or NULL on error.

Referenced by `OGRStyleMgr::GetStyleName()`.

13.65.2.5 int OGRStyleTable::IsExist (const char * *pszName*)

Get the index of a style in the table by its name.

Parameters

<i>pszName</i>	the name to look for.
----------------	-----------------------

Returns

The index of the style if found, -1 if not found or error.

Referenced by AddStyle(), RemoveStyle(), and Find().

13.65.2.6 GBool OGRStyleTable::LoadStyleTable (const char * *pszFilename*)

Load a style table from a file.

Parameters

<i>pszFilename</i>	the name of the file to load from.
--------------------	------------------------------------

Returns

TRUE on success, FALSE on error

13.65.2.7 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

<i>pszName</i>	the name of the style to modify.
<i>pszStyleString</i>	the style string.

Returns

TRUE on success, FALSE on error

References RemoveStyle(), and AddStyle().

13.65.2.8 void OGRStyleTable::Print (FILE * *fpOut*)

Print a style table to a FILE pointer.

Parameters

<i>fpOut</i>	the FILE pointer to print to.
--------------	-------------------------------

13.65.2.9 GBool OGRStyleTable::RemoveStyle (const char * *pszName*)

Remove a style in the table by its name.

Parameters

<i>pszName</i>	the name of the style to remove.
----------------	----------------------------------

Returns

TRUE on success, FALSE on error

References `IsExist()`.

Referenced by `ModifyStyle()`.

13.65.2.10 GBool OGRStyleTable::SaveStyleTable (const char * *pszFilename*)

Save a style table to a file.

Parameters

<i>pszFilename</i>	the name of the file to save to.
--------------------	----------------------------------

Returns

TRUE on success, FALSE on error

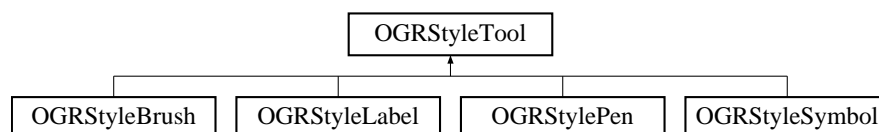
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- `ogrfeaturestyle.cpp`

13.66 OGRStyleTool Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleTool:



13.66.1 Detailed Description

This class represents a style tool

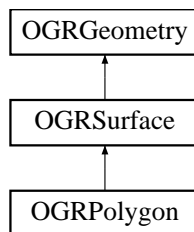
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

13.67 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 **Centroid** (OGRPoint *poPoint) const =0
Compute and return centroid of surface. The centroid is not necessarily within the geometry.
- virtual OGRErr **PointOnSurface** (OGRPoint *poPoint) const =0
This method relates to the SFCOM ISurface::get_PointOnSurface() method.

13.67.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons.

13.67.2 Member Function Documentation

13.67.2.1 OGRErr OGRSurface::Centroid (OGRPoint * poPoint) const [pure virtual]

Compute and return centroid of surface. The centroid is not necessarily within the geometry.

This method relates to the SFCOM ISurface::get_Centroid() method.

NOTE: Only implemented when GEOS included in build.

Parameters

<i>poPoint</i>	point to be set with the centroid location.
----------------	---

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implemented in **OGRPolygon** (p. 248).

13.67.2.2 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. 251).

13.67.2.3 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

<i>poPoint</i>	point to be set with an internal point.
----------------	---

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implemented in **OGRPolygon** (p. 255).

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

- **ogr_geometry.h**
- **ogrsurface.cpp**

13.68 OZIDatums Struct Reference

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

- ogr_srs_ozl.cpp

13.69 ParseContext Struct Reference

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

- cpl_minixml.cpp

13.70 PCIDatums Struct Reference

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

- ogr_srs_pci.cpp

13.71 projUV Struct Reference

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

- ogrct.cpp

13.72 StackContext Struct Reference

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

- cpl_minixml.cpp

13.73 swq_col_def Struct Reference

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

- swq.h

13.74 swq_field_list Struct Reference

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

- `swq.h`

13.75 `swq_field_op` Struct Reference

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

- `swq.h`

13.76 `swq_join_def` Struct Reference

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

- `swq.h`

13.77 `swq_order_def` Struct Reference

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

- `swq.h`

13.78 `swq_select` Struct Reference

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

- `swq.h`

13.79 `swq_summary` Struct Reference

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

- `swq.h`

13.80 `swq_table_def` Struct Reference

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

- `swq.h`

13.81 tm_unz_s Struct Reference

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

- cpl_minizip_unzip.h

13.82 unz_file_info_internal_s Struct Reference

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

- cpl_minizip_unzip.cpp

13.83 unz_file_info_s Struct Reference

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

- cpl_minizip_unzip.h

13.84 unz_file_pos_s Struct Reference

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

- cpl_minizip_unzip.h

13.85 unz_global_info_s Struct Reference

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

- cpl_minizip_unzip.h

13.86 unz_s Struct Reference

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

- cpl_minizip_unzip.cpp

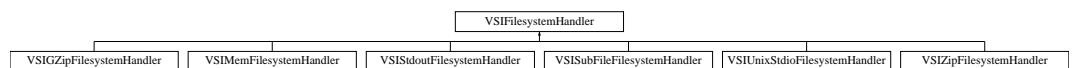
13.87 VSIFileManager Class Reference

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

- `cpl_vsi_virtual.h`
- `cpl_vsil.cpp`

13.88 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:

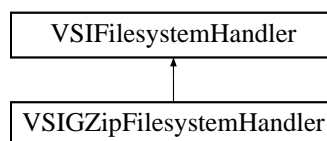


The documentation for this class was generated from the following file:

- `cpl_vsi_virtual.h`

13.89 VSIGZipFilesystemHandler Class Reference

Inheritance diagram for VSIGZipFilesystemHandler:

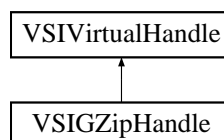


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

13.90 VSIGZipHandle Class Reference

Inheritance diagram for VSIGZipHandle:

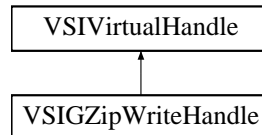


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

13.91 VSIGZipWriteHandle Class Reference

Inheritance diagram for VSIGZipWriteHandle:



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

- cpl_vsil_gzip.cpp

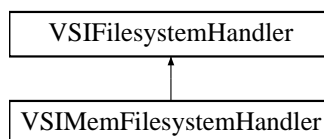
13.92 VSIMemFile Class Reference

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

- cpl_vsi_mem.cpp

13.93 VSIMemFilesystemHandler Class Reference

Inheritance diagram for VSIMemFilesystemHandler:

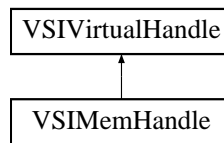


The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

13.94 VSIMemHandle Class Reference

Inheritance diagram for VSIMemHandle:

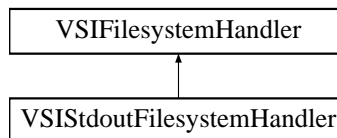


The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

13.95 VSIStdoutFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutFilesystemHandler:

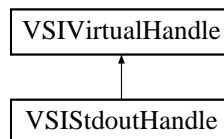


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

13.96 VSIStdoutHandle Class Reference

Inheritance diagram for VSIStdoutHandle:

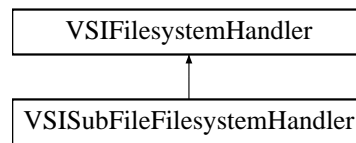


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

13.97 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

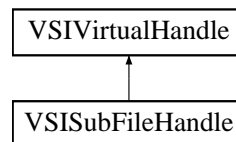


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

13.98 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:

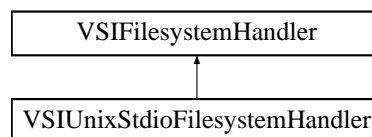


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

13.99 VSIUnixStdioFilesystemHandler Class Reference

Inheritance diagram for VSIUnixStdioFilesystemHandler:

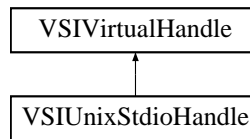


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

13.100 VSIUnixStdioHandle Class Reference

Inheritance diagram for VSIUnixStdioHandle:

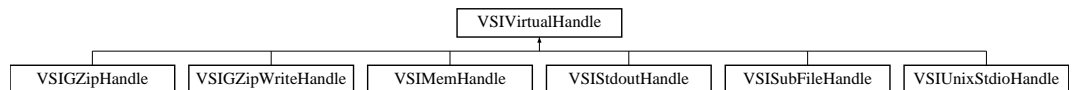


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

13.101 VSIVirtualHandle Class Reference

Inheritance diagram for VSIVirtualHandle:

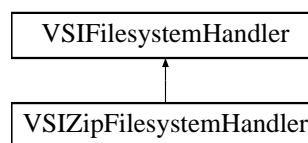


The documentation for this class was generated from the following file:

- cpl_vsi_virtual.h

13.102 VSIZipFilesystemHandler Class Reference

Inheritance diagram for VSIZipFilesystemHandler:



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

- cpl_vsil_gzip.cpp

13.103 ZIPContent Struct Reference

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

- cpl_vsil_gzip.cpp

13.104 ZIPEntry Struct Reference

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

- `cpl_vsil_gzip.cpp`

13.105 zlib_filefunc_def_s Struct Reference

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

- `cpl_minizip_ioapi.h`

Chapter 14

File Documentation

14.1 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** (FILE *)
- const char * **CPLReadLine2L** (FILE *, 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 * **CPLFormCifFilename** (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 *)

14.1.1 Detailed Description

Various convenience functions for CPL.

14.1.2 Function Documentation

14.1.2.1 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. 354) function if you want to specify custom delimiter.

IMPORTANT NOTE. Existence 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. 353), because such files distributed across the systems with different locales and floating point representation should 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

<i>nptr</i>	Pointer to string to convert.
-------------	-------------------------------

Returns

Converted value, if any.

References `CPLAtof()`, and `CPLStrtod()`.

Referenced by `CPLScanDouble()`, `CPLAtof()`, `OGRSpatialReference::SetStatePlane()`, `OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromProj4()`, `OGRSpatialReference::exportToProj4()`, `OGRSpatialReference::Validate()`, `OGRSpatialReference::GetAngularUnits()`, `OGRSpatialReference::GetLinearUnits()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::SetGeogCS()`, `OGRSpatialReference::importFromWMSAUTO()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetProjParm()`, `OGRSpatialReference::IsSameGeogCS()`, `OGRSpatialReference::GetTOWGS84()`, and `OGRSpatialReference::Fixup()`.

14.1.2.2 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. 353) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

References CPLAtofDelim(), and CPLStrtodDelim().

Referenced by CPLAtofDelim().

14.1.2.3 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

<i>nptr</i>	The string to convert.
-------------	------------------------

Returns

Converted value, if any. Zero on failure.

References CPLAtofM(), and CPLStrtodDelim().

Referenced by CPLAtofM(), and OGRSpatialReference::importFromProj4().

14.1.2.4 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 **CPL**`Error()` (p.380) 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 `VSIAlloc()` to get the memory, so any hooking of `VSIAlloc()` will apply to **CPL**`Calloc()` (p.354) as well. `CPLFree()` or `VSIFree()` can be used free memory allocated by **CPL**`Calloc()` (p.354).

Parameters

<i>nCount</i>	number of objects to allocate.
<i>nSize</i>	size (in bytes) of object to allocate.

Returns

pointer to newly allocated memory, only NULL if `nSize * nCount` is NULL.

14.1.2.5 `int CPLCheckForFile (char * pszFilename, char ** papszSiblingFiles)`

Check for file existence.

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 **VSI**`StatL()` (p.433) is used to test for the files existence, and no case insensitive testing is done.

Parameters

<i>pszFilename</i>	name of file to check for - filename case updated in some cases.
<i>papsz-SiblingFiles</i>	a list of files in the same directory as <code>pszFilename</code> if available, or NULL. This list should have no path components.

Returns

TRUE if a match is found, or FALSE if not.

References `VSIStatL()`, and `CPLGetFilename()`.

14.1.2.6 `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

<i>pszPath</i>	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().

14.1.2.7 void CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A **CPL****Error()** (p. 380) is issued if the file is not in the shared file list.

Parameters

<i>fp</i>	file handle from CPL OpenShared() (p. 366) to deaccess.
-----------	---

References VSIFCloseL().

14.1.2.8 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 **CSL-Destroy()** (p. 416).

Parameters

<i>pszOld- Filename</i>	path to old prototype file.
<i>pszNew- Filename</i>	path to new prototype file.
<i>papszFile- List</i>	list of other files associated with pszOldFilename to rename similarly.

Returns

a list of files corresponding to papszFileList but renamed to correspond to pszNew-Filename.

References CPLCorrespondingPaths(), CPLGetPath(), CPLGetBasename(), CPLGet-Filename(), and CPLFormFilename().

Referenced by CPLCorrespondingPaths().

14.1.2.9 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. 367).

Parameters

<i>dfDec</i>	Angle in decimal degrees.
--------------	---------------------------

Returns

Angle in packed DMS format.

14.1.2.10 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. 380) call.

Parameters

<i>fp</i>	File handle to write to.
-----------	--------------------------

14.1.2.11 `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

<i>pszBaseDir</i>	the name of the directory relative to which the path should be computed. <i>pszBaseDir</i> may be NULL in which case the original target is returned without relativizing.
<i>pszTarget</i>	the filename to be changed to be relative to <i>pszBaseDir</i> .
<i>pbGotRelative</i>	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()`.

14.1.2.12 `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. 358): single `'\r'` and `'\n'` and `'\r\n'` combination.

Parameters

<i>pszBuffer</i>	pointer to the targeting character buffer.
<i>nBufferSize</i>	maximum size of the string to read (not including terminating <code>'\0'</code>).
<i>fp</i>	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.

14.1.2.13 `const char* CPLFormCIFilename (const char * pszPath, const char * pszBasename,
const char * pszExtension)`

Case insensitive file searching, returning 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 equivalent to **CPLFormFilename()** (p. 359) on case insensitive file systems (like Windows).

Parameters

<i>pszPath</i>	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.
<i>psz-Basename</i>	file basename. May optionally have path and/or extension. May not be NULL.
<i>psz-Extension</i>	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()**, and **VSISatL()**.

Referenced by **CPLFormCIFilename()**.

14.1.2.14 `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

<i>pszPath</i>	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.
<i>psz-Basename</i>	file basename. May optionally have path and/or extension. May not be NULL.
<i>psz-Extension</i>	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 CPLUnlinkTree(), CPLFormFilename(), CPLFormCIFilename(), CPLCorrespondingPaths(), CPLGenerateTempFilename(), and OGRSFDriverRegistrar::AutoLoadDrivers().

14.1.2.15 `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

<i>pszStem</i>	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 CPLGenerateTempFilename(), and CPLFormFilename().

Referenced by CPLGenerateTempFilename().

14.1.2.16 `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

<i>pszFull-Filename</i>	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 CPLGetBasename(), CPLCorrespondingPaths(), and OGRSFDriver-Registrar::AutoLoadDrivers().

14.1.2.17 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. 376). If the given option was no defined with **CPLSetConfigOption()** (p. 376), it tries to find it in environment variables.

Parameters

<i>pszKey</i>	the key of the option to retrieve
<i>pszDefault</i>	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. 376)

14.1.2.18 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().

14.1.2.19 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. 364).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters

<i>pszFilename</i>	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 CPLGetDirname(), and OGRSFDriverRegistrar::AutoLoadDrivers().

14.1.2.20 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 platform.

Parameters

<i>pszPathBuf</i>	the buffer into which the path is placed.
<i>nMaxLength</i>	the buffer size, MAX_PATH+1 is suggested.

Returns

FALSE on failure or TRUE on success.

References CPLGetExecPath().

Referenced by CPLGetExecPath(), and OGRSFDriverRegistrar::AutoLoadDrivers().

14.1.2.21 const char* CPLGetExtension (const char * *pszFullFilename*)

Extract filename extension from full filename.

Returns a string containing the extension 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

<i>pszFull-Filename</i>	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 CPLGetExtension(), and OGRSFDriverRegistrar::AutoLoadDrivers().

14.1.2.22 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

<i>pszFull-Filename</i>	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(), CPLGetFilename(), and CPLCorrespondingPaths().

14.1.2.23 `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

<i>pszFilename</i>	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 CPLGetPath(), and CPLCorrespondingPaths().

14.1.2.24 `CPLSharedFileInfo* CPLGetSharedList (int * pnCount)`

Fetch list of open shared files.

Parameters

<i>pnCount</i>	place to put the count of entries.
----------------	------------------------------------

Returns

the pointer to the first in the array of shared file info structures.

14.1.2.25 `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. 380) or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. 364) 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-pending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters

<i>pszLibrary</i>	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.
<i>pszSymbol-Name</i>	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 **CPLGetSymbol()**, and **OGRSFDriverRegistrar::AutoLoadDrivers()**.

14.1.2.26 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

<i>pszFilename</i>	the filename with path to test.
--------------------	---------------------------------

Returns

TRUE if the filename is relative or FALSE if it is absolute.

References **CPLIsFilenameRelative()**.

Referenced by **CPLProjectRelativeFilename()**, **CPLIsFilenameRelative()**, and **CPL-ExtractRelativePath()**.

14.1.2.27 void* CPLMalloc (size_t nSize)

Safe version of malloc().

This function is like the C library malloc(), but raises a CE_Fatal error with **CPL_Error()** (p. 380) 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. 366) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLMalloc()** (p. 366).

Parameters

<i>nSize</i>	size (in bytes) of memory block to allocate.
--------------	--

Returns

pointer to newly allocated memory, only NULL if nSize is zero.

14.1.2.28 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 VSIOpen() or **VSIOpenL()** (p. 425) function is used to actually open the file, when an existing file handle can't be shared.

Parameters

<i>pszFilename</i>	the name of the file to open.
<i>pszAccess</i>	the normal fopen()/VSIOpen() style access string.
<i>bLarge</i>	If TRUE VSIOpenL() (p. 425) (for large files) will be used instead of VSIOpen().

Returns

a file handle or NULL if opening fails.

References VSIOpenL().

14.1.2.29 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.
2. The degrees are separated out: deg = ang/1000000 (fractional portion truncated)
3. The minutes are separated out: min = (ang - deg * 1000000) / 1000 (fractional portion truncated)
4. The seconds are then computed: sec = ang - deg * 1000000 - min * 1000
5. The total angle in seconds is computed: sec = deg * 3600.0 + min * 60.0 + sec
6. 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

<i>dfPacked</i>	Angle in packed DMS format.
-----------------	-----------------------------

Returns

Angle in decimal degrees.

14.1.2.30 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

<i>pszBuffer</i>	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.
<i>pszFormat</i>	Format specifier (for example, "%16.9E").
<i>dfValue</i>	Numerical value to print.
<i>pszLocale</i>	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 <i>pszLocale</i> 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.

14.1.2.31 int CPLPrintInt32 (char * *pszBuffer*, GInt32 *iValue*, int *nMaxLen*)

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	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.
<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

14.1.2.32 int CPLPrintPointer (char * *pszBuffer*, void * *pValue*, int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	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.
<i>pValue</i>	Pointer to ASCII encode.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

14.1.2.33 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

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

14.1.2.34 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

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

14.1.2.35 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

<i>pszBuffer</i>	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.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.
<i>pszFormat</i>	Controls the output format. Options are the same as for strftime(3) function.
<i>poBroken-Time</i>	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.
<i>pszLocale</i>	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 adjusted differently from the C/POSIX defaults. To solve these problems this option was introduced.

Returns

Number of characters printed.

14.1.2.36 int CPLPrintUIntBig (char * *pszBuffer*, GUIntBig *iValue*, int *nMaxLen*)

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	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.
<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

14.1.2.37 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

<i>pszProject-Dir</i>	the directory relative to which the secondary files path should be interpreted.
<i>psz-Secondary-Filename</i>	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 `CPLProjectRelativeFilename()`, and `CPLIsFilenameRelative()`.

Referenced by `CPLProjectRelativeFilename()`.

14.1.2.38 `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 ... equivalent 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. 371).

Note that **`CPLReadLine()`** (p. 371) uses `VSIFGets()`, so any hooking of VSI file services should apply to **`CPLReadLine()`** (p. 371) as well.

`CPLReadLine()` (p. 371) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **`CPLReadLine()`** (p. 371) may be called with a `NULL FILE *` at any time to free this working buffer.

Parameters

<i>fp</i>	file pointer opened with <code>VSIFOpen()</code> .
-----------	--

Returns

pointer to an internal buffer containing a line of text read from the file or `NULL` if the end of file was encountered.

14.1.2.39 `const char* CPLReadLine2L (FILE * fp, int nMaxCars, char ** papszOptions)`

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. 371), but reading from a large file API handle.

Parameters

<i>fp</i>	file pointer opened with VSIFOpenL() (p. 425).
<i>nMaxCars</i>	maximum number of characters allowed, or -1 for no limit.
<i>papsz-Options</i>	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 reached.

Since

GDAL 1.7.0

References **VSIFReadL()**, **VSIFSeekL()**, and **VSIFTellL()**.

14.1.2.40 `const char* CPLReadLineL (FILE * fp)`

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. 371), but reading from a large file API handle.

Parameters

<i>fp</i>	file pointer opened with VSIFOpenL() (p. 425).
-----------	---

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

14.1.2.41 `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. 380) 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 **VSIFRealloc()** to get the memory, so any hooking of

VSIRealloc() will apply to **CPLRealloc()** (p. 372) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLRealloc()** (p. 372).

It is also safe to pass NULL in as the existing memory block for **CPLRealloc()** (p. 372), in which case it uses VSIMalloc() to allocate a new block.

Parameters

<i>pData</i>	existing memory block which should be copied to the new block.
<i>nNewSize</i>	new size (in bytes) of memory block to allocate.

Returns

pointer to allocated memory, only NULL if nNewSize is zero.

14.1.2.42 const char* CPLResetExtension (const char * *pszPath*, const char * *pszExt*)

Replace the extension with the provided one.

Parameters

<i>pszPath</i>	the input path, this string is not altered.
<i>pszExt</i>	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().

14.1.2.43 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. 353) to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	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().

14.1.2.44 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

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	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.

14.1.2.45 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

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	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.

14.1.2.46 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

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to read. Less characters will be read if a null character is encountered.
<i>bTrimSpaces</i>	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
<i>bNormalize</i>	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 CPL-Free().

14.1.2.47 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

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	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.

14.1.2.48 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

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	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.

14.1.2.49 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. 361) method.

This mechanism is similar to environment variables, but options set with **CPLSetConfigOption()** (p. 376) overrides, for **CPLGetConfigOption()** (p. 361) point of view, values defined in the environment.

If **CPLSetConfigOption()** (p. 376) 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

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option

14.1.2.50 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. 361) method.

This function sets the configuration option that only applies in the current thread, as opposed to **CPLSetConfigOption()** (p. 376) which sets an option that applies on all threads.

Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option

14.1.2.51 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. 380) instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to **CPLStrdup()** (p. 376) as well.

Memory allocated with **CPLStrdup()** (p. 376) can be freed with **CPLFree()** or **VSIFree()**.

It is also safe to pass a NULL string into **CPLStrdup()** (p. 376). **CPLStrdup()** (p. 376) will allocate and return a zero length string (as opposed to a NULL string).

Parameters

<i>pszString</i>	input string to be duplicated. May be NULL.
------------------	---

Returns

pointer to a newly allocated copy of the string. Free with **CPLFree()** or **VSIFree()**.

14.1.2.52 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

<i>pszString</i>	input string to be converted.
------------------	-------------------------------

Returns

pointer to the same string, *pszString*.

14.1.2.53 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. 378) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 353) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

Returns

Converted value, if any.

References **CPLStrtod()**, and **CPLStrtodDelim()**.

Referenced by **CPLAtof()**, and **CPLStrtod()**.

14.1.2.54 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. 353) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

References `CPLStrtodDelim()`.

Referenced by `CPLAtofDelim()`, `CPLAtofM()`, `CPLStrtodDelim()`, `CPLStrtod()`, and `CPLStrtofDelim()`.

14.1.2.55 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. 379) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 353) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

Returns

Converted value, if any.

References `CPLStrtof()`, and `CPLStrtofDelim()`.

Referenced by `CPLStrtof()`.

14.1.2.56 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. 353) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

References `CPLStrtofDelim()`, and `CPLStrtodDelim()`.

Referenced by `CPLStrtofDelim()`, and `CPLStrtof()`.

14.1.2.57 int CPLUnlinkTree (const char * *pszPath*)

Returns

0 on successful completion, -1 if function fails.

References `VSIUnlink()`, `CPLFormFilename()`, and `VSIrmdir()`.

14.2 cpl_error.h File Reference

```
#include "cpl_port.h"
```

Functions

- void **CPLError** (CPLerr eErrClass, int err_no, const char *fmt,...)
- void **CPLErrorReset** (void)
- int **CPLGetLastErrorNo** (void)
- CPLerr **CPLGetLastErrorType** (void)
- const char * **CPLGetLastErrorMsg** (void)
- CPLErrorHandler **CPLSetErrorHandler** (CPLErrorHandler)
- void **CPLPushErrorHandler** (CPLErrorHandler)
- void **CPLPopErrorHandler** (void)
- void **CPLDebug** (const char *, const char *,...)
- void **_CPLAssert** (const char *, const char *, int)

14.2.1 Detailed Description

CPL error handling services.

14.2.2 Function Documentation

14.2.2.1 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. 380) only if the condition fails. **_CPLAssert()** (p. 380) will generate a CE_Fatal error call to **CPLError()** (p. 380), 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. 380) directly.

14.2.2.2 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 determined 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

<i>pszCategory</i>	name of the debugging message category.
<i>pszFormat</i>	printf() style format string for message to display. Remaining arguments are assumed to be for format.

14.2.2.3 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. 382).

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 occurred, and that **CPL_Error()** (p. 380) should not return.

The default behaviour of **CPL_Error()** (p. 380) is to report errors to `stderr`, and to `abort()` after reporting a `CE_Fatal` error. It is expected that some applications will want to suppress 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 **CPL_GetLastErrorNo()** (p. 381) and **CPL_GetLastErrorMsg()** (p. 381).

Parameters

<i>eErrClass</i>	one of <code>CE_Warning</code> , <code>CE_Failure</code> or <code>CE_Fatal</code> .
<i>err_no</i>	the error number (<code>CPL_E_*</code>) from cpl_error.h (p. 379).
<i>fmt</i>	a <code>printf()</code> style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to <code>printf()</code> .

14.2.2.4 void CPL_ErrorReset (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.

14.2.2.5 const char* CPL_GetLastErrorMsg (void)

Get the last error message.

Fetches the last error message posted with **CPL_Error()** (p. 380), that hasn't been cleared by **CPL_ErrorReset()** (p. 381). 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.

14.2.2.6 int CPL_GetLastErrorNo (void)

Fetch the last error number.

This is the error number, not the error class.

Returns

the error number of the last error to occur, or `CPL_E_None` (0) if there are no posted errors.

14.2.2.7 CPLerr CPLGetLastErrorType (void)

Fetch the last error type.

This is the error class, not the error number.

Returns

the error number of the last error to occur, or CE_None (0) if there are no posted errors.

14.2.2.8 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. 382) call. This method has no effect if there are no error handlers on the current threads error handler stack.

14.2.2.9 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. 382).

The **CPLSetErrorHandler()** (p. 382) docs have further information on how CPLError handlers work.

Parameters

<i>pfnError-HandlerNew</i>	new error handler function.
----------------------------	-----------------------------

14.2.2.10 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 **CPL_Error()** (p. 380) before calling the handler. If the error handler function is passed a CE_Fatal

class error and returns, then **CPL_Error()** (p. 380) 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 **CPL_SetErrorHandler()** (p. 382) 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 precedence over the global error handlers set with **CPL_SetErrorHandler()** (p. 382). Generally speaking **CPL_SetErrorHandler()** (p. 382) would be used to set a desired global error handler, while **CPLPushErrorHandler()** (p. 382) would be used to install a temporary local error handler, such as `CPLQuietErrorHandler()` to suppress error reporting in a limited segment of code.

Parameters

<i>pfnError-HandlerNew</i>	new error handler function.
----------------------------	-----------------------------

Returns

returns the previously installed error handler.

14.3 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`)

14.3.1 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.

14.3.2 Function Documentation

14.3.2.1 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

<i>set</i>	the hash set
------------	--------------

References `_CPLList::pData`, and `_CPLList::psNext`.

14.3.2.2 int CPLHashSetEqualPointer (const void * *elt1*, const void * *elt2*)

Equality function for arbitrary pointers

Parameters

<i>elt1</i>	the first arbitrary pointer to compare
<i>elt2</i>	the second arbitrary pointer to compare

Returns

TRUE if the pointers are equal

14.3.2.3 int CPLHashSetEqualStr (const void * *elt1*, const void * *elt2*)

Equality function for strings

Parameters

<i>elt1</i>	the first string to compare. May be NULL.
<i>elt2</i>	the second string to compare. May be NULL.

Returns

TRUE if the strings are equal

14.3.2.4 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

<i>set</i>	the hash set.
<i>fnIterFunc</i>	the function called on each element.
<i>user_data</i>	the user data provided to the function.

References _CPLList::pData, and _CPLList::psNext.

14.3.2.5 unsigned long CPLHashSetHashPointer (const void * *elt*)

Hash function for an arbitrary pointer

Parameters

<i>elt</i>	the arbitrary pointer to hash
------------	-------------------------------

Returns

the hash value of the pointer

14.3.2.6 unsigned long CPLHashSetHashStr (const void * *elt*)

Hash function for a zero-terminated string

Parameters

<i>elt</i>	the string to hash. May be NULL.
------------	----------------------------------

Returns

the hash value of the string

14.3.2.7 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

<i>set</i>	the hash set
<i>elt</i>	the new element to insert in the hash set

Returns

TRUE if the element was not already in the hash set

14.3.2.8 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

<i>set</i>	the hash set
<i>elt</i>	the element to look up in the hash set

Returns

the element found in the hash set or NULL

14.3.2.9 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

<i>fnHashFunc</i>	hash function. May be NULL.
<i>fnEqualFunc</i>	equal function. May be NULL.

<i>fnFreeElt- Func</i>	element free function. May be NULL.
----------------------------	-------------------------------------

Returns

a new hash set

14.3.2.10 int CPLHashSetRemove (CPLHashSet * *set*, const void * *elt*)

Removes an element from a hash set

Parameters

<i>set</i>	the hash set
<i>elt</i>	the new element to remove from the hash set

Returns

TRUE if the element was in the hash set

References _CPLList::pData, and _CPLList::pNext.

14.3.2.11 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

<i>set</i>	the hash set
------------	--------------

Returns

the number of elements in the hash set

14.4 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. 388)*
- **int CPLHTTPParseMultipartMime** (CPLHTTPResult *psResult)
Parses a a MIME multipart message.

14.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

14.4.2 Function Documentation

14.4.2.1 void CPLHTTPDestroyResult (CPLHTTPResult * psResult)

Clean the memory associated with the return value of **CPLHTTPFetch()** (p. 388)

Parameters

<i>psResult</i>	pointer to the return value of CPLHTTPFetch() (p. 388)
-----------------	---

References CPLHTTPResult::pabyData, CPLHTTPResult::pszErrBuf, and CPLHTTPResult::pszContentType.

14.4.2.2 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

14.4.2.3 CPLHTTPResult* CPLHTTPFetch (const char * pszURL, char ** papszOptions)

Fetch a document from an url and return in a string.

Parameters

<i>pszURL</i>	valid URL recognized by underlying download library (libcurl)
<i>papsz-Options</i>	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : <ul style="list-style-type: none"> • 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-ogcwk" • HTTPAUTH=[BASIC/NTLM/ANY] to specify an authentication scheme to use. • USERPWD=userid:password to specify a user and password for authentication

Returns

a `CPLHTTPResult*` structure that must be freed by **CPLHTTPDestroyResult()** (p. 388), or NULL if libcurl support is disabled

References `CPLHTTPResult::nStatus`, `CPLHTTPResult::pszContentType`, and `CPLHTTPResult::pszErrBuf`.

14.4.2.4 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

<i>psResult</i>	pointer to the return value of CPLHTTPFetch() (p. 388)
-----------------	---

Returns

TRUE if the message contains MIME multipart message.

References `CPLHTTPResult::nMimePartCount`, `CPLHTTPResult::pszContentType`, `CPLHTTPResult::pabyData`, `CPLHTTPResult::pasMimePart`, `CPLMimePart::papsz-Headers`, `CPLMimePart::pabyData`, `CPLHTTPResult::nDataLen`, and `CPLMimePart::nDataLen`.

14.5 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)

14.5.1 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.

14.5.2 Typedef Documentation

14.5.2.1 typedef struct **_CPLList CPLList**

List element structure.

14.5.3 Function Documentation

14.5.3.1 **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

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.

Returns

pointer to the head of modified list.

References `_CPLList::psNext`, and `_CPLList::pData`.

14.5.3.2 int CPLListCount (CPLList * *psList*)

Return the number of elements in a list.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

Returns

number of elements in a list.

References `_CPLList::psNext`.

14.5.3.3 void CPLListDestroy (CPLList * *psList*)

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

References `_CPLList::psNext`.

14.5.3.4 CPLList* CPLListGet (CPLList * *psList*, int *nPosition*)

Return the pointer to the specified element in a list.

Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	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`.

14.5.3.5 void* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

Returns

pointer to the data object contained in given list element.

References `_CPLList::pData`.

14.5.3.6 `CPLList* CPLListGetLast (CPLList * psList)`

Return the pointer to last element in a list.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

Returns

pointer to last element in a list.

References `_CPLList::psNext`.

14.5.3.7 `CPLList* CPLListGetNext (CPLList * psElement)`

Return the pointer to next element in a list.

Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

Returns

pointer to the list element preceded by the given element.

References `_CPLList::psNext`.

14.5.3.8 `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

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.
<i>nPosition</i>	position number to insert an object.

Returns

pointer to the head of modified list.

References `_CPLList::pData`, and `_CPLList::psNext`.

14.5.3.9 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

<i>psList</i>	pointer to list head.
<i>nPosition</i>	position number to delete an element.

Returns

pointer to the head of modified list.

References `_CPLList::psNext`.

14.6 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.

14.6.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

14.6.2 Typedef Documentation

14.6.2.1 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. 398). 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. 88) structures.

14.6.3 Enumeration Type Documentation

14.6.3.1 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

14.6.4 Function Documentation

14.6.4.1 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

<i>psParent</i>	the node to attach the child to. May not be NULL.
<i>psChild</i>	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References **CPLXMLNode::psChild**, **CPLXMLNode::eType**, **CXT_Attribute**, and **CPLXMLNode::psNext**.

14.6.4.2 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.395).

Parameters

<i>psOlder-Sibling</i>	the node to attach the sibling after.
<i>psNew-Sibling</i>	the node to add at the end of <i>psOlderSiblings</i> <i>psNext</i> chain.

References CPLXMLNode::*psNext*.

14.6.4.3 void CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifies 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

<i>pszTarget</i>	the string to be adjusted. It is altered in place.
------------------	--

14.6.4.4 CPLXMLNode* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p.88) tree.

Parameters

<i>psTree</i>	the tree to duplicate.
---------------	------------------------

Returns

a copy of the whole tree.

References CPLXMLNode::*eType*, CPLXMLNode::*pszValue*, CPLXMLNode::*psNext*, and CPLXMLNode::*psChild*.

14.6.4.5 **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

<i>psParent</i>	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
<i>pszName</i>	the element name to create.
<i>pszValue</i>	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.

14.6.4.6 **CPLXMLNode*** CPLCreateXMLNode (**CPLXMLNode** * *poParent*, **CPLXMLNodeType** *eType*, const char * *pszText*)

Create an document tree item.

Create a single **CPLXMLNode** (p. 88) object with the desired value and type, and attach it as a child of the indicated parent.

Parameters

<i>poParent</i>	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
<i>eType</i>	the type of the newly created node
<i>pszText</i>	the value of the newly created node

Returns

the newly created node, now owned by the caller (or parent node).

References CPLXMLNode::eType, CPLXMLNode::pszValue, CPLXMLNode::psChild, and CPLXMLNode::psNext.

14.6.4.7 void CPLDestroyXMLNode (CPLXMLNode * *psNode*)

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. 88) and all its children nodes.

Parameters

<i>psNode</i>	the tree to free.
---------------	-------------------

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

14.6.4.8 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. Searching 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

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated).

Returns

the requested element node, or NULL if not found.

References CPLXMLNode::psChild, CPLXMLNode::psNext, CPLXMLNode::eType, CXT_Text, and CPLXMLNode::pszValue.

14.6.4.9 `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. 398) (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

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
<i>pszDefault</i>	the value to return if a corresponding value is not found, may be NULL.

Returns

the requested value or pszDefault if not found.

References CPLXMLNode::eType, CXT_Attribute, CPLXMLNode::psChild, CXT_Text, CPLXMLNode::pszValue, CXT_Element, and CPLXMLNode::psNext.

14.6.4.10 `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 **CPL_ParseXMLString()** (p. 400). Errors in reading the file or parsing the XML will be reported by **CPL_Error()** (p. 380).

The "large file" API is used, so XML files can come from virtualized files.

Parameters

<i>pszFilename</i>	the file to open.
--------------------	-------------------

Returns

NULL on failure, or the document tree on success.

References VSIFOpenL(), VSIFSeekL(), VSIFTellL(), VSIFCloseL(), and VSIFReadL().

14.6.4.11 CPLXMLNode* CPLParseXMLString (const char * *pszString*)

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. 88) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via **CPL_Error()** (p. 380). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. 399) convenience function can be used to parse from a file.

The returned document tree is owned by the caller and should be freed with **CPL_DestroyXMLNode()** (p. 398) 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

<i>pszString</i>	the document to parse.
------------------	------------------------

Returns

parsed tree or NULL on error.

References CXT_Element, CPLXMLNode::pszValue, CXT_Attribute, CXT_Text, CXT_Comment, and CXT_Literal.

14.6.4.12 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

<i>psParent</i>	the node to the child is attached to.
<i>psChild</i>	the child to remove.

Returns

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

14.6.4.13 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 child's first child, and so on.

Use **CPLGetXMLNode()** (p.398) to find a specific child, or along a specific node path.

Parameters

<i>psRoot</i>	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszElement</i>	the name of the element or attribute to search for.

Returns

The matching node or NULL on failure.

References CPLXMLNode::eType, CXT_Element, CXT_Attribute, CPLXMLNode::psz-Value, CPLXMLNode::psChild, and CPLXMLNode::psNext.

14.6.4.14 char* CPLSerializeXMLTree (CPLXMLNode * *psNode*)

Convert tree into string document.

This function converts a **CPLXMLNode** (p.88) 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

<i>psNode</i>	
---------------	--

Returns

the document on success or NULL on failure.

References CPLXMLNode::psNext.

14.6.4.15 `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. 401)) and then written to the named file. Errors writing the file will be reported by **CPLError()** (p. 380). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters

<i>psTree</i>	the document tree to write.
<i>pszFilename</i>	the name of the file to write to.

Returns

TRUE on success, FALSE otherwise.

References VSIFOpenL(), VSIFWriteL(), and VSIFCloseL().

14.6.4.16 `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

<i>psRoot</i>	the subdocument to be updated.
<i>pszPath</i>	the dot seperated path to the target element/attribute.
<i>pszValue</i>	the text value to assign.

Returns

TRUE on success.

References CPLXMLNode::eType, CXT_Element, CPLXMLNode::psChild, CPLXMLNode::psNext, CXT_Text, CPLXMLNode::pszValue, and CXT_Attribute.

14.6.4.17 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

<i>psRoot</i>	the document to operate on.
<i>pszNamespace</i>	the name space prefix (not including colon), or NULL.
<i>bRecurse</i>	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CPLXMLNode::eType, CXT_Element, CXT_Attribute, CPLXMLNode::pszValue, CPLXMLNode::psChild, and CPLXMLNode::psNext.

14.7 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**

14.7.1 Detailed Description

ODBC Abstraction Layer (C++).

14.8 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>
```

Defines

- #define **CPL_LSBINT16PTR**(x) ((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8)
- #define **CPL_LSBINT32PTR**(x)

14.8.1 Detailed Description

Core portability definitions for CPL.

14.8.2 Define Documentation

14.8.2.1 #define CPL_LSBINT16PTR(x) ((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8)

Return a Int16 from the 2 bytes ordered in LSB order at address x

14.8.2.2 #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

14.9 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLRectObj**

Functions

- **CPLQuadTree * CPLQuadTreeCreate** (const **CPLRectObj** *pGlobalBounds, - CPLQuadTreeGetBoundsFunc pfnGetBounds)
- void **CPLQuadTreeDestroy** (**CPLQuadTree** *hQuadtree)
- void **CPLQuadTreeSetBucketCapacity** (**CPLQuadTree** *hQuadtree, int n-BucketCapacity)
- 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 *pnFeatureCount)
- void **CPLQuadTreeForeach** (const **CPLQuadTree** *hQuadtree, CPLQuadTreeForeachFunc pfnForeach, void *pUserData)

14.9.1 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

14.9.2 Function Documentation

14.9.2.1 **CPLQuadTree*** CPLQuadTreeCreate (const **CPLRectObj** * *pGlobalBounds*, CPLQuadTreeGetBoundsFunc *pfnGetBounds*)

Create a new quadtree

Parameters

<i>pGlobal-Bounds</i>	a pointer to the global extent of all the elements that will be inserted
<i>pfnGet-Bounds</i>	a user provided function to get the bounding box of the inserted elements

Returns

a newly allocated quadtree

14.9.2.2 void CPLQuadTreeDestroy (**CPLQuadTree** * *hQuadTree*)

Destroy a quadtree

Parameters

<i>hQuadTree</i>	the quad tree to destroy
------------------	--------------------------

14.9.2.3 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

<i>hQuadTree</i>	the quad tree
<i>pfnForeach</i>	the function called on each element.
<i>pUserData</i>	the user data provided to the function.

14.9.2.4 int CPLQuadTreeGetAdvisedMaxDepth (int *nExpectedFeatures*)

Returns the optimal depth of a quadtree to hold *nExpectedFeatures*

Parameters

<i>nExpectedFeatures</i>	the expected maximum number of elements to be inserted
--------------------------	--

Returns

the optimal depth of a quadtree to hold *nExpectedFeatures*

14.9.2.5 void CPLQuadTreeInsert (CPLQuadTree * *hQuadTree*, void * *hFeature*)

Insert a feature into a quadtree

Parameters

<i>hQuadTree</i>	the quad tree
<i>hFeature</i>	the feature to insert

14.9.2.6 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

<i>hQuadTree</i>	the quad tree
<i>pAoi</i>	the pointer to the area of interest
<i>pnFeatureCount</i>	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

14.9.2.7 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

<i>hQuadTree</i>	the quad tree
<i>nBucketCapacity</i>	the maximum capacity of a node of a quadtree

14.9.2.8 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

<i>hQuadTree</i>	the quad tree
<i>nMaxDepth</i>	the maximum depth allowed

14.10 cpl_string.h File Reference

```
#include "cpl_vsi.h" #include "cpl_error.h" #include "cpl-
_conv.h" #include <string>
```

Classes

- class **CPLString**

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 *pszDelimiter, int nCSLFlags)
- 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)
- 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)

14.10.1 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. 419), and **CSLFetchNameValue()**).

14.10.2 Function Documentation

14.10.2.1 char* CPLBinaryToHex (int nBytes, const GByte * pabyData)

Binary to hexadecimal translation.

Parameters

<i>nBytes</i>	number of bytes of binary data in pabyData.
<i>pabyData</i>	array of data bytes to translate.

Returns

hexadecimal translation, zero terminated. Free with CPLFree().

14.10.2.2 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 reconstituted 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 equivalent (<, > 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 alphanumeric 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.415) currently does not support this format, only **CPLEscapeString()** (p.409). See cpl_csv.cpp for csv parsing support.

Parameters

<i>pszInput</i>	the string to escape.
<i>nLength</i>	The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.
<i>nScheme</i>	the encoding scheme to use.

Returns

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

14.10.2.3 `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

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
<i>chReplacementChar</i>	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

14.10.2.4 `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.

Parameters

<i>pszValue</i>	the string to analyze
-----------------	-----------------------

Returns

returns the type of the value contained in the string.

14.10.2.5 `GByte* CPLHexToBinary (const char * pszHex, int * pnBytes)`

Hexadecimal to binary translation

Parameters

<i>pszHex</i>	the input hex encoded string.
<i>pnBytes</i>	the returned count of decoded bytes placed here.

Returns

returns binary buffer of data - free with CPLFree().

14.10.2.6 int CPLIsUTF8 (const char * *pabyData*, int *nLen*)

Test if a string is encoded as UTF-8.

Parameters

<i>pabyData</i>	input string to test
<i>nLen</i>	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

14.10.2.7 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 V-SIMalloc(), 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 delimiter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. 411).

Parameters

<i>pszName-Value</i>	string in "NAME=VALUE" format.
<i>ppszKey</i>	optional pointer though which to return the name portion.

Returns

the value portion (pointing into original string).

14.10.2.8 `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. 380).

Parameters

<i>pszSource</i>	a NUL terminated string.
<i>pszSrc-Encoding</i>	the source encoding.
<i>pszDst-Encoding</i>	the destination encoding.

Returns

a NUL terminated string which should be freed with CPLFree().

Since

GDAL 1.6.0

14.10.2.9 `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. 380).

Parameters

<i>pszSource</i>	the source wchar_t string, terminated with a 0 wchar_t.
<i>pszSrc-Encoding</i>	the source encoding, typically CPL_ENC_UCS2.
<i>pszDst-Encoding</i>	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

14.10.2.10 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. 380).

Parameters

<i>pszSource</i>	input multi-byte character string.
<i>pszSrc-Encoding</i>	source encoding, typically CPL_ENC_UTF8.
<i>pszDst-Encoding</i>	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

14.10.2.11 `size_t CPLStrcat (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 strcat`.

Truncation can be detected by testing if the return value of `CPLStrcat` is greater or equal to `nDestSize`.

```
char szDest[5];
CPLStrncpy(szDest, "ab", sizeof(szDest));
if (CPLStrcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

<i>pszDest</i>	destination buffer. Must be NUL terminated before running <code>CPLStrcat</code>
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

Returns

the theoretical 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

14.10.2.12 `size_t CPLStrncpy (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 strncpy`.

Truncation can be detected by testing if the return value of `CPLStrncpy` is greater or equal to `nDestSize`.

```
char szDest[5];
if (CPLStrncpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

<i>pszDest</i>	destination buffer
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	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

14.10.2.13 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. 415) 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

<i>pszStr</i>	a NUL terminated string
<i>nMaxLen</i>	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

14.10.2.14 char* CPLUnescapeString (const char * *pszInput*, int * *pnLength*, int *nScheme*)

Unescape a string.

This function does the opposite of **CPLEscapeString()** (p. 409). 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

<i>pszInput</i>	the input string. This is a zero terminated string.
<i>pnLength</i>	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
<i>nScheme</i>	the escaped scheme to undo (see CPLEscapeString() (p. 409) for a list).

Returns

a copy of the unescaped string that should be freed by the application using CPL-Free() when no longer needed.

14.10.2.15 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. 416) as an end condition for loops as it will result in $O(n^2)$ behavior.

Parameters

<i>papszStrList</i>	the string list to count.
---------------------	---------------------------

Returns

the number of entries.

14.10.2.16 void CSLDestroy (char ** *papszStrList*)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters

<i>papszStrList</i>	the list to free.
---------------------	-------------------

14.10.2.17 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. 416).

Parameters

<i>papszStrList</i>	the input string list.
---------------------	------------------------

Returns

newly allocated copy.

14.10.2.18 int CSLFindName (char ** *papszStrList*, const char * *pszName*)

Find StringList entry with given key name.

Parameters

<i>papszStrList</i>	the string list to search.
<i>pszName</i>	the key value to look for (case insensitive).

Returns

-1 on failure or the list index of the first occurrence matching the given key.

14.10.2.19 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

<i>papszList</i>	the string list to be searched.
<i>pszTarget</i>	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

14.10.2.20 char CSLLoad (const char * *pszFname*)**

Load a text file into a string list.

The VSI*L API is used, so **VSIFileOpenL()** (p. 425) 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. 372)).

If reading the file fails a **CPLError()** (p. 380) will be issued and NULL returned.

Parameters

<i>pszFname</i>	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. 416)

14.10.2.21 `char** CSLoad2 (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. 425) 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. 372)).

If reading the file fails a **CPLError()** (p. 380) will be issued and NULL returned.

Parameters

<i>pszFname</i>	the name of the file to read.
<i>nMaxLines</i>	maximum number of lines to read before stopping, or -1 for no limit.
<i>nMaxCols</i>	maximum number of characters in a line before stopping, or -1 for no limit.
<i>papszOptions</i>	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. 416)

Since

GDAL 1.7.0

References **VSIFOpenL()**, **VSIFEofL()**, and **VSIFCloseL()**.

14.10.2.22 `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 precedence.

Parameters

<i>papszOrig</i>	the original list, being modified.
<i>papszOverride</i>	the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns

updated list.

14.10.2.23 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. 417)).

Parameters

<i>papsz-Haystack</i>	the string list to be searched.
<i>pszNeedle</i>	the substring to be searched for.

Returns

the index of the string within the list or -1 on failure.

14.10.2.24 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

<i>papszList</i>	the original list, the modified version is returned.
<i>pszName</i>	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
<i>pszValue</i>	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.

14.10.2.25 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. 411) 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 desirable to include some white space in the new separator. eg. ": " or " = ".

Parameters

<i>papszList</i>	the list to update. Component strings may be freed but the list array will remain at the same location.
<i>psz-Separator</i>	the new separator string to insert.

14.10.2.26 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

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

Returns

TRUE or FALSE.

14.10.2.27 char CSLTokenizeString2 (const char * *pszString*, const char * *pszDelimiters*, int *nCSLTFlags*)**

Tokenize a string.

This function will split a string into tokens based on specified 'delimiter(s)' with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. 416) 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

<i>pszString</i>	the string to be split into tokens.
<i>psz-Delimiters</i>	one or more characters to be used as token delimiters.
<i>nCSLTFlags</i>	an ORing of one or more of the CSLT_ flag values.

Returns

a string list of tokens owned by the caller.

14.11 cpl_vsi.h File Reference

```
#include "cpl_port.h" #include <unistd.h> #include <sys/stat.h>
```


- GByte * **VSIGetMemFileBuffer** (const char *pszFilename, vsi_l_offset *pnDataLength, int bUnlinkAndSeize)

Fetch buffer underlying memory file.

14.11.1 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 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.

14.11.2 Function Documentation

14.11.2.1 int VSIFCloseL (FILE * 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

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
-----------	--

Returns

0 on success or -1 on failure.

References VSIFCloseL().

Referenced by CPLCloseShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIFCloseL().

14.11.2.2 int VSIFEOF (FILE * fp)

Test for end of file.

Returns TRUE (non-zero) if the file read/write offset is currently at the end of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
-----------	--

Returns

TRUE if at EOF else FALSE.

References VSIFeofL().

Referenced by CSLLoad2(), and VSIFeofL().

14.11.2.3 int VSIFFlushL (FILE * *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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
-----------	--

Returns

0 on success or -1 on error.

References VSIFFlushL().

Referenced by VSIFFlushL().

14.11.2.4 FILE* VSIFFileFromMemBuffer (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

<i>pszFilename</i>	the filename to be created.
<i>pabyData</i>	the data buffer for the file.
<i>nDataLength</i>	the length of buffer in bytes.
<i>bTake-Ownership</i>	TRUE to transfer "ownership" of buffer or FALSE.

Returns

open file handle on created file (see **VSIFOpenL()** (p. 425)).

References VSIFFileFromMemBuffer(), and VSIIInstallMemFileHandler().

Referenced by VSIFFileFromMemBuffer().

14.11.2.5 FILE* 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 "FILE *" returned by this function is not really 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.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

<i>pszFilename</i>	the file to open.
<i>pszAccess</i>	access requested (ie. "r", "r+", "w").

Returns

NULL on failure, or the file handle.

References VSIFOpenL().

Referenced by CPOpenShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIFOpenL().

14.11.2.6 int VSIFPrintfL (FILE * *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. 427).

Analog of the POSIX `fprintf()` call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
<i>pszFormat</i>	the printf style format string.

Returns

the number of bytes written or -1 on an error.

References `VSIFPrintfL()`, and `VSIFWriteL()`.

Referenced by `VSIFPrintfL()`.

14.11.2.7 `size_t VSIFReadL (void * pBuffer, size_t nSize, size_t nCount, FILE * 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 `VSIFFileHandler` virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fread()` call.

Parameters

<i>pBuffer</i>	the buffer into which the data should be read (at least <code>nCount * nSize</code> bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).

Returns

number of objects successfully read.

References `VSIFReadL()`.

Referenced by `CPLReadLine2L()`, `CPLParseXMLFile()`, and `VSIFReadL()`.

14.11.2.8 `int VSIFSeekL (FILE * 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 `VSIFFileHandler` virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fseek()` call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 on failure.

References VSIFSeekL().

Referenced by CPLReadLine2L(), CPLParseXMLFile(), and VSIFSeekL().

14.11.2.9 vsi_l_offset VSIFTell (FILE * *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

<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).
-----------	--

Returns

file offset in bytes.

References VSIFTell().

Referenced by CPLReadLine2L(), CPLParseXMLFile(), and VSIFTell().

14.11.2.10 size_t VSIFWriteL (const void * *pBuffer*, size_t *nSize*, size_t *nCount*, FILE * *fp*)

Write bytes to file.

Writes *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

<i>pBuffer</i>	the buffer from which the data should be written (at least <i>nCount</i> * <i>nSize</i> bytes in size).
----------------	---

<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with VSIFOpenL() (p. 425).

Returns

number of objects successfully written.

References VSIFWriteL().

Referenced by CPLSerializeXMLTreeToFile(), VSIFwriteL(), and VSIFprintfL().

14.11.2.11 `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 existence.

Parameters

<i>pszFilename</i>	the name of the file to grab the buffer of.
<i>pnDataLength</i>	(file) length returned in this variable.
<i>bUnlinkAndSeize</i>	TRUE to remove the file, or FALSE to leave unaltered.

Returns

pointer to memory buffer or NULL on failure.

References VSIGetMemFileBuffer().

Referenced by VSIGetMemFileBuffer().

14.11.2.12 `void VSInstallGZipFileHandler (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>

References VSInstallGZipFileHandler().

Referenced by VSInstallGZipFileHandler().

14.11.2.13 void VSInstallMemFileHandler (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. 431) 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. 425), **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( VSIFFileFromMemBuffer( "/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 VSInstallMemFileHandler().

Referenced by VSInstallMemFileHandler(), and VSIFFileFromMemBuffer().

14.11.2.14 void VSInstallSubFileHandler (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. 425), 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 **VSISetL()** (p. 433), **VSIFOpenL()** (p. 425) and operations based on the file handle returned by **VSIFOpenL()** (p. 425) operate properly.

References VSInstallSubFileHandler().

Referenced by VSInstallSubFileHandler().

14.11.2.15 void VSInstallZipFileHandler (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.

Additional documentation is to be found at <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip>

References VSInstallZipFileHandler().

Referenced by VSInstallZipFileHandler().

14.11.2.16 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. 380). 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 VSIMalloc2(), OGRGeometryCollection::importFromWkb(), and OGRPolygon::importFromWkb().

14.11.2.17 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 **CPL_Error()** (p. 380). 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().

14.11.2.18 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

<i>psz-Pathname</i>	the path to the directory to create.
<i>mode</i>	the permissions mode.

Returns

0 on success or -1 on an error.

References VSIMkdir().

Referenced by VSIMkdir().

14.11.2.19 char** VSIReadDir (const char * pszPath)

Read names in a directory.

This function abstracts access to directory contents. 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 **CSL_Destroy()** (p. 416) when no longer needed.

Note that no error is issued via **CPL_Error()** (p. 380) 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

<i>pszPath</i>	the relative, or absolute path of a directory to read.
----------------	--

Returns

The list of entries in the directory, or NULL if the directory doesn't exist.

References VSIRReadDir().

Referenced by VSIRReadDir().

14.11.2.20 int VSIRRename (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

<i>oldpath</i>	the name of the file to be renamed.
<i>newpath</i>	the name the file should be given.

Returns

0 on success or -1 on an error.

References VSIRRename().

Referenced by VSIRRename().

14.11.2.21 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

<i>pszDirname</i>	the path of the directory to be deleted.
-------------------	--

Returns

0 on success or -1 on an error.

References VSIRmdir().

Referenced by CPLUnlinkTree(), and VSIRmdir().

14.11.2.22 int VSISatL (const char * *pszFilename*, VSISatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSISat(), 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

<i>pszFilename</i>	the path of the filesystem object to be queried.
<i>psStatBuf</i>	the structure to load with information.

Returns

0 on success or -1 on an error.

References VSISatL().

Referenced by CPLCheckForFile(), CPLFormCIFilename(), and VSISatL().

14.11.2.23 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

<i>pszFilename</i>	the path of the file to be deleted.
--------------------	-------------------------------------

Returns

0 on success or -1 on an error.

References VSIUnlink().

Referenced by CPLUnlinkTree(), and VSIUnlink().

14.12 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.
- 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.
- 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 ps-Envelope 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_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.
- 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_GetBoundary** (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_Difference** (OGRGeometryH, OGRGeometryH)
Compute difference.
- OGRGeometryH **OGR_G_SymmetricDifference** (OGRGeometryH, OGRGeometryH)
Compute symmetric difference.
- double **OGR_G_Distance** (OGRGeometryH, OGRGeometryH)
Compute distance between two geometries.
- double **OGR_G_GetArea** (OGRGeometryH)
Compute geometry area.
- void **OGR_G_Empty** (OGRGeometryH)
Clear geometry information. This restores the geometry to its 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.
- int **OGR_G_GetPointCount** (OGRGeometryH)
Fetch number of points from a geometry.
- double **OGR_G_GetX** (OGRGeometryH, int)
Fetch the x coordinate of a point from a geometry.
- double **OGR_G_GetY** (OGRGeometryH, int)
Fetch the y 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**)
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. 140) that is already part of an **OGRFeatureDefn** (p. 134).*
- **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.

- const char * **OGR_GetFieldName** (OGRFieldType)
Fetch human readable name for a field type.
- OGRFeatureDefnH **OGR_FD_Create** (const char *)
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. 134) 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.
- **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_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)
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.
- 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. 134).*
- 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_SetStyleString-Directly()** (p. 471) 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. 470) except that it assumes ownership of the passed string.*

 - 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_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 data-source 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.
- 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. 450).*

- 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)

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 **)

This function attempts to create a new data source based on the passed driver.

- OGRDataSourceH **OGR_Dr_CopyDataSource** (OGRSFDriverH, OGRDataSourceH, const char *, char **)

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 *)

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.

- 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)
- OGRStyleMgr (p. 330) 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)
- OGRStyleToolH **OGR_ST_Create** (OGRSTClassId eClassId)
- OGRStyleTool (p. 338) 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_GetParamDbf** (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_SetParamDbf** (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)
OGRStyleTable (p. 334) 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)

14.12.1 Detailed Description

C API and defines for **OGRFeature** (p. 115), **OGRGeometry** (p. 151), and **OGRDataSource** (p. 105) related classes.

See also: **ogr_geometry.h** (p. 542), **ogr_feature.h** (p. 541), **ogrsf_frmts.h** (p. 573), **ogr_featurestyle.h** (p. 542)

14.12.2 Function Documentation

14.12.2.1 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. 450) 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. 260).

Parameters

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>hSrcDS</i>	source datasource
<i>pszNew-Name</i>	the name for the new data source.
<i>papsz-Options</i>	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. 105) handle on success.

References **OGR_Dr_CopyDataSource()**.

Referenced by **OGR_Dr_CopyDataSource()**.

14.12.2.2 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. 450) 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. 260).

Parameters

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>pszName</i>	the name for the new data source.
<i>papsz-Options</i>	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. 105) handle on success.

References OGR_Dr_CreateDataSource(), OGRSFDriver::CreateDataSource(), OGRDataSource::SetDriver(), and OGRDataSource::GetDriver().

Referenced by OGR_Dr_CreateDataSource().

14.12.2.3 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. 261).

Parameters

<i>hDriver</i>	handle to the driver on which data source deletion is based.
<i>pszDataSource</i>	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().

14.12.2.4 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. 261).

Parameters

<i>hDriver</i>	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().

14.12.2.5 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. 262).

Parameters

<i>hDriver</i>	handle to the driver that is used to open file.
<i>pszName</i>	the name of the file, or data source to try and open.
<i>bUpdate</i>	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. 105). This **OGRDataSource** (p. 105) should be closed by deleting the object when it is no longer needed.

References OGR_Dr_Open(), OGRDataSource::GetDriver(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_Open().

14.12.2.6 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 misspelling.

This function is the same as the C++ method **OGRSFDriver::TestCapability()** (p. 262).

Parameters

<i>hDriver</i>	handle to the driver to test the capability against.
<i>pszCap</i>	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

References OGR_Dr_TestCapability().

Referenced by OGR_Dr_TestCapability().

14.12.2.7 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. 106)

Parameters

<i>hDS</i>	handle to the data source where to create the new layer
<i>hSrcLayer</i>	handle to the source layer.
<i>pszNew-Name</i>	the name of the layer to create.
<i>papsz-Options</i>	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().

14.12.2.8 **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. 107).

Parameters

<i>hDS</i>	The dataset handle.
<i>pszName</i>	the name for the new layer. This should ideally not match any existing layer on the datasource.
<i>hSpatialRef</i>	handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available.
<i>eType</i>	the geometry type for the layer. Use <code>wkbUnknown</code> if there are no constraints on the types geometry to be written.
<i>papszOptions</i>	a <code>StringList</code> 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. 192) handle on success.

Example:

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

...

OGRLayerH *hLayer;
char      *papszOptions;

if( OGR_DS_TestCapability( hDS, ODS_CCreateLayer ) )
{
    ...
}

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()**.

14.12.2.9 OGRErr OGR_DS_DeleteLayer (OGRDataSourceH *hDS*, int *iLayer*)

Delete the indicated layer from the datasource.

If this method is supported the ODSDeleteLayer capability will test TRUE on the **OGRDataSource** (p. 105).

This method is the same as the C++ method **OGRDataSource::DeleteLayer()** (p. 108).

Parameters

<i>hDS</i>	handle to the datasource
<i>iLayer</i>	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().

14.12.2.10 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. 109).

Parameters

<i>hDataSource</i>	handle to allocated datasource object.
--------------------	--

References OGR_DS_Destroy().

Referenced by OGR_DS_Destroy().

14.12.2.11 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. 192) handle representing a results set from the query. Note that this **OGRLayer** (p. 192) is in addition to the layers in the data store and must be destroyed with OGR_DS_ReleaseResultSet() 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. 109);

Parameters

<i>hDS</i>	handle to the data source on which the SQL query is executed.
<i>pszSQL-Command</i>	the SQL statement to execute.
<i>hSpatial-Filter</i>	handle to a geometry which represents a spatial filter.
<i>pszDialect</i>	allows control of the statement dialect. By default it is assumed to be "generic" SQL, whatever that is.

Returns

an handle to a **OGRLayer** (p. 192) containing the results of the query. Deallocate with **OGR_DS_ReleaseResultSet()**.

References **OGR_DS_ExecuteSQL()**.

Referenced by **OGR_DS_ExecuteSQL()**.

14.12.2.12 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. 110)

Parameters

<i>hDS</i>	handle to the datasource
------------	--------------------------

Returns

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

References **OGR_DS_GetDriver()**.

Referenced by **OGR_DS_GetDriver()**.

14.12.2.13 OGRLayerH OGR_DS_GetLayer (OGRDataSourceH *hDS*, int *iLayer*)

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. 105) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayer()** (p. 110).

Parameters

<i>hDS</i>	handle to the data source from which to get the layer.
<i>iLayer</i>	a layer number between 0 and OGR_DS_GetLayerCount() (p. 452)-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().

14.12.2.14 OGRLayerH OGR_DS_GetLayerByName (OGRDataSourceH *hDS*, const char * *pszLayerName*)

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. 105) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayerByName()** (p. 110).

Parameters

<i>hDS</i>	handle to the data source from which to get the layer.
<i>pszLayerName</i>	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().

14.12.2.15 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. 111).

Parameters

<i>hDS</i>	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().

14.12.2.16 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. 259) 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. 111).

Parameters

<i>hDS</i>	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().

14.12.2.17 void OGR_DS_ReleaseResultSet (OGRDataSourceH *hDS*, OGRLayerH *hLayer*)

Release results of **OGR_DS_ExecuteSQL()** (p. 450).

This function should only be used to deallocate OGRLayers resulting from an **OGR_DS_ExecuteSQL()** (p. 450) call on the same **OGRDataSource** (p. 105). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 105) may cause errors.

This function is the same as the C++ method **OGRDataSource::ReleaseResultSet()**.

Parameters

<i>hDS</i>	an handle to the data source on which was executed an SQL query.
<i>hLayer</i>	handle to the result of a previous OGR_DS_ExecuteSQL() (p. 450) call.

References OGR_DS_ReleaseResultSet().

Referenced by OGR_DS_ReleaseResultSet().

14.12.2.18 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. 450) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRDataSource::SyncToDisk()** (p. 114)

Parameters

<i>hDS</i>	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().

14.12.2.19 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 misspelling.

This function is the same as the C++ method **OGRDataSource::TestCapability()** (p. 114).

Parameters

<i>hDS</i>	handle to the data source against which to test the capability.
<i>psz-Capability</i>	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

References OGR_DS_TestCapability().

Referenced by OGR_DS_TestCapability().

14.12.2.20 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. 134).

This function is the same as the C++ method **OGRFeature::Clone()** (p. 118).

Parameters

<i>hFeat</i>	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().

14.12.2.21 OGRFeatureH OGR_F_Create (OGRFeatureDefnH *hDefn*)

Feature factory.

Note that the **OGRFeature** (p. 115) will increment the reference count of it's defining **OGRFeatureDefn** (p. 134). Destruction of the **OGRFeatureDefn** (p. 134) 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. 118).

Parameters

<i>hDefn</i>	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().

14.12.2.22 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. 119).

Parameters

<i>hFeat</i>	handle to the feature to destroy.
--------------	-----------------------------------

References OGR_F_Destroy().

Referenced by OGR_F_Destroy().

14.12.2.23 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. 119).

Parameters

<i>hFeat</i>	handle to the feature to dump.
<i>fpOut</i>	the stream to write to, such as strout.

References OGR_F_DumpReadable().

Referenced by OGR_F_DumpReadable().

14.12.2.24 int OGR_F_Equal (OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat*)

Test if two features are the same.

Two features are considered equal if they share them (handle equality) same **OGR-FeatureDefn** (p. 134), 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. 120).

Parameters

<i>hFeat</i>	handle to one of the feature.
<i>hOtherFeat</i>	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().

14.12.2.25 OGRFeatureDefnH OGR_F_GetDefnRef (OGRFeatureH *hFeat*)

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. 120).

Parameters

<i>hFeat</i>	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().

14.12.2.26 long OGR_F_GetFID (OGRFeatureH *hFeat*)

Get feature identifier.

This function is the same as the C++ method **OGRFeature::GetFID()** (p. 120).

Parameters

<i>hFeat</i>	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().

14.12.2.27 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. 121).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnBytes</i>	location to place count of bytes returned.

Returns

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsBinary().

Referenced by OGR_F_GetFieldAsBinary().

14.12.2.28 `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. 121).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(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().

14.12.2.29 `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. 122).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

References OGR_F_GetFieldAsDouble().

Referenced by OGR_F_GetFieldAsDouble().

14.12.2.30 `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. 122).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	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. It's 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().

14.12.2.31 `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. 123).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

References OGR_F_GetFieldAsInteger().

Referenced by OGR_F_GetFieldAsInteger().

14.12.2.32 `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. 123).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	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. It's 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().

14.12.2.33 `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. 123).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value. This string is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsString().

Referenced by OGR_F_GetFieldAsString().

14.12.2.34 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. 416).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. 124).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsStringList().

Referenced by OGR_F_GetFieldAsStringList().

14.12.2.35 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. 134).

This function is the same as the C++ method **OGRFeature::GetFieldCount()** (p. 124).

Parameters

<i>hFeat</i>	handle to the feature to get the fields count from.
--------------	---

Returns

count of fields.

References OGR_F_GetFieldCount().

Referenced by OGR_F_GetFieldCount().

14.12.2.36 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. 125).

Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>i</i>	the field to fetch, from 0 to GetFieldCount()-1.

Returns

an handle to the field definition (from the **OGRFeatureDefn** (p. 134)). This is an internal reference, and should not be deleted or modified.

References OGR_F_GetFieldDefnRef().

Referenced by OGR_F_GetFieldDefnRef().

14.12.2.37 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. 137) method.

This function is the same as the C++ method **OGRFeature::GetFieldIndex()** (p. 125).

Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>pszName</i>	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().

14.12.2.38 OGRGeometryH OGR_F_GetGeometryRef (OGRFeatureH *hFeat*)

Fetch an handle to feature geometry.

This function is the same as the C++ method **OGRFeature::GetGeometryRef()** (p. 125).

Parameters

<i>hFeat</i>	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().

14.12.2.39 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. 126).

Parameters

<i>hFeat</i>	handle to the feature on which field is found.
<i>iField</i>	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().

14.12.2.40 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. 542) for services available to parse it.

This function is the same as the C++ method **OGRFeature::GetStyleString()** (p. 126).

Parameters

<i>hFeat</i>	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().

14.12.2.41 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. 126).

Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to test.

Returns

TRUE if the field has been set, otherwise false.

References OGR_F_IsFieldSet().

Referenced by OGR_F_IsFieldSet().

14.12.2.42 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. 127).

Parameters

<i>hFeat</i>	handle to the feature to set the feature id to.
<i>nFID</i>	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().

14.12.2.43 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nBytes</i>	the number of bytes in pabyData array.
<i>pabyData</i>	the data to apply.

References OGR_F_SetFieldBinary().

Referenced by OGR_F_SetFieldBinary().

14.12.2.44 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

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>nSecond</i>	(0-59)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGR_F_SetFieldDateTime().

Referenced by OGR_F_SetFieldDateTime().

14.12.2.45 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>dfValue</i>	the value to assign.

References OGR_F_SetFieldDouble().

Referenced by OGR_F_SetFieldDouble().

14.12.2.46 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

References OGR_F_SetFieldDoubleList().

Referenced by OGR_F_SetFieldDoubleList().

14.12.2.47 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>nValue</i>	the value to assign.

References OGR_F_SetFieldInteger().

Referenced by OGR_F_SetFieldInteger().

14.12.2.48 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

References OGR_F_SetFieldIntegerList().

Referenced by OGR_F_SetFieldIntegerList().

14.12.2.49 void OGR_F_SetFieldRaw (OGRFeatureH *hFeat*, int *iField*, OGRField * *psValue*)

Set field.

The passed value **OGRField** (p. 139) 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>psValue</i>	handle on the value to assign.

References OGR_F_SetFieldRaw().

Referenced by OGR_F_SetFieldRaw().

14.12.2.50 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pszValue</i>	the value to assign.

References OGR_F_SetFieldString().

Referenced by OGR_F_SetFieldString().

14.12.2.51 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. 127).

Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>papszValues</i>	the values to assign.

References OGR_F_SetFieldStringList().

Referenced by OGR_F_SetFieldStringList().

14.12.2.52 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 *hOtherFeat* does not need to have the same **OGRFeatureDefn** (p. 134). 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. 130).

Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	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().

14.12.2.53 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 *hOtherFeat* does not need to have the same **OGRFeatureDefn** (p. 134). 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. 468) 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. 130).

Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>panMap</i>	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.
<i>bForgiving</i>	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().

14.12.2.54 OGRErr OGR_F_SetGeometry (OGRFeatureH *hFeat*, OGRGeometryH *hGeom*)

Set feature geometry.

This function updates the features geometry, and operate exactly as SetGeometry-Directly(), 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. 132).

Parameters

<i>hFeat</i>	handle to the feature on which new geometry is applied to.
<i>hGeom</i>	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. 134) (checking not yet implemented).

References OGR_F_SetGeometry().

Referenced by OGR_F_SetGeometry().

14.12.2.55 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. 132).

Parameters

<i>hFeat</i>	handle to the feature on which to apply the geometry.
<i>hGeom</i>	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. 134) (checking not yet implemented).

References OGR_F_SetGeometryDirectly().

Referenced by OGR_F_SetGeometryDirectly().

14.12.2.56 void OGR_F_SetStyleString (OGRFeatureH *hFeat*, const char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleString-Directly()** (p. 471) 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. 133).

Parameters

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleString().

Referenced by OGR_F_SetStyleString().

14.12.2.57 void OGR_F_SetStyleStringDirectly (OGRFeatureH *hFeat*, char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleString()** (p. 470) except that it assumes ownership of the passed string.

This function is the same as the C++ method **OGRFeature::SetStyleStringDirectly()** (p. 133).

Parameters

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleStringDirectly().

Referenced by OGR_F_SetStyleStringDirectly().

14.12.2.58 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. 133).

Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to unset.

References OGR_F_UnsetField().

Referenced by OGR_F_UnsetField().

14.12.2.59 void OGR_FD_AddFieldDefn (OGRFeatureDefnH *hDefn*, OGRFieldDefnH *hNewField*)

Add a new field definition to the passed feature definition.

This function should only be called while there are no **OGRFeature** (p. 115) objects in existence based on this **OGRFeatureDefn** (p. 134). The **OGRFieldDefn** (p. 140) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeatureDefn::AddFieldDefn** (p. 135).

Parameters

<i>hDefn</i>	handle to the feature definition to add the field definition to.
<i>hNewField</i>	handle to the new field definition.

References OGR_FD_AddFieldDefn().

Referenced by OGR_FD_AddFieldDefn().

14.12.2.60 OGRFeatureDefnH OGR_FD_Create (const char * *pszName*)

Create a new feature definition object to hold the field definitions.

The **OGRFeatureDefn** (p. 134) 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. 135).

Parameters

<i>pszName</i>	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().

14.12.2.61 int OGR_FD_Dereference (OGRFeatureDefnH *hDefn*)

Decrements the reference count by one.

This function is the same as the C++ method **OGRFeatureDefn::Dereference()** (p. 136).

Parameters

<i>hDefn</i>	handle to the feature definition on witch OGRFeature (p. 115) are based on.
--------------	--

Returns

the updated reference count.

References OGR_FD_Dereference().

Referenced by OGR_FD_Dereference().

14.12.2.62 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

<i>hDefn</i>	handle to the feature definition to be destroyed.
--------------	---

References `OGR_FD_Destroy()`.

Referenced by `OGR_FD_Destroy()`.

14.12.2.63 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. 136).

Parameters

<i>hDefn</i>	handle to the feature definition to get the fields count from.
--------------	--

Returns

count of fields.

References `OGR_FD_GetFieldCount()`.

Referenced by `OGR_FD_GetFieldCount()`.

14.12.2.64 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. 136).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

Parameters

<i>hDefn</i>	handle to the feature definition to get the field definition from.
<i>iField</i>	the field to fetch, between 0 and <code>GetFieldCount()-1</code> .

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().

14.12.2.65 `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. 137).

Parameters

<i>hDefn</i>	handle to the feature definition to get field index from.
<i>pszField-Name</i>	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().

14.12.2.66 `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. 137).

Parameters

<i>hDefn</i>	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().

14.12.2.67 `const char* OGR_FD_GetName (OGRFeatureDefnH hDefn)`

Get name of the **OGRFeatureDefn** (p. 134) passed as an argument.

This function is the same as the C++ method **OGRFeatureDefn::GetName()** (p. 138).

Parameters

<i>hDefn</i>	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().

14.12.2.68 int OGR_FD_GetReferenceCount (OGRFeatureDefnH *hDefn*)

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. 138).

Parameters

<i>hDefn</i>	handle to the feature definition on witch OGRFeature (p. 115) are based on.
--------------	--

Returns

the current reference count.

References OGR_FD_GetReferenceCount().

Referenced by OGR_FD_GetReferenceCount().

14.12.2.69 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. 115) objects referencing this definition.

This function is the same as the C++ method **OGRFeatureDefn::Reference()** (p. 138).

Parameters

<i>hDefn</i>	handle to the feature definition on witch OGRFeature (p. 115) are based on.
--------------	--

Returns

the updated reference count.

References OGR_FD_Reference().

Referenced by OGR_FD_Reference().

14.12.2.70 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. 135).

Parameters

<i>hDefn</i>	handle to the feature definition to be released.
--------------	--

References OGR_FD_Release().

Referenced by OGR_FD_Release().

14.12.2.71 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. 138).

Parameters

<i>hDefn</i>	handle to the layer or feature definition to set the geometry type to.
<i>eType</i>	the new type to assign.

References OGR_FD_SetGeomType().

Referenced by OGR_FD_SetGeomType().

14.12.2.72 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. 141).

Parameters

<i>pszName</i>	the name of the new field definition.
<i>eType</i>	the type of the new field definition.

Returns

handle to the new field definition.

References OGR_Fld_Create().

Referenced by OGR_Fld_Create().

14.12.2.73 void OGR_Fld_Destroy (OGRFieldDefnH hDefn)

Destroy a field definition.

Parameters

<i>hDefn</i>	handle to the field definition to destroy.
--------------	--

References OGR_Fld_Destroy().

Referenced by OGR_Fld_Destroy().

14.12.2.74 OGRJustification OGR_Fld_GetJustify (OGRFieldDefnH hDefn)

Get the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::GetJustify()** (p. 142).

Parameters

<i>hDefn</i>	handle to the field definition to get justification from.
--------------	---

Returns

the justification.

References OGR_Fld_GetJustify().

Referenced by OGR_Fld_GetJustify().

14.12.2.75 const char* OGR_Fld_GetNameRef (OGRFieldDefnH hDefn)

Fetch name of this field.

This function is the same as the CPP method **OGRFieldDefn::GetNameRef()** (p. 142).

Parameters

<i>hDefn</i>	handle to the field definition.
--------------	---------------------------------

Returns

the name of the field definition.

References OGR_Fld_GetNameRef().

Referenced by OGR_Fld_GetNameRef().

14.12.2.76 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. 142).

Parameters

<i>hDefn</i>	handle to the field definition to get precision from.
--------------	---

Returns

the precision.

References OGR_Fld_GetPrecision().

Referenced by OGR_Fld_GetPrecision().

14.12.2.77 OGRFieldType OGR_Fld_GetType (OGRFieldDefnH hDefn)

Fetch type of this field.

This function is the same as the CPP method **OGRFieldDefn::GetType()** (p. 142).

Parameters

<i>hDefn</i>	handle to the field definition to get type from.
--------------	--

Returns

field type.

References OGR_Fld_GetType().

Referenced by OGR_Fld_GetType().

14.12.2.78 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. 143).

Parameters

<i>hDefn</i>	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().

14.12.2.79 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. 143).

Parameters

<i>hDefn</i>	handle to the field definition to set to.
<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>nPrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJ-Undefined.

References OGR_Fld_Set().

Referenced by OGR_Fld_Set().

14.12.2.80 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. 144).

Parameters

<i>hDefn</i>	handle to the field definition to set justification to.
<i>eJustify</i>	the new justification.

References OGR_Fld_SetJustify().

Referenced by OGR_Fld_SetJustify().

14.12.2.81 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. 144).

Parameters

<i>hDefn</i>	handle to the field definition to apply the new name to.
<i>pszName</i>	the new name to apply.

References OGR_Fld_SetName().

Referenced by OGR_Fld_SetName().

14.12.2.82 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. 144).

Parameters

<i>hDefn</i>	handle to the field definition to set precision to.
<i>nPrecision</i>	the new precision.

References OGR_Fld_SetPrecision().

Referenced by OGR_Fld_SetPrecision().

14.12.2.83 void OGR_Fld_SetType (OGRFieldDefnH *hDefn*, OGRFieldType *eType*)

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 140) that is already part of an **OGRFeatureDefn** (p. 134).

This function is the same as the CPP method **OGRFieldDefn::SetType()** (p. 144).

Parameters

<i>hDefn</i>	handle to the field definition to set type to.
<i>eType</i>	the new field type.

References OGR_Fld_SetType().

Referenced by OGR_Fld_SetType().

14.12.2.84 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. 145).

Parameters

<i>hDefn</i>	handle to the field definition to set width to.
<i>nNewWidth</i>	the new width.

References OGR_Fld_SetWidth().

Referenced by OGR_Fld_SetWidth().

14.12.2.85 OGRErr OGR_G_AddGeometry (OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom*)

Add a geometry to a geometry container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174).

Parameters

<i>hGeom</i>	existing geometry container.
<i>hNewSubGeom</i>	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 wkbPolygon, OGRLinearRing::WkbSize(), OGRLineString::getGeometryType(), wkbLineString, wkbMultiPoint, wkbMultiLineString, wkbMultiPolygon, and wkbGeometryCollection.

14.12.2.86 OGRErr OGR_G_AddGeometryDirectly (OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom*)

Add a geometry directly to an existing geometry container.

Some subclasses of **OGRGeometryCollection** (p. 172) 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. 174).

There is no SFCOM analog to this method.

Parameters

<i>hGeom</i>	existing geometry.
<i>hNewSub-Geom</i>	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 wkbPolygon, OGRLinearRing::WkbSize(), OGRLineString::getGeometryType(), wkbLineString, wkbMultiPoint, wkbMultiLineString, wkbMultiPolygon, and wkbGeometryCollection.

14.12.2.87 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

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.
<i>dfZ</i>	z coordinate of point to add.

References wkbPoint, and wkbLineString.

14.12.2.88 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

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.

References wkbPoint, and wkbLineString.

14.12.2.89 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. 268), 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. 154).

Parameters

<i>hGeom</i>	handle on the geometry to apply the new spatial reference system.
<i>hSRS</i>	handle on the new spatial reference system to apply.

References OGR_G_AssignSpatialReference().

Referenced by OGR_G_AssignSpatialReference().

14.12.2.90 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 segments 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. 155).

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

<i>hTarget</i>	the geometry.
<i>dfDist</i>	the buffer distance to be applied.
<i>nQuadSegs</i>	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().

14.12.2.91 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. 155).

Parameters

<i>hGeom</i>	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().

14.12.2.92 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. 156).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if *hThis* contains *hOther* geometry, otherwise FALSE.

References OGR_G_Contains().

Referenced by OGR_G_Contains().

14.12.2.93 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. 156).

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

<i>hTarget</i>	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().

14.12.2.94 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. 507) on the returned geometry to establish the number of bytes it required in WKB format.

The **OGRGeometryFactory::createFromWkb()** (p. 187) CPP method is the same as this function.

Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>phGeometry</i>	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL in case of failure.
<i>nBytes</i>	the number of bytes of data available in <i>pabyData</i> , 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 OGR_G_CreateFromWkb(), and OGRGeometryFactory::createFromWkb().

Referenced by OGR_G_CreateFromWkb().

14.12.2.95 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. 188) CPP method is the same as this function.

Parameters

<i>ppszData</i>	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.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>phGeometry</i>	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 OGR_G_CreateFromWkt(), and OGRGeometryFactory::createFromWkt().

Referenced by OGR_G_CreateFromWkt().

14.12.2.96 OGRGeometryH OGR_G_CreateGeometry (OGRwkbGeometryType *eGeometryType*)

Create an empty geometry of desired type.

This is equivalent 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. 188).

Parameters

<i>eGeometry-Type</i>	the type code of the geometry to be created.
-----------------------	--

Returns

handle to the newly create geometry or NULL on failure.

References OGR_G_CreateGeometry(), and OGRGeometryFactory::createGeometry().

Referenced by OGR_G_CreateGeometry().

14.12.2.97 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. 157).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if they are crossing, otherwise FALSE.

References OGR_G_Crosses().

Referenced by OGR_G_Crosses().

14.12.2.98 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. 189).

Parameters

<i>hGeom</i>	handle to the geometry to delete.
--------------	-----------------------------------

References OGR_G_DestroyGeometry(), and OGRGeometryFactory::destroyGeometry().

Referenced by OGR_G_DestroyGeometry().

14.12.2.99 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. 157).

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

<i>hThis</i>	the geometry.
<i>hOther</i>	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().

14.12.2.100 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. 157).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if they are disjoint, otherwise FALSE.

References OGR_G_Disjoint().

Referenced by OGR_G_Disjoint().

14.12.2.101 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. 158).

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

<i>hFirst</i>	the first geometry to compare against.
<i>hOther</i>	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().

14.12.2.102 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. 158).

Parameters

<i>hGeom</i>	handle on the geometry to dump.
<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.

References OGR_G_DumpReadable().

Referenced by OGR_G_DumpReadable().

14.12.2.103 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. 159).

Parameters

<i>hGeom</i>	handle on the geometry to empty.
--------------	----------------------------------

References OGR_G_Empty().

Referenced by OGR_G_Empty().

14.12.2.104 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. 159) method.

Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOther</i>	handle on the other geometry to test against.

Returns

TRUE if equivalent or FALSE otherwise.

References OGR_G_Equals().

Referenced by OGR_G_Equals().

14.12.2.105 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. 160).

Parameters

<i>hGeom</i>	handle on the geometry to convert to a well know binary data from.
<i>eOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyDstBuffer</i>	a buffer into which the binary representation is written. This buffer must be at least OGR_G_WkbSize() (p. 507) byte in size.

Returns

Currently OGRERR_NONE is always returned.

References OGR_G_ExportToWkb().

Referenced by OGR_G_ExportToWkb().

14.12.2.106 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. 161).

Parameters

<i>hGeom</i>	handle on the geometry to convert to a text format from.
<i>ppszSrcText</i>	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().

14.12.2.107 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. 161).

Parameters

<i>hGeom</i>	handle on the geometry to convert.
--------------	------------------------------------

References OGR_G_FlattenTo2D().

Referenced by OGR_G_FlattenTo2D().

14.12.2.108 double OGR_G_GetArea (OGRGeometryH *hGeom*)

Compute geometry area.

Computes the area for an **OGRLinearRing** (p. 206), **OGRPolygon** (p. 246) or **OGR-MultiPolygon** (p. 233). Undefined for all other geometry types (returns zero).

This function utilizes the C++ get_Area() methods such as **OGRPolygon::get_Area()**

(p. 251).

Parameters

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

Returns

the area or 0.0 for unsupported geometry types.

References `wkbPolygon`, `wkbMultiPolygon`, `wkbLinearRing`, `wkbLineString`, and `wkbGeometryCollection`.

14.12.2.109 OGRGeometryH OGR_G_GetBoundary (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_GetBoundary()** (p. 492).

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

<i>hTarget</i>	The Geometry to calculate the boundary of.
----------------	--

Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

References `OGR_G_GetBoundary()`.

Referenced by `OGR_G_GetBoundary()`.

14.12.2.110 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. 162).

Parameters

<i>hGeom</i>	handle on the geometry to get the dimension of the coordinates from.
--------------	--

Returns

in practice this always returns 2 indicating that coordinates are specified within a two dimensional space.

References OGR_G_GetCoordinateDimension().

Referenced by OGR_G_GetCoordinateDimension().

14.12.2.111 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_GetCoordinateDimension()** (p. 492) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. 162).

Parameters

<i>hGeom</i>	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().

14.12.2.112 void OGR_G_GetEnvelope (OGRGeometryH *hGeom*, OGREnvelope * *psEnvelope*)

Computes and returns the bounding envelope for this geometry in the passed ps-Envelope structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. 163).

Parameters

<i>hGeom</i>	handle of the geometry to get envelope from.
<i>psEnvelope</i>	the structure in which to place the results.

References OGR_G_GetEnvelope().

Referenced by OGR_G_GetEnvelope().

14.12.2.113 `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.

Parameters

<i>hGeom</i>	single geometry or geometry container from which to get the number of elements.
--------------	---

Returns

the number of elements.

References `wkbPolygon`, `wkbMultiPoint`, `wkbMultiLineString`, `wkbMultiPolygon`, and `wkbGeometryCollection`.

14.12.2.114 `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. 163).

Parameters

<i>hGeom</i>	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()`.

14.12.2.115 `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 until the next change to the geometry container. Use `OGR_G_Clone()` (p. 484) to make a copy.

This function relates to the SFCOM IGeometryCollection::get_Geometry() method.

This function is the same as the CPP method **OGRGeometryCollection::get-GeometryRef()** (p. 179).

Parameters

<i>hGeom</i>	handle to the geometry container from which to get a geometry from.
<i>iSubGeom</i>	the index of the geometry to fetch, between 0 and getNumGeometries() - 1.

Returns

handle to the requested geometry.

References wkbPolygon, wkbMultiPoint, wkbMultiLineString, wkbMultiPolygon, and wkbGeometryCollection.

14.12.2.116 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. 163).

Parameters

<i>hGeom</i>	handle on the geometry to get type from.
--------------	--

Returns

the geometry type code.

References OGR_G_GetGeometryType(), and wkbUnknown.

Referenced by OGR_G_GetGeometryType().

14.12.2.117 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

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>i</i>	the vertex to fetch, from 0 to getNumPoints()-1, zero for a point.
<i>pdfX</i>	value of x coordinate.
<i>pdfY</i>	value of y coordinate.
<i>pdfZ</i>	value of z coordinate.

References `wkbPoint`, and `wkbLineString`.

14.12.2.118 `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

<i>hGeom</i>	handle to the geometry from which to get the number of points.
--------------	--

Returns

the number of points.

References `wkbPoint`, `wkbLineString`, and `OGRLineString::getNumPoints()`.

14.12.2.119 `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. 164).

Parameters

<i>hGeom</i>	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()`.

14.12.2.120 `double OGR_G_GetX (OGRGeometryH hGeom, int i)`

Fetch the x coordinate of a point from a geometry.

Parameters

<i>hGeom</i>	handle to the geometry from which to get the x coordinate.
<i>i</i>	point to get the x coordinate.

Returns

the X coordinate of this point.

References wkbPoint, and wkbLineString.

14.12.2.121 `double OGR_G::GetY (OGRGeometryH hGeom, int i)`

Fetch the x coordinate of a point from a geometry.

Parameters

<i>hGeom</i>	handle to the geometry from which to get the y coordinate.
<i>i</i>	point to get the Y coordinate.

Returns

the Y coordinate of this point.

References wkbPoint, and wkbLineString.

14.12.2.122 `double OGR_G::GetZ (OGRGeometryH hGeom, int i)`

Fetch the z coordinate of a point from a geometry.

Parameters

<i>hGeom</i>	handle to the geometry from which to get the Z coordinate.
<i>i</i>	point to get the Z coordinate.

Returns

the Z coordinate of this point.

References wkbPoint, and wkbLineString.

14.12.2.123 `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. 164).

Parameters

<i>hGeom</i>	handle on the geometry to assign the well know binary data to.
<i>pabyData</i>	the binary input data.
<i>nSize</i>	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().

14.12.2.124 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. 165).

Parameters

<i>hGeom</i>	handle on the geometry to assign well know text data to.
<i>ppszSrcText</i>	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().

14.12.2.125 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. 499) function can be used to test if two geometries intersect.

This function is the same as the C++ method **OGRGeometry::Intersection()** (p. 165).

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

<i>hThis</i>	the geometry.
<i>hOther</i>	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().

14.12.2.126 int OGR_G_Intersects (OGRGeometryH *hGeom*, OGRGeometryH *hOtherGeom*)

Do these features intersect?

Currently this is not implemented in a rigorous fashion, and generally just tests whether the envelopes of the two features intersect. Eventually this will be made rigorous.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. 166).

Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOther-Geom</i>	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().

14.12.2.127 int OGR_G_IsEmpty (OGRGeometryH *hGeom*)

Test if the geometry is empty.

This method is the same as the CPP method **OGRGeometry::IsEmpty()** (p. 166).

Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsEmpty().

Referenced by OGR_G_IsEmpty().

14.12.2.128 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. 167).

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

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsRing().

Referenced by OGR_G_IsRing().

14.12.2.129 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. 167) method.

If OGR is built without the GEOS library, this function will always return FALSE.

Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

Returns

TRUE if object is simple, otherwise FALSE.

References OGR_G_IsSimple().

Referenced by OGR_G_IsSimple().

14.12.2.130 int OGR_G_IsValid (OGRGeometryH *hGeom*)

Test if the geometry is valid.

This function is the same as the C++ method **OGRGeometry::IsValid()** (p. 167).

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

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsValid().

Referenced by OGR_G_IsValid().

14.12.2.131 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. 168).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if they are overlapping, otherwise FALSE.

References OGR_G_Overlaps().

Referenced by OGR_G_Overlaps().

14.12.2.132 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. 182).

Parameters

<i>hGeom</i>	the existing geometry to delete from.
<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	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 wkbPolygon, wkbMultiPoint, wkbMultiLineString, wkbMultiPolygon, and wkbGeometryCollection.

14.12.2.133 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. 168).

Parameters

<i>hGeom</i>	handle on the geometry to segmentize
<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization

References OGR_G_Segmentize().

Referenced by OGR_G_Segmentize().

14.12.2.134 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 *i*Point is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

Parameters

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.
<i>dfZ</i>	input Z coordinate to assign (defaults to zero).

References `wkbPoint`, and `wkbLineString`.

14.12.2.135 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 *i*Point is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

Parameters

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.

References `wkbPoint`, and `wkbLineString`.

14.12.2.136 OGRGeometryH OGR_G_SymmetricDifference (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. 169).

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

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

References OGR_G_SymmetricDifference().

Referenced by OGR_G_SymmetricDifference().

14.12.2.137 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. 169).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if they are touching, otherwise FALSE.

References OGR_G_Touches().

Referenced by OGR_G_Touches().

14.12.2.138 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. 99) object, and the actual

SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 268) of the **OGRCoordinateTransformation** (p. 99) will be assigned to the geometry.

This function is the same as the CPP method **OGRGeometry::transform** (p. 170).

Parameters

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hTransform</i>	handle on the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

References OGR_G_Transform().

Referenced by OGR_G_Transform().

14.12.2.139 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. 99) object it is significantly more expensive to use this function to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 99) 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. 170).

Parameters

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hSRS</i>	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().

14.12.2.140 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. 171).

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

<i>hThis</i>	the geometry.
<i>hOther</i>	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().

14.12.2.141 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. 171).

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

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

Returns

TRUE if *hThis* is within *hOther*, otherwise FALSE.

References OGR_G_Within().

Referenced by OGR_G_Within().

14.12.2.142 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. 172).

Parameters

<i>hGeom</i>	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().

14.12.2.143 const char* OGR_GetFieldName (OGRFieldType *eType*)

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldName()** (p. 141).

Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

Returns

the name.

References OGR_GetFieldName(), and OGRFieldDefn::GetFieldName().

Referenced by OGR_GetFieldName().

14.12.2.144 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

<i>hLayer</i>	handle to the layer
---------------	---------------------

Returns

OGRERR_NONE on success.

References OGR_L_CommitTransaction().

Referenced by OGR_L_CommitTransaction().

14.12.2.145 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. 194).

Parameters

<i>hLayer</i>	handle to the layer to write the feature to.
<i>hFeat</i>	the handle of the feature to write to disk.

Returns

OGRERR_NONE on success.

References OGR_L_CreateFeature().

Referenced by OGR_L_CreateFeature().

14.12.2.146 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. 134) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 134) used by a layer directly.

This function is the same as the C++ method **OGRLayer::CreateField()** (p. 194).

Parameters

<i>hLayer</i>	handle to the layer to write the field definition.
<i>hField</i>	handle of the field definition to write to disk.
<i>bApproxOK</i>	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().

14.12.2.147 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. 518) 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. 195).

Parameters

<i>hLayer</i>	handle to the layer
<i>nFID</i>	the feature id to be deleted from the layer

Returns

OGRERR_NONE on success.

References OGR_L_DeleteFeature().

Referenced by OGR_L_DeleteFeature().

14.12.2.148 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. 509) without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

This function is the same as the C++ method **OGRLayer::GetExtent()** (p. 195).

Parameters

<i>hLayer</i>	handle to the layer from which to get extent.
<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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().

14.12.2.149 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 OGR_NULLFID. 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. 457)) 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. 510); 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. 510) call.

The returned feature should be free with **OGR_F_Destroy()** (p. 456).

This function is the same as the C++ method **OGRLayer::GetFeature()** (p. 196).

Parameters

<i>hLayer</i>	handle to the layer that owned the feature.
<i>nFeatureId</i>	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().

14.12.2.150 `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.

This function is the same as the CPP **OGRLayer::GetFeatureCount()** (p. 197).

Parameters

<i>hLayer</i>	handle to the layer that owned the features.
<i>bForce</i>	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().

14.12.2.151 `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. 197)

Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

Returns

fid column name.

References OGR_L::GetFIDColumn().

Referenced by OGR_L::GetFIDColumn().

14.12.2.152 `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. 197)

Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

Returns

geometry column name.

References **OGR_L_GetGeometryColumn()**.

Referenced by **OGR_L_GetGeometryColumn()**.

14.12.2.153 OGRFeatureDefnH OGR_L_GetLayerDefn (OGRLayerH *hLayer*)

Fetch the schema information for this layer.

The returned handle to the **OGRFeatureDefn** (p. 134) is owned by the **OGRLayer** (p. 192), 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. 198).

Parameters

<i>hLayer</i>	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()**.

14.12.2.154 OGRFeatureH OGR_L_GetNextFeature (OGRLayerH *hLayer*)

Fetch the next available feature from this layer.

The returned feature becomes the responsibility of the caller to delete with **OGR_F_Destroy()** (p. 456). It is critical that all features associated with an **OGRLayer** (p. 192) (more specifically an **OGRFeatureDefn** (p. 134)) be deleted before that layer/data-source 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. 514) function can be used to start at the beginning again. Random reading, writing and spatial filtering will be added to the **OGRLayer** (p. 192) in the future.

This function is the same as the C++ method **OGRLayer::GetNextFeature()** (p. 198).

Parameters

<i>hLayer</i>	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().

14.12.2.155 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. 199).

Parameters

<i>hLayer</i>	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().

14.12.2.156 OGRSpatialReferenceH OGR_L_GetSpatialRef (OGRLayerH *hLayer*)

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 192) and should not be modified or freed by the application.

This function is the same as the C++ method **OGRLayer::GetSpatialRef()** (p. 199).

Parameters

<i>hLayer</i>	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().

14.12.2.157 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. 200).

Parameters

<i>hLayer</i>	handle to the layer on which features are read.
---------------	---

References OGR_L_ResetReading().

Referenced by OGR_L_ResetReading().

14.12.2.158 OGRErr OGR_L_RollbackTransaction (OGRLayerH hLayer)

For datasources which support transactions, RollbackTransaction will roll back a data-source 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

<i>hLayer</i>	handle to the layer
---------------	---------------------

Returns

OGRERR_NONE on success.

References OGR_L_RollbackTransaction().

Referenced by OGR_L_RollbackTransaction().

14.12.2.159 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. 512) 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. 514)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. 201).

Parameters

<i>hLayer</i>	handle to the layer on which attribute query will be executed.
<i>pszQuery</i>	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().

14.12.2.160 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 **OGR-Feature** (p. 115).

Use OGR_L_TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **OGR_L_SetFeature()** (p. 515).

This function is the same as the C++ method **OGRLayer::SetFeature()** (p. 201).

Parameters

<i>hLayer</i>	handle to the layer to write the feature.
<i>hFeat</i>	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().

14.12.2.161 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. 202)

Parameters

<i>hLayer</i>	handle to the layer
<i>nIndex</i>	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()`.

14.12.2.162 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. 512) 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. 493)) 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. 513)). In the future this may be generalized.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. 202).

Parameters

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>hGeom</i>	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().

14.12.2.163 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.512) 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.199)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGR-Layer::SetSpatialFilter()** (p.202). 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.203).

Parameters

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

References OGR_L_SetSpatialFilterRect().

Referenced by OGR_L_SetSpatialFilterRect().

14.12.2.164 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

<i>hLayer</i>	handle to the layer
---------------	---------------------

Returns

OGRERR_NONE on success.

References OGR_L_StartTransaction().

Referenced by OGR_L_StartTransaction().

14.12.2.165 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. 450) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRLayer::SyncToDisk()** (p. 204)

Parameters

<i>hLayer</i>	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().

14.12.2.166 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. 192) 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. 192) 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. 115) 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. 511)) 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. 509)) 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.
- **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. 204).

Parameters

<i>hLayer</i>	handle to the layer to get the capability from.
<i>pszCap</i>	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().

14.12.2.167 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. 331).

Parameters

<i>hSM</i>	handle to the style manager.
<i>hST</i>	the style tool defining the part to add.

Returns

TRUE on success, FALSE on errors.

References OGR_SM_AddPart().

Referenced by OGR_SM_AddPart().

14.12.2.168 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()**.

Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyle-Name</i>	the name of the style to add.
<i>pszStyle-String</i>	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().

14.12.2.169 **OGRStyleMgrH** **OGR_SM_Create** (**OGRStyleTableH** *hStyleTable*)

OGRStyleMgr (p. 330) factory.

This function is the same as the C++ method **OGRStyleMgr::OGRStyleMgr()** (p. 331).

Parameters

<i>hStyleTable</i>	pointer to OGRStyleTable (p. 334) 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()**.

14.12.2.170 **void** **OGR_SM_Destroy** (**OGRStyleMgrH** *hSM*)

Destroy Style Manager.

This function is the same as the C++ method **OGRStyleMgr::~~OGRStyleMgr()** (p. 331).

Parameters

<i>hSM</i>	handle to the style manager to destroy.
------------	---

References **OGR_SM_Destroy()**.

Referenced by **OGR_SM_Destroy()**.

14.12.2.171 **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. 331).

Parameters

<i>hSM</i>	handle to the style manager.
<i>nPartId</i>	the part number (0-based index).
<i>pszStyle-String</i>	(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().

14.12.2.172 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. 332).

Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyle-String</i>	(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().

14.12.2.173 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. 332).

Parameters

<i>hSM</i>	handle to the style manager.
<i>hFeat</i>	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().

14.12.2.174 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. 333).

Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyle-String</i>	the style string to use (can be NULL).

Returns

TRUE on success, FALSE on errors.

References OGR_SM_InitStyleString().

Referenced by OGR_SM_InitStyleString().

14.12.2.175 OGRStyleToolH OGR_ST_Create (OGRSTClassId eClassId)

OGRStyleTool (p. 338) factory.

This function is a constructor for **OGRStyleTool** (p. 338) derived classes.

Parameters

<i>eClassId</i>	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().

14.12.2.176 void OGR_ST_Destroy (OGRStyleToolH hST)

Destroy Style Tool.

Parameters

<i>hST</i>	handle to the style tool to destroy.
------------	--------------------------------------

References OGR_ST_Destroy().

Referenced by OGR_ST_Destroy().

14.12.2.177 `double OGR_ST_GetParamDbf (OGRStyleToolH hST, int eParam, int * bValuesNull)`

Get Style Tool parameter value as a double.

Maps to the **OGRStyleTool** (p. 338) subclasses' GetParamDbf() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValuesNull</i>	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 bValuesNull.

References OGR_ST_GetParamDbf().

Referenced by OGR_ST_GetParamDbf().

14.12.2.178 `int OGR_ST_GetParamNum (OGRStyleToolH hST, int eParam, int * bValuesNull)`

Get Style Tool parameter value as an integer.

Maps to the **OGRStyleTool** (p. 338) subclasses' GetParamNum() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValuesNull</i>	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 bValuesNull.

References OGR_ST_GetParamNum().

Referenced by OGR_ST_GetParamNum().

14.12.2.179 `const char* OGR_ST_GetParamStr (OGRStyleToolH hST, int eParam, int * bValuesNull)`

Get Style Tool parameter value as string.

Maps to the **OGRStyleTool** (p. 338) subclasses' GetParamStr() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	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().

14.12.2.180 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

<i>hST</i>	handle to the style tool.
<i>pszColor</i>	the color to parse
<i>pnRed</i>	pointer to an int in which the red value will be returned
<i>pnGreen</i>	pointer to an int in which the green value will be returned
<i>pnBlue</i>	pointer to an int in which the blue value will be returned
<i>pnAlpha</i>	pointer to an int in which the (optional) alpha value will be returned

Returns

TRUE if the color could be successfully parsed, or FALSE in case of errors.

References OGR_ST_GetRGBFromString().

Referenced by OGR_ST_GetRGBFromString().

14.12.2.181 const char* OGR_ST_GetStyleString (OGRStyleToolH *hST*)

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. 338) subclasses' GetStyleString() methods.

Parameters

<i>hST</i>	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().

14.12.2.182 OGRSTClassId OGR_ST_GetType (OGRStyleToolH *hST*)

Determine type of Style Tool.

Parameters

<i>hST</i>	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().

14.12.2.183 OGRSTUnitId OGR_ST_GetUnit (OGRStyleToolH *hST*)

Get Style Tool units.

Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

Returns

the style tool units.

References OGR_ST_GetUnit().

Referenced by OGR_ST_GetUnit().

14.12.2.184 void OGR_ST_SetParamDbl (OGRStyleToolH *hST*, int *eParam*, double *dfValue*)

Set Style Tool parameter value from a double.

Maps to the **OGRStyleTool** (p. 338) subclasses' SetParamDbf() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>dfValue</i>	the new parameter value

References OGR_ST_SetParamDbf().

Referenced by OGR_ST_SetParamDbf().

14.12.2.185 void OGR_ST_SetParamNum (OGRStyleToolH *hST*, int *eParam*, int *nValue*)

Set Style Tool parameter value from an integer.

Maps to the **OGRStyleTool** (p. 338) subclasses' SetParamNum() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>nValue</i>	the new parameter value

References OGR_ST_SetParamNum().

Referenced by OGR_ST_SetParamNum().

14.12.2.186 void OGR_ST_SetParamStr (OGRStyleToolH *hST*, int *eParam*, const char * *pszValue*)

Set Style Tool parameter value from a string.

Maps to the **OGRStyleTool** (p. 338) subclasses' SetParamStr() methods.

Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>pszValue</i>	the new parameter value

References OGR_ST_SetParamStr().

Referenced by OGR_ST_SetParamStr().

14.12.2.187 void OGR_ST_SetUnit (OGRStyleToolH *hST*, OGRSTUnitId *eUnit*, double *dfGroundPaperScale*)

Set Style Tool units.

This function is the same as OGRStyleTool::SetUnit()

Parameters

<i>hST</i>	handle to the style tool.
<i>eUnit</i>	the new unit.
<i>dfGround-PaperScale</i>	ground to paper scale factor.

References OGR_ST_SetUnit().

Referenced by OGR_ST_SetUnit().

14.12.2.188 OGRStyleTableH OGR_STBL_Create (void)

OGRStyleTable (p. 334) 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().

14.12.2.189 void OGR_STBL_Destroy (OGRStyleTableH *hSTBL*)

Destroy Style Table.

Parameters

<i>hSTBL</i>	handle to the style table to destroy.
--------------	---------------------------------------

References OGR_STBL_Destroy().

Referenced by OGR_STBL_Destroy().

14.12.2.190 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. 336).

Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszName</i>	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().

14.12.2.191 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.336).

Parameters

<i>hStyleTable</i>	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().

14.12.2.192 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

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

Returns

the next style string or NULL on error.

References OGR_STBL_GetNextStyle().

Referenced by OGR_STBL_GetNextStyle().

14.12.2.193 `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. 337).

Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to load from.

Returns

TRUE on success, FALSE on error

References OGR_STBL_LoadStyleTable().

Referenced by OGR_STBL_LoadStyleTable().

14.12.2.194 `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

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

References OGR_STBL_ResetStyleStringReading().

Referenced by OGR_STBL_ResetStyleStringReading().

14.12.2.195 `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. 338).

Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to save to.

Returns

TRUE on success, FALSE on error

References OGR_STBL_SaveStyleTable().

Referenced by OGR_STBL_SaveStyleTable().

14.12.2.196 OGRGeometryH OGRBuildPolygonFromEdges (OGRGeometryH *hLines*, int *bBestEffort*, int *bAutoClose*, double *dfTolerance*, OGRErr * *peErr*)

Build a ring from a bunch of arcs.

Parameters

<i>hLines</i>	handle to an OGRGeometryCollection (p. 172) (or OGRMultiLineString (p. 226)) containing the line string geometries to be built into rings.
<i>bBestEffort</i>	not yet implemented???
<i>bAutoClose</i>	indicates if the ring should be close when first and last points of the ring are the same.
<i>dfTolerance</i>	tolerance into which two arcs are considered close enough to be joined.
<i>peErr</i>	OGRErr_NONE on success, or OGRErr_FAILURE on failure.

Returns

an handle to the new geometry, a polygon.

References OGRBuildPolygonFromEdges(), OGRGeometry::getGeometryType(), wkbGeometryCollection, OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::getGeometryRef(), wkbLineString, wkbMultiLineString, OGRLineString::getNumPoints(), OGRPolygon::getNumInteriorRings(), OGRLineString::getX(), OGRLineString::getY(), OGRLineString::addPoint(), OGRLineString::getZ(), and OGRPolygon::addRingDirectly().

Referenced by OGRBuildPolygonFromEdges().

14.12.2.197 void OGRCleanupAll (void)

Cleanup all OGR related resources.

This function will destroy the **OGRSFDriverRegistrar** (p. 263) along with all registered drivers, and then cleanup long lived OSR (**OGRSpatialReference** (p. 268)) 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. 552)

- CPLFinderClean()
- VSICleanupFileManager()
- CPLFreeConfig()
- CPLCleanupTLS()

References OGRCleanupAll(), and OSRCleanup().

Referenced by OGRCleanupAll().

14.12.2.198 OGRSFDriverH OGRGetDriver (int *iDriver*)

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriver()** (p. 264).

Parameters

<i>iDriver</i>	the driver index, from 0 to GetDriverCount()-1.
----------------	---

Returns

handle to the driver, or NULL if *iDriver* is out of range.

References OGRGetDriver(), and OGRSFDriverRegistrar::GetDriver().

Referenced by OGRGetDriver().

14.12.2.199 OGRSFDriverH OGRGetDriverByName (const char * *pszName*)

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverByName()** (p. 265)

Parameters

<i>pszName</i>	the driver name
----------------	-----------------

Returns

the driver, or NULL if no driver with that name is found

References OGRGetDriverByName(), OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::GetDriverByName().

Referenced by OGRGetDriverByName().

14.12.2.200 int OGRGetDriverCount (void)

Fetch the number of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverCount()** (p. 265).

Returns

the drivers count.

References OGRGetDriverCount(), and OGRSFDriverRegistrar::GetDriverCount().

Referenced by OGRGetDriverCount().

14.12.2.201 OGRDataSourceH OGRGetOpenDS (int iDS)

Return the iDS th datasource opened.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetOpenDS** (p. 265).

Parameters

<i>iDS</i>	the index of the dataset to return (between 0 and GetOpenDSCount() - 1)
------------	---

References OGRGetOpenDS(), OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::GetOpenDS().

Referenced by OGRGetOpenDS().

14.12.2.202 int OGRGetOpenDSCount (void)

Return the number of opened datasources.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetOpenDSCount()** (p. 266)

Returns

the number of opened datasources.

References OGRGetOpenDSCount(), OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::GetOpenDSCount().

Referenced by OGRGetOpenDSCount().

14.12.2.203 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. 263) function, not do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 263) instance.

If this function fails, **CPLGetLastErrorMsg()** (p. 381) can be used to check if there is an error message explaining why.

This function is the same as the C++ method **OGRSFDriverRegistrar::Open()** (p. 266).

Parameters

<i>pszName</i>	the name of the file, or data source to open.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>pahDriver-List</i>	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. 105). This **OGRDataSource** (p. 105) 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().

14.12.2.204 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. 267).

Parameters

<i>hDriver</i>	handle to the driver to add.
----------------	------------------------------

References OGRRegisterDriver(), OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::RegisterDriver().

Referenced by OGRRegisterDriver().

14.12.2.205 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. 112)

Parameters

<i>hDS</i>	handle to the data source to release
------------	--------------------------------------

Returns

OGRERR_NONE on success or an error code.

References OGRReleaseDataSource(), and OGRSFDriverRegistrar::GetRegistrar().

Referenced by OGRReleaseDataSource().

14.12.2.206 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().

14.13 ogr_core.h File Reference

```
#include "cpl_port.h" #include "gdal_version.h"
```

Classes

- class **OGREnvelope**
- union **OGRField**

Defines

- #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)

14.13.1 Detailed Description

Core portability services for cross-platform OGR code.

14.13.2 Define Documentation

14.13.2.1 `#define GDAL_CHECK_VERSION(pszCallingComponentName
) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR,
 pszCallingComponentName)`

Helper macro for GDALCheckVersion

14.13.3 Typedef Documentation

14.13.3.1 `typedef enum ogr_style_tool_param_brush_id OGRSTBrushParam`

List of parameters for use with **OGRStyleBrush** (p. 329).

14.13.3.2 `typedef enum ogr_style_tool_class_id OGRSTClassId`

OGRStyleTool (p. 338) derived class types (returned by GetType()).

14.13.3.3 `typedef enum ogr_style_tool_param_label_id OGRSTLabelParam`

List of parameters for use with **OGRStyleLabel** (p. 329).

14.13.3.4 `typedef enum ogr_style_tool_param_pen_id OGRSTPenParam`

List of parameters for use with **OGRStylePen** (p. 334).

14.13.3.5 `typedef enum ogr_style_tool_param_symbol_id OGRSTSymbolParam`

List of parameters for use with **OGRStyleSymbol** (p. 334).

14.13.3.6 `typedef enum ogr_style_tool_units_id OGRSTUnitId`

List of units supported by OGRStyleTools.

14.13.4 Enumeration Type Documentation

14.13.4.1 `enum ogr_style_tool_class_id`

OGRStyleTool (p. 338) derived class types (returned by GetType()).

14.13.4.2 enum `ogr_style_tool_param_brush_id`

List of parameters for use with **OGRStyleBrush** (p. 329).

14.13.4.3 enum `ogr_style_tool_param_label_id`

List of parameters for use with **OGRStyleLabel** (p. 329).

14.13.4.4 enum `ogr_style_tool_param_pen_id`

List of parameters for use with **OGRStylePen** (p. 334).

14.13.4.5 enum `ogr_style_tool_param_symbol_id`

List of parameters for use with **OGRStyleSymbol** (p. 334).

14.13.4.6 enum `ogr_style_tool_units_id`

List of units supported by OGRStyleTools.

14.13.4.7 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

14.13.4.8 enum `OGRJustification`

Display justification for field values.

14.13.4.9 enum OGRwkbGeometryType

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGRGeometry::getGeometryType()** (p. 163) 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

14.13.5 Function Documentation

14.13.5.1 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 primarily intended for external plugins.

Parameters

<i>nVersion-Major</i>	Major version to be tested against
<i>nVersion-Minor</i>	Minor version to be tested against
<i>pszCalling-Component-Name</i>	If not NULL, in case of version mismatch, the method will issue a failure mentioning the name of the calling component.

14.13.5.2 `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

<i>eType</i>	the geometry type.
--------------	--------------------

Returns

internal human readable string, or NULL on failure.

References OGRGeometryTypeToName(), wkbUnknown, wkbPoint, wkbPoint25D, wkbLineString, wkbLineString25D, wkbPolygon, wkbPolygon25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbGeometryCollection, wkbGeometryCollection25D, and wkbNone.

Referenced by OGRGeometryTypeToName().

14.13.5.3 `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 encountered will establish the preliminary type.

Parameters

<i>eMain</i>	the first input geometry type.
<i>eExtra</i>	the second input geometry type.

Returns

the merged geometry type.

References OGRMergeGeometryTypes(), wkbUnknown, wkbNone, wkbGeometryCollection, wkbMultiPoint, wkbMultiLineString, and wkbMultiPolygon.

Referenced by OGRMergeGeometryTypes().

14.13.5.4 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

<i>pszInput</i>	the input date string.
<i>psField</i>	the OGRField (p. 139) that will be updated with the parsed result.
<i>nOptions</i>	parsing options, for now always 0.

Returns

TRUE if apparently successful or FALSE on failure.

References OGRParseDate().

Referenced by OGRFeature::SetField(), and OGRParseDate().

14.14 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**

14.14.1 Detailed Description

Simple feature classes.

14.15 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**

14.15.1 Detailed Description

Simple feature style classes.

14.16 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**

14.16.1 Detailed Description

Simple feature geometry classes.

14.17 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)**

14.17.1 Detailed Description

Coordinate systems services.

14.17.2 Function Documentation

14.17.2.1 OGRCoordinateTransformation* OGRCreateCoordinateTransformation (OGRSpatialReference * poSource, OGRSpatialReference * poTarget)

Create transformation object.

This is the same as the C function OCTNewCoordinateTransformation().

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

The delete operator, or **OCTDestroyCoordinateTransformation()** (p. 550) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

Parameters

<i>poSource</i>	source spatial reference system.
<i>poTarget</i>	target spatial reference system.

Returns

NULL on failure or a ready to use transformation object.

References `OGRCreateCoordinateTransformation()`.

Referenced by `OGRCreateCoordinateTransformation()`, and `OGRGeometry::transformTo()`.

14.18 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. 268) object.*
- OGRSpatialReferenceH CPL_STDCALL **OSRClone** (OGRSpatialReferenceH)
*Make a duplicate of this **OGRSpatialReference** (p. 268).*
- void CPL_STDCALL **OSRDestroySpatialReference** (OGRSpatialReferenceH)
***OGRSpatialReference** (p. 268) 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 **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 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 **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 **OSRSetLinearUnitsAndUpdateParameters** (OGRSpatialReferenceH, const char *, double)
Set the linear units for the projection.
- double **OSRGetLinearUnits** (OGRSpatialReferenceH, 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 **OSRIsSameGeogCS** (OGRSpatialReferenceH, OGRSpatialReferenceH)
Do the GeogCS'es match?
- int **OSRIsSame** (OGRSpatialReferenceH, OGRSpatialReferenceH)
Do these two spatial references describe the same system ?
- OGRErr **OSRSetLocalCS** (OGRSpatialReferenceH hSRS, const char *psz-Name)
Set the user visible LOCAL_CS name.
- OGRErr **OSRSetProjCS** (OGRSpatialReferenceH hSRS, const char *psz-Name)
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. 268).*

- OGRErr **OSRSetTOWGS84** (OGRSpatialReferenceH hSRS, double, double, double, double, double, double, double, double, 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 **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.
- 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. 268)::*
- OGRErr **OSRSetUTM** (OGRSpatialReferenceH hSRS, int nZone, int bNorth)
Set UTM projection definition.
- int **OSRGetUTMZone** (OGRSpatialReferenceH hSRS, int *pbNorth)
Get utm zone information.
- 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 **OSRSetEquiarectangular** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquiarectangular2** (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 **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.
- void CPL_STDCALL **OCTDestroyCoordinateTransformation** (OGRCoordinateTransformationH)
OGRCoordinateTransformation (p. 99) destructor.
- char ** **OPTGetProjectionMethods** ()
- char ** **OPTGetParameterList** (const char *pszProjectionMethod, char **ppszUserName)
- int **OPTGetParameterInfo** (const char *pszProjectionMethod, const char *pszParameterName, char **ppszUserName, char **ppszType, double *pdfDefaultValue)

14.18.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr_spatialref.h** (p. 543)

14.18.2 Function Documentation

14.18.2.1 void CPL_STDCALL OCTDestroyCoordinateTransformation (OGRCoordinateTransformationH *hCT*)

OGRCoordinateTransformation (p. 99) destructor.

This function is the same as **OGRCoordinateTransformation::DestroyCT()** (p. 100)

Parameters

<i>hCT</i>	the object to delete
------------	----------------------

References OCTDestroyCoordinateTransformation().

Referenced by OCTDestroyCoordinateTransformation().

14.18.2.2 `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

<i>pszProjectionMethod</i>	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 "Traverse_Mercator".
<i>pszParameterName</i>	name of the parameter to fetch information about. This is the internal name such as "central_meridian" (SRS_PP_CENTRAL_MERIDIAN).
<i>ppszUserName</i>	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.
<i>ppszType</i>	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.
<i>pdfDefaultValue</i>	location at which to put the default value for this parameter. The pointer may be NULL.

Returns

TRUE if parameter found, or FALSE otherwise.

14.18.2.3 `char** OPTGetParameterList (const char * pszProjectionMethod, char ** ppszUserName)`

Fetch the parameters for a given projection method.

Parameters

<i>pszProjectionMethod</i>	internal name of projection methods to fetch the parameters for, such as "Traverse_Mercator" (SRS_PT_TRANSVERSE_MERCATOR).
<i>ppszUserName</i>	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.

14.18.2.4 `char** OPTGetProjectionMethods ()`

Fetch list of possible projection methods.

Returns

Returns NULL terminated list of projection methods. This should be freed with **CSLDestroy()** (p. 416) when no longer needed.

14.18.2.5 `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().

14.18.2.6 `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().

14.18.2.7 `OGRSpatialReferenceH CPL_STDCALL OSRClone (OGRSpatialReferenceH hSRS)`

Make a duplicate of this **OGRSpatialReference** (p. 268).

This function is the same as **OGRSpatialReference::Clone()** (p. 276)

References OSRClone().

Referenced by OSRClone().

14.18.2.8 `OGRSpatialReferenceH CPL_STDCALL OSRCloneGeogCS (OGRSpatialReferenceH hSource)`

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. 268) object.

This function is the same as **OGRSpatialReference::CloneGeogCS()** (p. 276).

References OSRCloneGeogCS().

Referenced by OSRCloneGeogCS().

14.18.2.9 **OGRErr** OSRCopyGeogCSFrom (**OGRSpatialReferenceH** *hSRS*,
OGRSpatialReferenceH *hSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. 268).

This function is the same as **OGRSpatialReference::CopyGeogCSFrom()** (p. 276)

References OSRCopyGeogCSFrom().

Referenced by OSRCopyGeogCSFrom().

14.18.2.10 **int** OSRDereference (**OGRSpatialReferenceH** *hSRS*)

Decrements the reference count by one.

This function is the same as **OGRSpatialReference::Dereference()** (p. 276)

References OSRDereference().

Referenced by OSRDereference().

14.18.2.11 **void** CPL_STDCALL OSRDestroySpatialReference (**OGRSpatialReferenceH** *hSRS*)

OGRSpatialReference (p. 268) destructor.

This function is the same as **OGRSpatialReference::~~OGRSpatialReference()** (p. 275) and **OGRSpatialReference::DestroySpatialReference()** (p. 277)

Parameters

<i>hSRS</i>	the object to delete
-------------	----------------------

References OSRDestroySpatialReference().

Referenced by OSRDestroySpatialReference().

14.18.2.12 **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. 278)

References OSRExportToMICoordSys().

Referenced by OSRExportToMICoordSys().

14.18.2.13 **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. 279).

References OSRExportToPCI().

Referenced by OSRExportToPCI().

14.18.2.14 OGRErr CPL_STDCALL OSRExportToPrettyWkt (OGRSpatialReferenceH hSRS, char ** ppszReturn, int bSimplify)

Convert this SRS into a nicely formatted WKT string for display to a person.

This function is the same as **OGRSpatialReference::exportToPrettyWkt()** (p. 279).

References OSRExportToPrettyWkt().

Referenced by OSRExportToPrettyWkt().

14.18.2.15 OGRErr CPL_STDCALL OSRExportToProj4 (OGRSpatialReferenceH hSRS, char ** ppszReturn)

Export coordinate system in PROJ.4 format.

This function is the same as **OGRSpatialReference::exportToProj4()** (p. 280).

References OSRExportToProj4().

Referenced by OSRExportToProj4().

14.18.2.16 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. 280).

References OSRExportToUSGS().

Referenced by OSRExportToUSGS().

14.18.2.17 OGRErr CPL_STDCALL OSRExportToWkt (OGRSpatialReferenceH hSRS, char ** ppszReturn)

Convert this SRS into WKT format.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. 281).

References OSRExportToWkt().

Referenced by OSRExportToWkt().

14.18.2.18 `OGRErr OSRExportToXML (OGRSpatialReferenceH hSRS, char ** ppszRawXML, const char * pszDialect)`

Export coordinate system in XML format.

This function is the same as **OGRSpatialReference::exportToXML()** (p. 281).

References OSRExportToXML().

Referenced by OSRExportToXML().

14.18.2.19 `OGRErr OSRFixup (OGRSpatialReferenceH hSRS)`

Fixup as needed.

This function is the same as **OGRSpatialReference::Fixup()** (p. 282).

References OSRFixup().

Referenced by OSRFixup().

14.18.2.20 `OGRErr OSRFixupOrdering (OGRSpatialReferenceH hSRS)`

Correct parameter ordering to match CT Specification.

This function is the same as **OGRSpatialReference::FixupOrdering()** (p. 283).

References OSRFixupOrdering().

Referenced by OSRFixupOrdering().

14.18.2.21 `double OSRGetAngularUnits (OGRSpatialReferenceH hSRS, char ** ppszName)`

Fetch angular geographic coordinate system units.

This function is the same as **OGRSpatialReference::GetAngularUnits()** (p. 283)

References OSRGetAngularUnits().

Referenced by OSRGetAngularUnits().

14.18.2.22 `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. 284)

References OSRGetAttrValue().

Referenced by OSRGetAttrValue().

14.18.2.23 `const char* OSRGetAuthorityCode (OGRSpatialReferenceH hSRS, const char * pszTargetKey)`

Get the authority code for a node.

This function is the same as **OGRSpatialReference::GetAuthorityCode()** (p. 285).

References OSRGetAuthorityCode().

Referenced by OSRGetAuthorityCode().

14.18.2.24 `const char* OSRGetAuthorityName (OGRSpatialReferenceH hSRS, const char * pszTargetKey)`

Get the authority name for a node.

This function is the same as **OGRSpatialReference::GetAuthorityName()** (p. 285).

References OSRGetAuthorityName().

Referenced by OSRGetAuthorityName().

14.18.2.25 `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. 286)

References OSRGetAxis().

Referenced by OSRGetAxis().

14.18.2.26 `double OSRGetInvFlattening (OGRSpatialReferenceH hSRS, OGRErr * pnErr)`

Get spheroid inverse flattening.

This function is the same as **OGRSpatialReference::GetInvFlattening()** (p. 287)

References OSRGetInvFlattening().

Referenced by OSRGetInvFlattening().

14.18.2.27 `double OSRGetLinearUnits (OGRSpatialReferenceH hSRS, char ** ppszName)`

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetLinearUnits()** (p. 287)

References OSRGetLinearUnits().

Referenced by OSRGetLinearUnits().

14.18.2.28 double OSRGetNormProjParm (OGRSpatialReferenceH *hSRS*, const char * *pszName*, double *dfDefaultValue*, OGRErr * *pnErr*)

This function is the same as **OGRSpatialReference** (p. 268)::

This function is the same as **OGRSpatialReference::GetNormProjParm()** (p. 288)

References OSRGetNormProjParm().

Referenced by OSRGetNormProjParm().

14.18.2.29 double OSRGetPrimeMeridian (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch prime meridian info.

This function is the same as **OGRSpatialReference::GetPrimeMeridian()** (p. 288)

References OSRGetPrimeMeridian().

Referenced by OSRGetPrimeMeridian().

14.18.2.30 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. 289)

References OSRGetProjParm().

Referenced by OSRGetProjParm().

14.18.2.31 double OSRGetSemiMajor (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid semi major axis.

This function is the same as **OGRSpatialReference::GetSemiMajor()** (p. 289)

References OSRGetSemiMajor().

Referenced by OSRGetSemiMajor().

14.18.2.32 double OSRGetSemiMinor (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid semi minor axis.

This function is the same as **OGRSpatialReference::GetSemiMinor()** (p. 290)

References OSRGetSemiMinor().

Referenced by OSRGetSemiMinor().

14.18.2.33 `OGRErr OSRGetTOWGS84 (OGRSpatialReferenceH hSRS, double * pdfCoeff, int nCoeffCount)`

Fetch TOWGS84 parameters, if available.

This function is the same as **OGRSpatialReference::GetTOWGS84()** (p. 290).

References OSRGetTOWGS84().

Referenced by OSRGetTOWGS84().

14.18.2.34 `int OSRGetUTMZone (OGRSpatialReferenceH hSRS, int * pbNorth)`

Get utm zone information.

This is the same as the C++ method **OGRSpatialReference::GetUTMZone()** (p. 291)

References OSRGetUTMZone().

Referenced by OSRGetUTMZone().

14.18.2.35 `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. 292).

14.18.2.36 `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. 292).

14.18.2.37 `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. 293)

References OSRImportFromESRI().

Referenced by OSRImportFromESRI().

14.18.2.38 `OGRErr OSRImportFromMICoordSys (OGRSpatialReferenceH hSRS, const char * pszCoordSys)`

Import Mapinfo style CoordSys definition.

This method is the equivalent of the C++ method **OGRSpatialReference::importFromMICoordSys** (p. 294)

References OSRImportFromMICoordSys().

Referenced by OSRImportFromMICoordSys().

14.18.2.39 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. 296).

References OSRImportFromPCI().

Referenced by OSRImportFromPCI().

14.18.2.40 OGRErr OSRImportFromProj4 (OGRSpatialReferenceH hSRS, const char * pszProj4)

Import PROJ.4 coordinate string.

This function is the same as **OGRSpatialReference::importFromProj4()** (p. 297).

References OSRImportFromProj4().

Referenced by OSRImportFromProj4().

14.18.2.41 OGRErr OSRImportFromUrl (OGRSpatialReferenceH hSRS, const char * pszUrl)

Set spatial reference from a URL.

This function is the same as **OGRSpatialReference::importFromUrl()** (p. 298)

References OSRImportFromUrl().

Referenced by OSRImportFromUrl().

14.18.2.42 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. 299).

References OSRImportFromUSGS().

Referenced by OSRImportFromUSGS().

14.18.2.43 OGRErr OSRImportFromWkt (OGRSpatialReferenceH hSRS, char ** ppszInput)

Import from WKT string.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. 304).

References OSRImportFromWkt().

Referenced by OSRImportFromWkt().

14.18.2.44 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. 305).

References OSRImportFromXML().

Referenced by OSRImportFromXML().

14.18.2.45 int OSRIsGeographic (OGRSpatialReferenceH *hSRS*)

Check if geographic coordinate system.

This function is the same as **OGRSpatialReference::IsGeographic()** (p. 305).

References OSRIsGeographic().

Referenced by OSRIsGeographic().

14.18.2.46 int OSRIsLocal (OGRSpatialReferenceH *hSRS*)

Check if local coordinate system.

This function is the same as **OGRSpatialReference::IsLocal()** (p. 305).

References OSRIsLocal().

Referenced by OSRIsLocal().

14.18.2.47 int OSRIsProjected (OGRSpatialReferenceH *hSRS*)

Check if projected coordinate system.

This function is the same as **OGRSpatialReference::IsProjected()** (p. 306).

References OSRIsProjected().

Referenced by OSRIsProjected().

14.18.2.48 int OSRIsSame (OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2*)

Do these two spatial references describe the same system ?

This function is the same as **OGRSpatialReference::IsSame()** (p. 306).

References OSRIsSame().

Referenced by OSRIsSame().

14.18.2.49 `int OSRIsSameGeogCS (OGRSpatialReferenceH hSRS1, OGRSpatialReferenceH hSRS2)`

Do the GeogCS'es match?

This function is the same as **OGRSpatialReference::IsSameGeogCS()** (p. 306).

References OSRIsSameGeogCS().

Referenced by OSRIsSameGeogCS().

14.18.2.50 `OGRERR OSRMorphFromESRI (OGRSpatialReferenceH hSRS)`

Convert in place from ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphFromESRI()** (p. 307)

References OSRMorphFromESRI().

Referenced by OSRMorphFromESRI().

14.18.2.51 `OGRERR OSRMorphToESRI (OGRSpatialReferenceH hSRS)`

Convert in place to ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphToESRI()** (p. 307)

References OSRMorphToESRI().

Referenced by OSRMorphToESRI().

14.18.2.52 `OGRSpatialReferenceH CPL_STDCALL OSRNewSpatialReference (const char * pszWKT)`

Constructor.

This function is the same as **OGRSpatialReference::OGRSpatialReference()**

References OSRNewSpatialReference(), and **OGRSpatialReference::importFromWkt()**.

Referenced by OSRNewSpatialReference().

14.18.2.53 `int OSRReference (OGRSpatialReferenceH hSRS)`

Increments the reference count by one.

This function is the same as **OGRSpatialReference::Reference()** (p. 308)

References OSRReference().

Referenced by OSRReference().

14.18.2.54 void OSRRelease (OGRSpatialReferenceH *hSRS*)

Decrements the reference count by one, and destroy if zero.

This function is the same as **OGRSpatialReference::Release()** (p. 308)

References OSRRelease().

Referenced by OSRRelease().

14.18.2.55 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().

14.18.2.56 OGRErr OSRSetAE (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Azimuthal Equidistant

References OSRSetAE().

Referenced by OSRSetAE().

14.18.2.57 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. 308)

References OSRSetAngularUnits().

Referenced by OSRSetAngularUnits().

14.18.2.58 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. 319)

References OSRSetAttrValue().

Referenced by OSRSetAttrValue().

14.18.2.59 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. 309).

References OSRSetAuthority().

Referenced by OSRSetAuthority().

14.18.2.60 OGRErr OSRSetBonne (OGRSpatialReferenceH *hSRS*, double *dfStandardParallel*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Bonne

References OSRSetBonne().

Referenced by OSRSetBonne().

14.18.2.61 OGRErr OSRSetCEA (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cylindrical Equal Area

References OSRSetCEA().

Referenced by OSRSetCEA().

14.18.2.62 OGRErr OSRSetCS (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cassini-Soldner

References OSRSetCS().

Referenced by OSRSetCS().

14.18.2.63 OGRErr OSRSetEC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equidistant Conic

References OSRSetEC().

Referenced by OSRSetEC().

14.18.2.64 OGRErr OSRSetEckert (OGRSpatialReferenceH *hSRS*, int *nVariation*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert I-VI

References OSRSetEckert().

Referenced by OSRSetEckert().

14.18.2.65 OGRErr OSRSetEckertIV (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert IV

References OSRSetEckertIV().

Referenced by OSRSetEckertIV().

14.18.2.66 OGRErr OSRSetEckertVI (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert VI

References OSRSetEckertVI().

Referenced by OSRSetEckertVI().

14.18.2.67 OGRErr OSRSetEquirectangular (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular

References OSRSetEquirectangular().

Referenced by OSRSetEquirectangular().

14.18.2.68 OGRErr OSRSetEquirectangular2 (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfPseudoStdParallel1*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular generalized form

References OSRSetEquirectangular2().

Referenced by OSRSetEquirectangular2().

14.18.2.69 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. 312)

References OSRSetFromUserInput().

Referenced by OSRSetFromUserInput().

14.18.2.70 **OGRErr OSRSetGaussSchreiberTMercator** (**OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References OSRSetGaussSchreiberTMercator().

Referenced by OSRSetGaussSchreiberTMercator().

14.18.2.71 **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. 313)

References OSRSetGeogCS().

Referenced by OSRSetGeogCS().

14.18.2.72 **OGRErr OSRSetGEOS** (**OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing*)

GEOS - Geostationary Satellite View

References OSRSetGEOS().

Referenced by OSRSetGEOS().

14.18.2.73 **OGRErr OSRSetGH** (**OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Goode Homolosine

References OSRSetGH().

Referenced by OSRSetGH().

14.18.2.74 **OGRErr OSRSetGnomonic** (**OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gnomonic

References OSRSetGnomonic().

Referenced by OSRSetGnomonic().

14.18.2.75 **OGRErr OSRSetGS** (**OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gall Stereographic

References OSRSetGS().

Referenced by OSRSetGS().

14.18.2.76 **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. 315)

References OSRSetHOM().

Referenced by OSRSetHOM().

14.18.2.77 **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. 315)

References OSRSetHOM2PNO().

Referenced by OSRSetHOM2PNO().

14.18.2.78 **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().

14.18.2.79 **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().

14.18.2.80 OGRErr OSRSetLAEA (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References OSRSetLAEA().

Referenced by OSRSetLAEA().

14.18.2.81 OGRErr OSRSetLCC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic

References OSRSetLCC().

Referenced by OSRSetLCC().

14.18.2.82 OGRErr OSRSetLCC1SP (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References OSRSetLCC1SP().

Referenced by OSRSetLCC1SP().

14.18.2.83 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().

14.18.2.84 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. 317)

References OSRSetLinearUnits().

Referenced by OSRSetLinearUnits().

14.18.2.85 **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. 317)

References OSRSetLinearUnitsAndUpdateParameters().

Referenced by OSRSetLinearUnitsAndUpdateParameters().

14.18.2.86 **OGRERR** OSRSetLocalCS (OGRSpatialReferenceH *hSRS*, const char * *pszName*)

Set the user visible LOCAL_CS name.

This function is the same as **OGRSpatialReference::SetLocalCS()** (p. 318)

References OSRSetLocalCS().

Referenced by OSRSetLocalCS().

14.18.2.87 **OGRERR** OSRSetMC (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double
dfCenterLong, double *dfFalseEasting*, double *dfFalseNorthing*)

Miller Cylindrical

References OSRSetMC().

Referenced by OSRSetMC().

14.18.2.88 **OGRERR** OSRSetMercator (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*,
double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double
dfFalseNorthing)

Mercator

References OSRSetMercator().

Referenced by OSRSetMercator().

14.18.2.89 **OGRERR** OSRSetMollweide (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*,
double *dfFalseEasting*, double *dfFalseNorthing*)

Mollweide

References OSRSetMollweide().

Referenced by OSRSetMollweide().

14.18.2.90 **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. 320)

References OSRSetNormProjParm().

Referenced by OSRSetNormProjParm().

14.18.2.91 **OGRErr OSRSetNZMG** (**OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

New Zealand Map Grid

References OSRSetNZMG().

Referenced by OSRSetNZMG().

14.18.2.92 **OGRErr OSRSetOrthographic** (**OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Orthographic

References OSRSetOrthographic().

Referenced by OSRSetOrthographic().

14.18.2.93 **OGRErr OSRSetOS** (**OGRSpatialReferenceH** *hSRS*, double *dfOriginLat*, double *dfCMeridian*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Oblique Stereographic

References OSRSetOS().

Referenced by OSRSetOS().

14.18.2.94 **OGRErr OSRSetPolyconic** (**OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polyconic

References OSRSetPolyconic().

Referenced by OSRSetPolyconic().

14.18.2.95 **OGRErr OSRSetProjCS** (**OGRSpatialReferenceH** *hSRS*, const char * *pszName*)

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetProjCS()** (p. 321)

References OSRSetProjCS().

Referenced by OSRSetProjCS().

14.18.2.96 `OGRErr OSRSetProjection (OGRSpatialReferenceH hSRS, const char *
pszProjection)`

Set a projection name.

This function is the same as **OGRSpatialReference::SetProjection()** (p. 321)

References OSRSetProjection().

Referenced by OSRSetProjection().

14.18.2.97 `OGRErr OSRSetProjParm (OGRSpatialReferenceH hSRS, const char *
pszParmName, double dfValue)`

Set a projection parameter value.

This function is the same as **OGRSpatialReference::SetProjParm()** (p. 322)

References OSRSetProjParm().

Referenced by OSRSetProjParm().

14.18.2.98 `OGRErr OSRSetPS (OGRSpatialReferenceH hSRS, double dfCenterLat, double
dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)`

Polar Stereographic

References OSRSetPS().

Referenced by OSRSetPS().

14.18.2.99 `OGRErr OSRSetRobinson (OGRSpatialReferenceH hSRS, double dfCenterLong,
double dfFalseEasting, double dfFalseNorthing)`

Robinson

References OSRSetRobinson().

Referenced by OSRSetRobinson().

14.18.2.100 `OGRErr OSRSetSinusoidal (OGRSpatialReferenceH hSRS, double dfCenterLong,
double dfFalseEasting, double dfFalseNorthing)`

Sinusoidal

References OSRSetSinusoidal().

Referenced by OSRSetSinusoidal().

14.18.2.101 OGRErr OSRSetSOC (OGRSpatialReferenceH *hSRS*, double *dfLatitudeOfOrigin*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References OSRSetSOC().

Referenced by OSRSetSOC().

14.18.2.102 OGRErr OSRSetStereographic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Stereographic

References OSRSetStereographic().

Referenced by OSRSetStereographic().

14.18.2.103 OGRErr OSRSetTM (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator

References OSRSetTM().

Referenced by OSRSetTM().

14.18.2.104 OGRErr OSRSetTMG (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Tunesia Mining Grid

References OSRSetTMG().

Referenced by OSRSetTMG().

14.18.2.105 OGRErr OSRSetTMSO (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References OSRSetTMSO().

Referenced by OSRSetTMSO().

14.18.2.106 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().

14.18.2.107 `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. 325).

References OSRSetTOWGS84().

Referenced by OSRSetTOWGS84().

14.18.2.108 `OGRErr OSRSetUTM (OGRSpatialReferenceH hSRS, int nZone, int bNorth)`

Set UTM projection definition.

This is the same as the C++ method **OGRSpatialReference::SetUTM()** (p. 326)

References OSRSetUTM().

Referenced by OSRSetUTM().

14.18.2.109 `OGRErr OSRSetVDG (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)`

VanDerGrinten

References OSRSetVDG().

Referenced by OSRSetVDG().

14.18.2.110 `OGRErr OSRSetWagner (OGRSpatialReferenceH hSRS, int nVariation, double dfFalseEasting, double dfFalseNorthing)`

Wagner I -- VII

14.18.2.111 `OGRErr OSRSetWellKnownGeogCS (OGRSpatialReferenceH hSRS, const char * pszName)`

Set a GeogCS based on well known name.

This function is the same as **OGRSpatialReference::SetWellKnownGeogCS()** (p. 327)

References OSRSetWellKnownGeogCS().

Referenced by OSRSetWellKnownGeogCS().

14.18.2.112 OGRErr OSRStripCTParms (OGRSpatialReferenceH hSRS)

Strip OGC CT Parameters.

This function is the same as **OGRSpatialReference::StripCTParms()** (p. 328).

References OSRStripCTParms().

Referenced by OSRStripCTParms().

14.18.2.113 OGRErr OSRValidate (OGRSpatialReferenceH hSRS)

Validate SRS tokens.

This function is the same as the C++ method **OGRSpatialReference::Validate()** (p. 328).

References OSRValidate().

Referenced by OSRValidate().

14.19 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.

14.19.1 Detailed Description

Classes related to registration of format support, and opening datasets.