SDTS_AL

Contents

1 ISO8211Lib **1**

2 sdts_al_main **5**

3 SDTS_AL_TUT **9**

4 sdts2shp.cpp **17**

5 ISO8211_Example **27**

6 Deprecated List **31**

7 Hierarchical Index **33**

7.1 Class Hierarchy 33

8 Class Index **37**

8.1 Class List 37

9 File Index **41**

9.1 File List 41

10 Class Documentation 43

10.1 _CPL_SHA256Context Struct Reference . . . 43

10.2 _CPLHashSet Struct Reference . . . 43

10.3 _CPLList Struct Reference . . . 43

10.3.1 Detailed Description . . . 43

10.3.2 Member Data Documentation . . . 43

10.3.2.1 pData . . . 44

10.3.2.2 psNext . . . 44

10.4 _CPLLock Struct Reference . . . 44

10.5 _CPLQuadTree Struct Reference . . . 44

10.6 _CPLSpawnedProcess Struct Reference . . . 44

10.7 _CPLSpinLock Struct Reference . . . 44

10.8 _MutexLinkedElt Struct Reference . . . 45

10.9 _QuadTreeNode Struct Reference . . . 45

10.10 CPL_SHA1Context Struct Reference . . . 45

10.11 CPLErrorContext Struct Reference . . . 45

10.12 CPLHTTPResult Struct Reference . . . 45

10.12.1 Detailed Description . . . 46

10.12.2 Member Data Documentation . . . 46

10.12.2.1 nDataAlloc . . . 46

10.12.2.2 nDataLen . . . 46

10.12.2.3 nMimePartCount . . . 46

10.12.2.4 nStatus . . . 46

10.12.2.5 pabyData . . . 46

10.12.2.6 papszHeaders . . . 46

10.12.2.7 pasMimePart . . . 47

10.12.2.8 pszContentType . . . 47

10.12.2.9 pszErrBuf . . . 47

10.13 CPLJSONArray Class Reference . . . 47

10.13.1 Detailed Description . . . 48

10.13.2 Member Function Documentation . . . 48
10.13.2.1 Add() [1/7] . . . 48
10.13.2.2 Add() [2/7] . . . 48
10.13.2.3 Add() [3/7] . . . 49
10.13.2.4 Add() [4/7] . . . 49
10.13.2.5 Add() [5/7] . . . 49
10.13.2.6 Add() [6/7] . . . 50
10.13.2.7 Add() [7/7] . . . 50
10.13.2.8 operator[]() [1/2] . . . 50
10.13.2.9 operator[]() [2/2] . . . 52
10.13.2.10 Size() . . . 52
10.14 CPLJSONDocument Class Reference . . . 53
10.14.1 Detailed Description . . . 53
10.14.2 Member Function Documentation . . . 53
10.14.2.1 GetRoot() . . . 53
10.14.2.2 Load() . . . 53
10.14.2.3 LoadChunks() . . . 54
10.14.2.4 LoadMemory() [1/2] . . . 54
10.14.2.5 LoadMemory() [2/2] . . . 55
10.14.2.6 LoadUrl() . . . 55
10.14.2.7 Save() . . . 56
10.14.2.8 SaveAsString() . . . 56
10.15 CPLJSONObject Class Reference . . . 57
10.15.1 Detailed Description . . . 58
10.15.2 Member Enumeration Documentation . . . 58
10.15.2.1 PrettyFormat . . . 58
10.15.2.2 Type . . . 59
10.15.3 Member Function Documentation . . . 59
10.15.3.1 Add() [1/8] . . . 59
10.15.3.2 Add() [2/8] . . . 59

10.15.3.3 Add() [3/8] 60
10.15.3.4 Add() [4/8] 60
10.15.3.5 Add() [5/8] 60
10.15.3.6 Add() [6/8] 61
10.15.3.7 Add() [7/8] 61
10.15.3.8 Add() [8/8] 62
10.15.3.9 AddNull() 62
10.15.3.10 Deinit() 62
10.15.3.11 Delete() 62
10.15.3.12 Format() 63
10.15.3.13 GetArray() 63
10.15.3.14 GetBool() 64
10.15.3.15 GetChildren() 64
10.15.3.16 GetDouble() 64
10.15.3.17 GetInteger() 65
10.15.3.18 GetLong() 65
10.15.3.19 GetObj() 66
10.15.3.20 GetString() 66
10.15.3.21 GetType() 67
10.15.3.22 IsValid() 67
10.15.3.23 operator[]() 67
10.15.3.24 Set() [1/6] 68
10.15.3.25 Set() [2/6] 68
10.15.3.26 Set() [3/6] 69
10.15.3.27 Set() [4/6] 69
10.15.3.28 Set() [5/6] 69
10.15.3.29 Set() [6/6] 70
10.15.3.30 SetNull() 70
10.15.3.31 ToArray() 71
10.15.3.32 ToBool() 71

10.15.3.33 ToDouble() . . . 71
10.15.3.34 ToInteger() . . . 72
10.15.3.35 ToLong() . . . 72
10.15.3.36 ToString() . . . 73
10.16 CPLMD5Context Struct Reference . . . 73
10.17 CPLMimePart Struct Reference . . . 73
10.17.1 Detailed Description . . . 73
10.17.2 Member Data Documentation . . . 74
10.17.2.1 nDataLen . . . 74
10.17.2.2 pabyData . . . 74
10.17.2.3 papszHeaders . . . 74
10.18 CPLODBCDriverInstaller Class Reference . . . 74
10.18.1 Detailed Description . . . 74
10.18.2 Member Function Documentation . . . 75
10.18.2.1 GetLastError() . . . 75
10.18.2.2 GetLastErrorCode() . . . 75
10.18.2.3 GetPathOut() . . . 75
10.18.2.4 GetUsageCount() . . . 75
10.18.2.5 InstallDriver() . . . 75
10.18.2.6 RemoveDriver() . . . 76
10.19 CPLODBCSession Class Reference . . . 76
10.19.1 Detailed Description . . . 77
10.19.2 Constructor & Destructor Documentation . . . 77
10.19.2.1 CPLODBCSession() . . . 77
10.19.2.2 ∼CPLODBCSession() . . . 77
10.19.3 Member Function Documentation . . . 77
10.19.3.1 BeginTransaction() . . . 77
10.19.3.2 ClearTransaction() . . . 77
10.19.3.3 CloseSession() . . . 77
10.19.3.4 CommitTransaction() . . . 77

10.19.3.5 EstablishSession() 77
10.19.3.6 Failed() 78
10.19.3.7 GetConnection() 78
10.19.3.8 GetEnvironment() 78
10.19.3.9 GetLastError() 78
10.19.3.10 IsInTransaction() 79
10.19.3.11 RollbackTransaction() 79
10.20 CPLODBCStatement Class Reference 79
10.20.1 Detailed Description 80
10.20.2 Constructor & Destructor Documentation 80
10.20.2.1 CPLODBCStatement() 80
10.20.2.2 ~CPLODBCStatement() 80
10.20.3 Member Function Documentation 80
10.20.3.1 Append() [1/3] 80
10.20.3.2 Append() [2/3] 81
10.20.3.3 Append() [3/3] 81
10.20.3.4 AppendEscaped() 81
10.20.3.5 Appendf() 82
10.20.3.6 Clear() 82
10.20.3.7 ClearColumnData() 82
10.20.3.8 CollectResultsInfo() 82
10.20.3.9 DumpResult() 82
10.20.3.10 ExecuteSQL() 83
10.20.3.11 Fetch() 83
10.20.3.12 GetColColumnDef() 84
10.20.3.13 GetColCount() 84
10.20.3.14 GetColData() [1/2] 84
10.20.3.15 GetColData() [2/2] 85
10.20.3.16 GetColDataLength() 85
10.20.3.17 GetColId() 85

10.20.3.18 GetColName() . . . 86
10.20.3.19 GetColNullable() . . . 86
10.20.3.20 GetColPrecision() . . . 86
10.20.3.21 GetColSize() . . . 88
10.20.3.22 GetColType() . . . 88
10.20.3.23 GetColTypeName() . . . 89
10.20.3.24 GetColumns() . . . 89
10.20.3.25 GetCommand() . . . 89
10.20.3.26 GetPrimaryKeys() . . . 90
10.20.3.27 GetRowCountAffected() . . . 90
10.20.3.28 GetStatement() . . . 90
10.20.3.29 GetTables() . . . 90
10.20.3.30 GetTypeMapping() . . . 91
10.20.3.31 GetTypeName() . . . 91
10.21 CPLRectObj Struct Reference . . . 92
10.21.1 Detailed Description . . . 92
10.21.2 Member Data Documentation . . . 92
10.21.2.1 maxx . . . 92
10.21.2.2 maxy . . . 92
10.21.2.3 minx . . . 92
10.21.2.4 miny . . . 93
10.22 CPLSharedFileInfo Struct Reference . . . 93
10.22.1 Detailed Description . . . 93
10.22.2 Member Data Documentation . . . 93
10.22.2.1 bLarge . . . 93
10.22.2.2 fp . . . 93
10.22.2.3 nRefCount . . . 94
10.22.2.4 pszAccess . . . 94
10.22.2.5 pszFilename . . . 94
10.23 CPLSharedFileInfoExtra Struct Reference . . . 94

10.24 CPLStdCallThreadInfo Struct Reference . . . 94
10.25 CPLVirtualMem Struct Reference . . . 94
10.26 CPLWorkerThread Struct Reference . . . 95
10.27 CPLWorkerThreadJob Struct Reference . . . 95
10.28 CPLWorkerThreadPool Class Reference . . . 95
10.28.1 Detailed Description . . . 95
10.28.2 Constructor & Destructor Documentation . . . 95
10.28.2.1 CPLWorkerThreadPool() . . . 95
10.28.2.2 ~CPLWorkerThreadPool() . . . 96
10.28.3 Member Function Documentation . . . 96
10.28.3.1 GetThreadCount() . . . 96
10.28.3.2 Setup() . . . 96
10.28.3.3 SubmitJob() . . . 96
10.28.3.4 SubmitJobs() . . . 97
10.28.3.5 WaitCompletion() . . . 97
10.29 CPLXMLNode Struct Reference . . . 98
10.29.1 Detailed Description . . . 98
10.29.2 Member Data Documentation . . . 98
10.29.2.1 eType . . . 98
10.29.2.2 psChild . . . 98
10.29.2.3 psNext . . . 99
10.29.2.4 pszValue . . . 99
10.30 CPLZip Struct Reference . . . 99
10.31 ctb Struct Reference . . . 99
10.32 curfile_info Struct Reference . . . 100
10.33 DDFField Class Reference . . . 100
10.33.1 Detailed Description . . . 100
10.33.2 Member Function Documentation . . . 100
10.33.2.1 Dump() . . . 100
10.33.2.2 GetData() . . . 101

10.33.2.3 GetDataSize() . . . 101
10.33.2.4 GetFieldDefn() . . . 101
10.33.2.5 GetInstanceData() . . . 101
10.33.2.6 GetRepeatCount() . . . 101
10.33.2.7 GetSubfieldData() . . . 102
10.34 DDFFieldDefn Class Reference . . . 102
10.34.1 Detailed Description . . . 103
10.34.2 Member Function Documentation . . . 103
10.34.2.1 Dump() . . . 103
10.34.2.2 FindSubfieldDefn() . . . 103
10.34.2.3 GetDefaultValue() . . . 104
10.34.2.4 GetDescription() . . . 104
10.34.2.5 GetFixedWidth() . . . 104
10.34.2.6 GetName() . . . 104
10.34.2.7 GetSubfield() . . . 104
10.34.2.8 GetSubfieldCount() . . . 105
10.34.2.9 IsRepeating() . . . 105
10.34.2.10 SetRepeatingFlag() . . . 105
10.35 DDFModule Class Reference . . . 105
10.35.1 Detailed Description . . . 106
10.35.2 Constructor & Destructor Documentation . . . 106
10.35.2.1 DDFModule() . . . 106
10.35.2.2 ~DDFModule() . . . 106
10.35.3 Member Function Documentation . . . 106
10.35.3.1 AddField() . . . 106
10.35.3.2 Close() . . . 107
10.35.3.3 Dump() . . . 107
10.35.3.4 FindFieldDefn() . . . 107
10.35.3.5 GetField() . . . 108
10.35.3.6 GetFieldCount() . . . 108

10.35.3.7 Open() . . . 108
10.35.3.8 ReadRecord() . . . 108
10.35.3.9 Rewind() . . . 109
10.36 DDFRecord Class Reference . . . 109
10.36.1 Detailed Description . . . 110
10.36.2 Member Function Documentation . . . 110
10.36.2.1 AddField() . . . 110
10.36.2.2 Clone() . . . 110
10.36.2.3 CloneOn() . . . 111
10.36.2.4 CreateDefaultFieldInstance() . . . 111
10.36.2.5 DeleteField() . . . 112
10.36.2.6 Dump() . . . 112
10.36.2.7 FindField() . . . 112
10.36.2.8 GetData() . . . 113
10.36.2.9 GetDataSize() . . . 113
10.36.2.10 GetField() . . . 113
10.36.2.11 GetFieldCount() . . . 113
10.36.2.12 GetFloatSubfield() . . . 114
10.36.2.13 GetIntSubfield() . . . 114
10.36.2.14 GetModule() . . . 115
10.36.2.15 GetStringSubfield() . . . 115
10.36.2.16 ResizeField() . . . 115
10.36.2.17 SetFieldRaw() . . . 116
10.36.2.18 SetFloatSubfield() . . . 116
10.36.2.19 SetIntSubfield() . . . 117
10.36.2.20 SetStringSubfield() . . . 117
10.36.2.21 Write() . . . 118
10.37 DDFSubfieldDefn Class Reference . . . 118
10.37.1 Detailed Description . . . 119
10.37.2 Member Enumeration Documentation . . . 119

10.37.2.1 DDFBinaryFormat . . . 119
10.37.3 Member Function Documentation . . . 119
10.37.3.1 Dump() . . . 119
10.37.3.2 DumpData() . . . 119
10.37.3.3 ExtractFloatData() . . . 120
10.37.3.4 ExtractIntData() . . . 120
10.37.3.5 ExtractStringData() . . . 121
10.37.3.6 FormatFloatValue() . . . 122
10.37.3.7 FormatIntValue() . . . 122
10.37.3.8 FormatStringValue() . . . 122
10.37.3.9 GetDataLength() . . . 122
10.37.3.10 GetDefaultValue() . . . 123
10.37.3.11 GetFormat() . . . 123
10.37.3.12 GetName() . . . 124
10.37.3.13 GetType() . . . 124
10.37.3.14 GetWidth() . . . 124
10.38 DefaultCSVFileNameTLS Struct Reference . . . 124
10.39 errHandler Struct Reference . . . 124
10.40 file_in_zip_read_info_s Struct Reference . . . 125
10.41 FindFileTLS Struct Reference . . . 125
10.42 GDALScaledProgressInfo Struct Reference . . . 125
10.43 linkedlist_data_s Struct Reference . . . 125
10.44 linkedlist_datablock_internal_s Struct Reference . . . 125
10.45 ParseContext Struct Reference . . . 125
10.46 SDTS_CATD Class Reference . . . 125
10.46.1 Detailed Description . . . 126
10.46.2 Member Function Documentation . . . 126
10.46.2.1 GetEntryFilePath() . . . 126
10.46.2.2 GetEntryType() . . . 126
10.46.2.3 GetEntryTypeDesc() . . . 127

10.47 SDTS_CATDEntry Class Reference . . . 127
10.48 SDTS_IREF Class Reference . . . 127
10.48.1 Detailed Description . . . 127
10.49 SDTS_XREF Class Reference . . . 128
10.49.1 Detailed Description . . . 128
10.49.2 Member Data Documentation . . . 128
10.49.2.1 nZone . . . 128
10.49.2.2 pszDatum . . . 128
10.49.2.3 pszSystemName . . . 128
10.50 SDTSAttrReader Class Reference . . . 129
10.50.1 Detailed Description . . . 129
10.50.2 Member Function Documentation . . . 129
10.50.2.1 IsSecondary() . . . 129
10.51 SDTSAttrRecord Class Reference . . . 129
10.51.1 Detailed Description . . . 130
10.51.2 Member Function Documentation . . . 130
10.51.2.1 Dump() . . . 130
10.51.3 Member Data Documentation . . . 130
10.51.3.1 poATTR . . . 130
10.51.3.2 poWholeRecord . . . 130
10.52 SDTSDataset Class Reference . . . 131
10.53 SDTSFeature Class Reference . . . 131
10.53.1 Detailed Description . . . 131
10.53.2 Member Function Documentation . . . 132
10.53.2.1 Dump() . . . 132
10.53.3 Member Data Documentation . . . 132
10.53.3.1 nAttributes . . . 132
10.53.3.2 oModId . . . 132
10.53.3.3 paoATID . . . 132
10.54 SDTSIndexedReader Class Reference . . . 133

10.54.1 Detailed Description . . . 133
10.54.2 Member Function Documentation . . . 133
10.54.2.1 ClearIndex() . . . 133
10.54.2.2 FillIndex() . . . 133
10.54.2.3 GetIndexedFeatureRef() . . . 133
10.54.2.4 GetNextFeature() . . . 134
10.54.2.5 IsIndexed() . . . 134
10.54.2.6 Rewind() . . . 134
10.54.2.7 ScanModuleReferences() . . . 134
10.55 SDTSLineReader Class Reference . . . 135
10.55.1 Detailed Description . . . 135
10.55.2 Member Function Documentation . . . 135
10.55.2.1 AttachToPolygons() . . . 135
10.56 SDTSModId Class Reference . . . 136
10.56.1 Detailed Description . . . 136
10.56.2 Member Data Documentation . . . 136
10.56.2.1 nRecord . . . 136
10.56.2.2 szModule . . . 136
10.56.2.3 szName . . . 137
10.56.2.4 szOBRP . . . 137
10.57 SDTSPointReader Class Reference . . . 137
10.57.1 Detailed Description . . . 137
10.58 SDTSPolygonReader Class Reference . . . 138
10.58.1 Detailed Description . . . 138
10.58.2 Member Function Documentation . . . 138
10.58.2.1 AssembleRings() . . . 138
10.59 SDTSRasterBand Class Reference . . . 139
10.60 SDTSRasterReader Class Reference . . . 139
10.60.1 Detailed Description . . . 139
10.60.2 Member Function Documentation . . . 140

10.60.2.1 GetBlock() 140
10.60.2.2 GetBlockXSize() 140
10.60.2.3 GetBlockYSize() 140
10.60.2.4 GetMinMax() 140
10.60.2.5 GetRasterType() 141
10.60.2.6 GetTransform() 141
10.60.2.7 GetXSize() 142
10.60.2.8 GetYSize() 142
10.61 SDTSRawLine Class Reference 142
10.61.1 Detailed Description 143
10.61.2 Member Function Documentation 143
10.61.2.1 Dump() 143
10.61.3 Member Data Documentation 143
10.61.3.1 nVertices 143
10.61.3.2 oEndNode 143
10.61.3.3 oLeftPoly 144
10.61.3.4 oRightPoly 144
10.61.3.5 oStartNode 144
10.61.3.6 padfX 144
10.61.3.7 padfY 144
10.61.3.8 padfZ 144
10.62 SDTSRawPoint Class Reference 145
10.62.1 Detailed Description 145
10.62.2 Member Function Documentation 145
10.62.2.1 Dump() 145
10.62.3 Member Data Documentation 145
10.62.3.1 dfX 146
10.62.3.2 dfY 146
10.62.3.3 dfZ 146
10.62.3.4 oAreaId 146

10.63 SDTSRawPolygon Class Reference . . . 146
10.63.1 Detailed Description . . . 147
10.63.2 Member Function Documentation . . . 147
10.63.2.1 AssembleRings() . . . 147
10.63.2.2 Dump() . . . 148
10.63.3 Member Data Documentation . . . 148
10.63.3.1 nRings . . . 148
10.63.3.2 nVertices . . . 148
10.63.3.3 padfX . . . 148
10.63.3.4 padfY . . . 149
10.63.3.5 padfZ . . . 149
10.63.3.6 panRingStart . . . 149
10.64 SDTSTransfer Class Reference . . . 149
10.64.1 Detailed Description . . . 149
10.64.2 Member Function Documentation . . . 150
10.64.2.1 FindLayer() . . . 150
10.64.2.2 GetAttr() . . . 151
10.64.2.3 GetBounds() . . . 151
10.64.2.4 GetCATD() . . . 152
10.64.2.5 GetLayerCATDEntry() . . . 152
10.64.2.6 GetLayerIndexedReader() . . . 152
10.64.2.7 GetLayerRasterReader() . . . 153
10.64.2.8 GetLayerType() . . . 153
10.64.2.9 GetXREF() . . . 154
10.64.2.10 Open() . . . 154
10.65 SFRegion Class Reference . . . 154
10.66 StackContext Struct Reference . . . 155
10.67 tm_unz_s Struct Reference . . . 155
10.68 tm_zip_s Struct Reference . . . 155
10.69 TupleEnvVarOptionName Struct Reference . . . 155

10.70 unz_file_info_internal_s Struct Reference . . . 155
10.71 unz_file_info_s Struct Reference . . . 155
10.72 unz_file_pos_s Struct Reference . . . 156
10.73 unz_global_info_s Struct Reference . . . 156
10.74 unz_s Struct Reference . . . 156
10.75 VSIArchiveContent Class Reference . . . 156
10.76 VSIArchiveEntry Struct Reference . . . 156
10.77 VSIArchiveEntryFileOffset Class Reference . . . 156
10.78 VSIArchiveFilesystemHandler Class Reference . . . 157
10.79 VSIArchiveReader Class Reference . . . 157
10.80 VSIDIR Struct Reference . . . 157
10.81 VSIDummyCryptFilesystemHandler Class Reference . . . 157
10.82 VSIErrorContext Struct Reference . . . 158
10.83 VSIFileManager Class Reference . . . 158
10.84 VSIFilesystemHandler Class Reference . . . 158
10.85 VSIReadDirRecursiveTask Struct Reference . . . 158
10.86 VSISparseFileFilesystemHandler Class Reference . . . 158
10.87 VSISparseFileHandle Class Reference . . . 159
10.87.1 Member Function Documentation . . . 159
10.87.1.1 Close() . . . 159
10.87.1.2 Eof() . . . 160
10.87.1.3 Read() . . . 160
10.87.1.4 Seek() . . . 161
10.87.1.5 Tell() . . . 161
10.87.1.6 Write() . . . 162
10.88 VSISubFileFilesystemHandler Class Reference . . . 162
10.89 VSISubFileHandle Class Reference . . . 163
10.89.1 Member Function Documentation . . . 163
10.89.1.1 Close() . . . 163
10.89.1.2 Eof() . . . 164

10.89.1.3 Read() 164

10.89.1.4 Seek() 165

10.89.1.5 Tell() 165

10.89.1.6 Write() 166

10.90 VSIVirtualHandle Class Reference 166

10.90.1 Detailed Description 167

10.90.2 Member Function Documentation 167

10.90.2.1 Close() 167

10.90.2.2 Eof() 168

10.90.2.3 Flush() 168

10.90.2.4 GetNativeFileDescriptor() 168

10.90.2.5 GetRangeStatus() 169

10.90.2.6 Read() 169

10.90.2.7 ReadMultiRange() 170

10.90.2.8 Seek() 170

10.90.2.9 Tell() 171

10.90.2.10 Truncate() 171

10.90.2.11 Write() 172

10.91 zip_fileinfo Struct Reference 172

10.92 zip_internal Struct Reference 173

10.93 zlib_filefunc_def_s Struct Reference 173

11 File Documentation 175

11.1 cpl_conv.h File Reference . . . 175

11.1.1 Detailed Description . . . 177

11.1.2 Macro Definition Documentation . . . 177

11.1.2.1 CPLFree . . . 177

11.1.3 Typedef Documentation . . . 177

11.1.3.1 CPLFileFinder . . . 178

11.1.4 Function Documentation . . . 178

11.1.4.1 CPLAtof() . . . 178

11.1.4.2 CPLAtofDelim() . . . 178

11.1.4.3 CPLAtofM() . . . 179

11.1.4.4 CPLAtoGIntBig() . . . 179

11.1.4.5 CPLAtoGIntBigEx() . . . 180

11.1.4.6 CPLCalloc() . . . 180

11.1.4.7 CPLCheckForFile() . . . 181

11.1.4.8 CPLCleanTrailingSlash() . . . 181

11.1.4.9 CPLCloseFileInZip() . . . 182

11.1.4.10 CPLCloseShared() . . . 182

11.1.4.11 CPLCloseZip() . . . 182

11.1.4.12 CPLCopyFile() . . . 183

11.1.4.13 CPLCopyTree() . . . 183

11.1.4.14 CPLCorrespondingPaths() . . . 183

11.1.4.15 CPLCreateFileInZip() . . . 183

11.1.4.16 CPLCreateZip() . . . 184

11.1.4.17 CPLDecToDMS() . . . 184

11.1.4.18 CPLDecToPackedDMS() . . . 184

11.1.4.19 CPLDefaultFindFile() . . . 184

11.1.4.20 CPLDMSToDec() . . . 185

11.1.4.21 CPLDumpSharedList() . . . 185

11.1.4.22 CPLExpandTilde() . . . 185

11.1.4.23 CPLExtractRelativePath() 186
11.1.4.24 CPLFGets() 186
11.1.4.25 CPLFinderClean() 187
11.1.4.26 CPLFindFile() 187
11.1.4.27 CPLFormCIFilename() 187
11.1.4.28 CPLFormFilename() 187
11.1.4.29 CPLGenerateTempFilename() 188
11.1.4.30 CPLGetBasename() 188
11.1.4.31 CPLGetConfigOption() 189
11.1.4.32 CPLGetConfigOptions() 190
11.1.4.33 CPLGetCurrentDir() 190
11.1.4.34 CPLGetDirname() 190
11.1.4.35 CPLGetExtension() 191
11.1.4.36 CPLGetFilename() 191
11.1.4.37 CPLGetHomeDir() 192
11.1.4.38 CPLGetPath() 192
11.1.4.39 CPLGetSharedList() 193
11.1.4.40 CPLGetSymbol() 193
11.1.4.41 CPLGetThreadLocalConfigOption() 194
11.1.4.42 CPLGetThreadLocalConfigOptions() 194
11.1.4.43 CPLIsFilenameRelative() 194
11.1.4.44 CPLIsPowerOfTwo() 195
11.1.4.45 CPLMalloc() 195
11.1.4.46 CPLMoveFile() 196
11.1.4.47 CPLOpenShared() 196
11.1.4.48 CPLPackedDMSToDec() 196
11.1.4.49 CPLPopFileFinder() 197
11.1.4.50 CPLPopFinderLocation() 197
11.1.4.51 CPLPrintInt32() 197
11.1.4.52 CPLPrintPointer() 198

11.1.4.53 CPLPrintString() 198
11.1.4.54 CPLPrintStringFill() 199
11.1.4.55 CPLPrintTime() 199
11.1.4.56 CPLPrintUIntBig() 200
11.1.4.57 CPLProjectRelativeFilename() 200
11.1.4.58 CPLPushFileFinder() 201
11.1.4.59 CPLPushFinderLocation() 201
11.1.4.60 CPLReadLine() 201
11.1.4.61 CPLReadLineL() 202
11.1.4.62 CPLRealloc() 202
11.1.4.63 CPLResetExtension() 203
11.1.4.64 CPLScanDouble() 203
11.1.4.65 CPLScanLong() 204
11.1.4.66 CPLScanPointer() 204
11.1.4.67 CPLScanString() 204
11.1.4.68 CPLScanUIntBig() 205
11.1.4.69 CPLScanULong() 205
11.1.4.70 CPLSetConfigOption() 206
11.1.4.71 CPLSetConfigOptions() 206
11.1.4.72 CPLsetlocale() 207
11.1.4.73 CPLSetThreadLocalConfigOption() 207
11.1.4.74 CPLSetThreadLocalConfigOptions() 208
11.1.4.75 CPLStat() 208
11.1.4.76 CPLStrdup() 208
11.1.4.77 CPLStringToComplex() 209
11.1.4.78 CPLStrlwr() 209
11.1.4.79 CPLStrtod() 209
11.1.4.80 CPLStrtodDelim() 210
11.1.4.81 CPLStrtof() 210
11.1.4.82 CPLStrtofDelim() 211

11.1.4.83 CPLSymlink() . . . 211
11.1.4.84 CPLUnlinkTree() . . . 212
11.1.4.85 CPLWriteFileInZip() . . . 212
11.1.4.86 CPLZLibInflate() . . . 212
11.2 cpl_error.h File Reference . . . 213
11.2.1 Detailed Description . . . 214
11.2.2 Macro Definition Documentation . . . 214
11.2.2.1 CPLAssert . . . 214
11.2.2.2 CPLE_AppDefined . . . 214
11.2.2.3 CPLE_AssertionFailed . . . 214
11.2.2.4 CPLE_AWSAccessDenied . . . 214
11.2.2.5 CPLE_AWSBucketNotFound . . . 215
11.2.2.6 CPLE_AWSInvalidCredentials . . . 215
11.2.2.7 CPLE_AWSObjectNotFound . . . 215
11.2.2.8 CPLE_AWSSignatureDoesNotMatch . . . 215
11.2.2.9 CPLE_FileIO . . . 215
11.2.2.10 CPLE_HttpResponse . . . 215
11.2.2.11 CPLE_IllegalArg . . . 215
11.2.2.12 CPLE_None . . . 215
11.2.2.13 CPLE_NotSupported . . . 216
11.2.2.14 CPLE_NoWriteAccess . . . 216
11.2.2.15 CPLE_ObjectNull . . . 216
11.2.2.16 CPLE_OpenFailed . . . 216
11.2.2.17 CPLE_OutOfMemory . . . 216
11.2.2.18 CPLE_UserInterrupt . . . 216
11.2.2.19 VALIDATE_POINTER0 . . . 216
11.2.2.20 VALIDATE_POINTER1 . . . 217
11.2.3 Typedef Documentation . . . 217
11.2.3.1 CPLErrorHandler . . . 217
11.2.3.2 CPLErrorNum . . . 217

11.2.4 Enumeration Type Documentation . . . 217
11.2.4.1 CPLErr . . . 217
11.2.5 Function Documentation . . . 217
11.2.5.1 _CPLAssert() . . . 218
11.2.5.2 CPLDefaultErrorHandler() . . . 218
11.2.5.3 CPLEmergencyError() . . . 218
11.2.5.4 CPLErrorReset() . . . 218
11.2.5.5 CPLErrorSetState() . . . 219
11.2.5.6 CPLErrorV() . . . 219
11.2.5.7 CPLGetErrorCounter() . . . 219
11.2.5.8 CPLGetErrorHandlerUserData() . . . 220
11.2.5.9 CPLGetLastErrorMsg() . . . 220
11.2.5.10 CPLGetLastErrorNo() . . . 220
11.2.5.11 CPLGetLastErrorType() . . . 221
11.2.5.12 CPLLoggingErrorHandler() . . . 221
11.2.5.13 CPLPopErrorHandler() . . . 221
11.2.5.14 CPLPushErrorHandler() . . . 221
11.2.5.15 CPLPushErrorHandlerEx() . . . 222
11.2.5.16 CPLQuietErrorHandler() . . . 222
11.2.5.17 CPLSetCurrentErrorHandlerCatchDebug() . . . 222
11.2.5.18 CPLSetErrorHandler() . . . 223
11.2.5.19 CPLSetErrorHandlerEx() . . . 223
11.2.5.20 CPLTurnFailureIntoWarning() . . . 224
11.3 cpl_hash_set.h File Reference . . . 224
11.3.1 Detailed Description . . . 225
11.3.2 Typedef Documentation . . . 225
11.3.2.1 CPLHashSet . . . 225
11.3.2.2 CPLHashSetEqualFunc . . . 225
11.3.2.3 CPLHashSetFreeEltFunc . . . 225
11.3.2.4 CPLHashSetHashFunc . . . 225

11.3.2.5 CPLHashSetIterEltFunc . . . 225
11.3.3 Function Documentation . . . 225
11.3.3.1 CPLHashSetClear() . . . 225
11.3.3.2 CPLHashSetDestroy() . . . 226
11.3.3.3 CPLHashSetEqualPointer() . . . 226
11.3.3.4 CPLHashSetEqualStr() . . . 226
11.3.3.5 CPLHashSetForeach() . . . 228
11.3.3.6 CPLHashSetHashPointer() . . . 228
11.3.3.7 CPLHashSetHashStr() . . . 229
11.3.3.8 CPLHashSetInsert() . . . 229
11.3.3.9 CPLHashSetLookup() . . . 229
11.3.3.10 CPLHashSetNew() . . . 230
11.3.3.11 CPLHashSetRemove() . . . 230
11.3.3.12 CPLHashSetRemoveDeferRehash() . . . 231
11.3.3.13 CPLHashSetSize() . . . 231
11.4 cpl_http.h File Reference . . . 232
11.4.1 Detailed Description . . . 232
11.4.2 Function Documentation . . . 232
11.4.2.1 CPLHTTPDestroyMultiResult() . . . 233
11.4.2.2 CPLHTTPDestroyResult() . . . 233
11.4.2.3 CPLHTTPEnabled() . . . 233
11.4.2.4 CPLHTTPFetch() . . . 233
11.4.2.5 CPLHTTPFetchEx() . . . 236
11.4.2.6 CPLHTTPMultiFetch() . . . 236
11.4.2.7 CPLHTTPParseMultipartMime() . . . 237
11.4.2.8 GOA2GetAccessTokenFromCloudEngineVM() . . . 237
11.4.2.9 GOA2GetAccessTokenFromServiceAccount() . . . 238
11.4.2.10 GOA2GetAuthorizationURL() . . . 239
11.4.2.11 GOA2GetRefreshToken() . . . 239
11.5 cpl_json.h File Reference . . . 240

11.5.1 Detailed Description . . . 240
11.6 cpl_list.h File Reference . . . 240
11.6.1 Detailed Description . . . 241
11.6.2 Typedef Documentation . . . 241
11.6.2.1 CPLList . . . 241
11.6.3 Function Documentation . . . 241
11.6.3.1 CPLListAppend() . . . 241
11.6.3.2 CPLListCount() . . . 242
11.6.3.3 CPLListDestroy() . . . 242
11.6.3.4 CPLListGet() . . . 242
11.6.3.5 CPLListGetData() . . . 243
11.6.3.6 CPLListGetLast() . . . 243
11.6.3.7 CPLListGetNext() . . . 243
11.6.3.8 CPLListInsert() . . . 244
11.6.3.9 CPLListRemove() . . . 244
11.7 cpl_minixml.h File Reference . . . 245
11.7.1 Detailed Description . . . 246
11.7.2 Typedef Documentation . . . 246
11.7.2.1 CPLXMLNode . . . 246
11.7.3 Enumeration Type Documentation . . . 246
11.7.3.1 CPLXMLNodeType . . . 246
11.7.4 Function Documentation . . . 247
11.7.4.1 CPLAddXMLAttributeAndValue() . . . 247
11.7.4.2 CPLAddXMLChild() . . . 247
11.7.4.3 CPLAddXMLSibling() . . . 248
11.7.4.4 CPLCleanXMLElementName() . . . 248
11.7.4.5 CPLCloneXMLTree() . . . 249
11.7.4.6 CPLCreateXMLElementAndValue() . . . 249
11.7.4.7 CPLCreateXMLNode() . . . 250
11.7.4.8 CPLDestroyXMLNode() . . . 250

11.7.4.9 CPLGetXMLNode() . . . 250
11.7.4.10 CPLGetXMLValue() . . . 251
11.7.4.11 CPLParseXMLFile() . . . 252
11.7.4.12 CPLParseXMLString() . . . 252
11.7.4.13 CPLRemoveXMLChild() . . . 253
11.7.4.14 CPLSearchXMLNode() . . . 253
11.7.4.15 CPLSerializeXMLTree() . . . 254
11.7.4.16 CPLSerializeXMLTreeToFile() . . . 254
11.7.4.17 CPLSetXMLValue() . . . 255
11.7.4.18 CPLStripXMLNamespace() . . . 255
11.8 cpl_odbc.h File Reference . . . 256
11.8.1 Detailed Description . . . 256
11.9 cpl_port.h File Reference . . . 256
11.9.1 Detailed Description . . . 258
11.9.2 Macro Definition Documentation . . . 258
11.9.2.1 ABS . . . 258
11.9.2.2 CPL_FALLTHROUGH . . . 258
11.9.2.3 CPL_FORMAT_STRING . . . 258
11.9.2.4 CPL_FRMT_GB_WITHOUT_PREFIX . . . 258
11.9.2.5 CPL_FRMT_GIB . . . 258
11.9.2.6 CPL_FRMT_GUIB . . . 259
11.9.2.7 CPL_LSBINT16PTR . . . 259
11.9.2.8 CPL_LSBINT32PTR . . . 259
11.9.2.9 CPL_LSBPTR16 . . . 259
11.9.2.10 CPL_LSBPTR32 . . . 259
11.9.2.11 CPL_LSBPTR64 . . . 260
11.9.2.12 CPL_LSBSINT16PTR . . . 260
11.9.2.13 CPL_LSBSINT32PTR . . . 260
11.9.2.14 CPL_LSBUINT16PTR . . . 260
11.9.2.15 CPL_LSBUINT32PTR . . . 260

11.9.2.16 CPL_LSBWORD16 260
11.9.2.17 CPL_LSBWORD32 260
11.9.2.18 CPL_MSBPTR16 261
11.9.2.19 CPL_MSBPTR32 261
11.9.2.20 CPL_MSBPTR64 261
11.9.2.21 CPL_MSBWORD16 261
11.9.2.22 CPL_MSBWORD32 261
11.9.2.23 CPL_NO_RETURN 261
11.9.2.24 CPL_NULL_TERMINATED 261
11.9.2.25 CPL_PRINT_FUNC_FORMAT 262
11.9.2.26 CPL_RESTRICT 262
11.9.2.27 CPL_RETURNS_NONNULL 262
11.9.2.28 CPL_SCAN_FUNC_FORMAT 262
11.9.2.29 CPL_SCANF_FORMAT_STRING 262
11.9.2.30 CPL_SWAP16 262
11.9.2.31 CPL_SWAP16PTR 263
11.9.2.32 CPL_SWAP32 263
11.9.2.33 CPL_SWAP32PTR 263
11.9.2.34 CPL_SWAP64 264
11.9.2.35 CPL_SWAP64PTR 264
11.9.2.36 CPL_SWAPDOUBLE 264
11.9.2.37 CPL_UNUSED 264
11.9.2.38 CPL_WARN_UNUSED_RESULT 265
11.9.2.39 CPLIsNan 265
11.9.2.40 EQUAL 265
11.9.2.41 EQUALN 265
11.9.2.42 GINT64_MAX 265
11.9.2.43 GINT64_MIN 265
11.9.2.44 GINTBIG_MAX 265
11.9.2.45 GINTBIG_MIN 266

11.9.2.46 GUINT64_MAX . . . 266
11.9.2.47 GUINTBIG_MAX . . . 266
11.9.2.48 M_PI . . . 266
11.9.2.49 MAX . . . 266
11.9.2.50 MIN . . . 266
11.9.2.51 STARTS_WITH . . . 266
11.9.2.52 STARTS_WITH_CI . . . 267
11.9.2.53 STRCASECMP . . . 267
11.9.2.54 STRNCASECMP . . . 267
11.9.3 Typedef Documentation . . . 267
11.9.3.1 CSLConstList . . . 267
11.9.3.2 GBool . . . 267
11.9.3.3 GByte . . . 267
11.9.3.4 GInt16 . . . 268
11.9.3.5 GInt32 . . . 268
11.9.3.6 GInt64 . . . 268
11.9.3.7 GIntBig . . . 268
11.9.3.8 GPtrDiff_t . . . 268
11.9.3.9 GUInt16 . . . 268
11.9.3.10 GUInt32 . . . 268
11.9.3.11 GUInt64 . . . 268
11.9.3.12 GUIntBig . . . 269
11.10 cpl_quad_tree.h File Reference . . . 269
11.10.1 Detailed Description . . . 269
11.10.2 Typedef Documentation . . . 270
11.10.2.1 CPLQuadTree . . . 270
11.10.2.2 CPLQuadTreeDumpFeatureFunc . . . 270
11.10.2.3 CPLQuadTreeForeachFunc . . . 270
11.10.2.4 CPLQuadTreeGetBoundsFunc . . . 270
11.10.3 Function Documentation . . . 270

11.10.3.1 CPLQuadTreeCreate() 270
11.10.3.2 CPLQuadTreeDestroy() 271
11.10.3.3 CPLQuadTreeDump() 271
11.10.3.4 CPLQuadTreeForeach() 271
11.10.3.5 CPLQuadTreeGetAdvisedMaxDepth() 272
11.10.3.6 CPLQuadTreeGetStats() 272
11.10.3.7 CPLQuadTreeInsert() 272
11.10.3.8 CPLQuadTreeInsertWithBounds() 272
11.10.3.9 CPLQuadTreeSearch() 274
11.10.3.10 CPLQuadTreeSetBucketCapacity() 274
11.10.3.11 CPLQuadTreeSetMaxDepth() 274
11.11 cpl_string.h File Reference 275
11.11.1 Detailed Description 277
11.11.2 Macro Definition Documentation 277
11.11.2.1 CPL_ENC_ASCII 277
11.11.2.2 CPL_ENC_ISO8859_1 277
11.11.2.3 CPL_ENC_LOCALE 277
11.11.2.4 CPL_ENC_UCS2 278
11.11.2.5 CPL_ENC_UCS4 278
11.11.2.6 CPL_ENC_UTF16 278
11.11.2.7 CPL_ENC_UTF8 278
11.11.2.8 CPLES_BackslashQuotable 278
11.11.2.9 CPLES_CSV 278
11.11.2.10 CPLES_CSV_FORCE_QUOTING 278
11.11.2.11 CPLES_SQL 278
11.11.2.12 CPLES_URL 279
11.11.2.13 CPLES_XML 279
11.11.2.14 CPLES_XML_BUT_QUOTES 279
11.11.2.15 CSLT_ALLOWEMPTYTOKENS 279
11.11.2.16 CSLT_HONOURSTRINGS 279

11.11.2.17 CSLT_PRESERVEESCAPES . . . 279
11.11.2.18 CSLT_PRESERVEQUOTES . . . 279
11.11.2.19 CSLT_STRIPENDSPACES . . . 279
11.11.2.20 CSLT_STRIPLEADSPACES . . . 280
11.11.3 Enumeration Type Documentation . . . 280
11.11.3.1 CPLValueType . . . 280
11.11.4 Function Documentation . . . 280
11.11.4.1 CPLBase64DecodeInPlace() . . . 280
11.11.4.2 CPLBase64Encode() . . . 280
11.11.4.3 CPLBinaryToHex() . . . 280
11.11.4.4 CPLEncodingCharSize() . . . 281
11.11.4.5 CPLEscapeString() . . . 281
11.11.4.6 CPLForceToASCII() . . . 282
11.11.4.7 CPLGetValueType() . . . 283
11.11.4.8 CPLHexToBinary() . . . 283
11.11.4.9 CPLIsUTF8() . . . 283
11.11.4.10 CPLParseNameValue() . . . 284
11.11.4.11 CPLRecode() . . . 284
11.11.4.12 CPLRecodeFromWChar() . . . 285
11.11.4.13 CPLRecodeToWChar() . . . 286
11.11.4.14 CPLStrlcat() . . . 286
11.11.4.15 CPLStrlcpy() . . . 287
11.11.4.16 CPLStrlenUTF8() . . . 288
11.11.4.17 CPLStrnlen() . . . 288
11.11.4.18 CPLTestBoolean() . . . 289
11.11.4.19 CPLUnescapeString() . . . 289
11.11.4.20 CSLAddNameValue() . . . 290
11.11.4.21 CSLAddString() . . . 290
11.11.4.22 CSLAddStringMayFail() . . . 290
11.11.4.23 CSLCount() . . . 290

11.11.4.24 CSLDestroy() . . . 291
11.11.4.25 CSLDuplicate() . . . 291
11.11.4.26 CSLFetchBoolean() . . . 291
11.11.4.27 CSLFetchNameValue() . . . 292
11.11.4.28 CSLFetchNameValueDef() . . . 292
11.11.4.29 CSLFetchNameValueMultiple() . . . 292
11.11.4.30 CSLFindName() . . . 293
11.11.4.31 CSLFindString() . . . 293
11.11.4.32 CSLFindStringCaseSensitive() . . . 293
11.11.4.33 CSLGetField() . . . 294
11.11.4.34 CSLInsertString() . . . 294
11.11.4.35 CSLInsertStrings() . . . 294
11.11.4.36 CSLLoad() . . . 295
11.11.4.37 CSLLoad2() . . . 296
11.11.4.38 CSLMerge() . . . 296
11.11.4.39 CSLParseCommandLine() . . . 297
11.11.4.40 CSLPartialFindString() . . . 297
11.11.4.41 CSLPrint() . . . 298
11.11.4.42 CSLRemoveStrings() . . . 298
11.11.4.43 CSLSave() . . . 298
11.11.4.44 CSLSetNameValue() . . . 298
11.11.4.45 CSLSetNameValueSeparator() . . . 299
11.11.4.46 CSLTestBoolean() . . . 299
11.11.4.47 CSLTokenizeString() . . . 300
11.11.4.48 CSLTokenizeString2() . . . 300
11.11.4.49 CSLTokenizeStringComplex() . . . 301
11.12 cpl_virtualmem.h File Reference . . . 301
11.12.1 Detailed Description . . . 302
11.12.2 Typedef Documentation . . . 302
11.12.2.1 CPLVirtualMem . . . 302

11.12.2.2 CPLVirtualMemCachePageCbk . . . 302
11.12.2.3 CPLVirtualMemFreeUserData . . . 303
11.12.2.4 CPLVirtualMemUnCachePageCbk . . . 303
11.12.3 Enumeration Type Documentation . . . 303
11.12.3.1 CPLVirtualMemAccessMode . . . 304
11.12.4 Function Documentation . . . 304
11.12.4.1 CPLGetPageSize() . . . 304
11.12.4.2 CPLIsVirtualMemFileMapAvailable() . . . 304
11.12.4.3 CPLVirtualMemDeclareThread() . . . 305
11.12.4.4 CPLVirtualMemDerivedNew() . . . 305
11.12.4.5 CPLVirtualMemFileMapNew() . . . 306
11.12.4.6 CPLVirtualMemFree() . . . 306
11.12.4.7 CPLVirtualMemGetAccessMode() . . . 307
11.12.4.8 CPLVirtualMemGetAddr() . . . 307
11.12.4.9 CPLVirtualMemGetPageSize() . . . 308
11.12.4.10 CPLVirtualMemGetSize() . . . 308
11.12.4.11 CPLVirtualMemIsAccessThreadSafe() . . . 309
11.12.4.12 CPLVirtualMemIsFileMapping() . . . 309
11.12.4.13 CPLVirtualMemManagerTerminate() . . . 310
11.12.4.14 CPLVirtualMemNew() . . . 310
11.12.4.15 CPLVirtualMemPin() . . . 311
11.12.4.16 CPLVirtualMemUnDeclareThread() . . . 312
11.13 cpl_vsi.h File Reference . . . 312
11.13.1 Detailed Description . . . 315
11.13.2 Macro Definition Documentation . . . 316
11.13.2.1 CPLReadDir . . . 316
11.13.2.2 VSI_CALLOC_VERBOSE . . . 316
11.13.2.3 VSI_ISDIR . . . 316
11.13.2.4 VSI_ISLNK . . . 316
11.13.2.5 VSI_ISREG . . . 316

11.13.2.6 VSI_L_OFFSET_MAX . . . 316
11.13.2.7 VSI_MALLOC2_VERBOSE . . . 317
11.13.2.8 VSI_MALLOC3_VERBOSE . . . 317
11.13.2.9 VSI_MALLOC_ALIGNED_AUTO_VERBOSE . . . 317
11.13.2.10 VSI_MALLOC_VERBOSE . . . 317
11.13.2.11 VSI_REALLOC_VERBOSE . . . 317
11.13.2.12 VSI_STAT_EXISTS_FLAG . . . 317
11.13.2.13 VSI_STAT_NATURE_FLAG . . . 318
11.13.2.14 VSI_STAT_SET_ERROR_FLAG . . . 318
11.13.2.15 VSI_STAT_SIZE_FLAG . . . 318
11.13.2.16 VSI_STRDUP_VERBOSE . . . 318
11.13.3 Typedef Documentation . . . 318
11.13.3.1 vsi_l_offset . . . 318
11.13.3.2 VSILFILE . . . 318
11.13.3.3 VSIStatBufL . . . 318
11.13.3.4 VSIWriteFunction . . . 319
11.13.4 Enumeration Type Documentation . . . 319
11.13.4.1 VSIRangeStatus . . . 319
11.13.5 Function Documentation . . . 319
11.13.5.1 CPLGetPhysicalRAM() . . . 319
11.13.5.2 CPLGetUsablePhysicalRAM() . . . 320
11.13.5.3 VSICalloc() . . . 320
11.13.5.4 VSICallocVerbose() . . . 320
11.13.5.5 VSIFCloseL() . . . 320
11.13.5.6 VSIFEofL() . . . 321
11.13.5.7 VSIFFlushL() . . . 321
11.13.5.8 VSIFGetNativeFileDescriptorL() . . . 322
11.13.5.9 VSIFGetRangeStatusL() . . . 322
11.13.5.10 VSIFileFromMemBuffer() . . . 323
11.13.5.11 VSIFOpenExL() . . . 323

11.13.5.12 VSIFOpenL() 324
11.13.5.13 VSIFPutcL() 325
11.13.5.14 VSIFReadL() 325
11.13.5.15 VSIFReadMultiRangeL() 326
11.13.5.16 VSIFree() 327
11.13.5.17 VSIFreeAligned() 327
11.13.5.18 VSIFSeekL() 327
11.13.5.19 VSIFTellL() 328
11.13.5.20 VSIFTruncateL() 328
11.13.5.21 VSIFWriteL() 329
11.13.5.22 VSIGetActualURL() 329
11.13.5.23 VSIGetDiskFreeSpace() 330
11.13.5.24 VSIGetFileSystemOptions() 330
11.13.5.25 VSIGetFileSystemsPrefixes() 331
11.13.5.26 VSIGetMemFileBuffer() 331
11.13.5.27 VSIGetSignedURL() 332
11.13.5.28 VSIHasOptimizedReadMultiRange() 333
11.13.5.29 VSIIngestFile() 333
11.13.5.30 VSIInstallGZipFileHandler() 334
11.13.5.31 VSIInstallMemFileHandler() 334
11.13.5.32 VSIInstallSparseFileHandler() 335
11.13.5.33 VSIInstallStdinHandler() 335
11.13.5.34 VSIInstallStdoutHandler() 335
11.13.5.35 VSIInstallSubFileHandler() 336
11.13.5.36 VSIInstallTarFileHandler() 336
11.13.5.37 VSIInstallZipFileHandler() 337
11.13.5.38 VSIIsCaseSensitiveFS() 337
11.13.5.39 VSIMalloc() 338
11.13.5.40 VSIMalloc2() 338
11.13.5.41 VSIMalloc2Verbose() 338

11.13.5.42 VSIMalloc3() . . . 338
11.13.5.43 VSIMalloc3Verbose() . . . 339
11.13.5.44 VSIMallocAligned() . . . 339
11.13.5.45 VSIMallocAlignedAuto() . . . 339
11.13.5.46 VSIMallocAlignedAutoVerbose() . . . 340
11.13.5.47 VSIMallocVerbose() . . . 340
11.13.5.48 VSIMkdir() . . . 340
11.13.5.49 VSIMkdirRecursive() . . . 341
11.13.5.50 VSIReadDir() . . . 341
11.13.5.51 VSIReadDirEx() . . . 342
11.13.5.52 VSIReadDirRecursive() . . . 342
11.13.5.53 VSIRealloc() . . . 343
11.13.5.54 VSIReallocVerbose() . . . 343
11.13.5.55 VSIRename() . . . 343
11.13.5.56 VSIRewindL() . . . 344
11.13.5.57 VSIRmdir() . . . 344
11.13.5.58 VSIRmdirRecursive() . . . 345
11.13.5.59 VSIStatExL() . . . 345
11.13.5.60 VSIStatL() . . . 346
11.13.5.61 VSIStdoutSetRedirection() . . . 346
11.13.5.62 VSIStrdup() . . . 347
11.13.5.63 VSIStrdupVerbose() . . . 347
11.13.5.64 VSIStrerror() . . . 347
11.13.5.65 VSISupportsSparseFiles() . . . 347
11.13.5.66 VSIUnlink() . . . 348
11.14 cpl_worker_thread_pool.h File Reference . . . 348
11.14.1 Detailed Description . . . 348
11.15 sdtsdataset.cpp File Reference . . . 349
11.15.1 Detailed Description . . . 349

Index 351

Chapter 1

ISO8211Lib

Introduction

ISO8211Lib is intended to be a simple reader for ISO/IEC 8211 formatted files, particularly those that are part of SDTS and S-57 datasets. It consists of open source, easy to compile and integrate C++ code.

ISO 8211 Background

The `ISO 8211 FAQ` has some good background on ISO 8211 formatted files. I will briefly introduce it here, with reference to the library classes representing the components.

An 8211 file (DDFModule) consists of a series of logical records. The first record is special, and is called the DDR (Data Description Record). It basically contains definitions of all the data objects (fields or DDFFieldDefn objects) that can occur on the following data records.

The remainder of the records are known as DRs (data records - DDFRecord). They each contain one or more field (DDFField) instances. What fields appear on what records is not defined by ISO 8211, though more specific requirements may be implied by a particular data standard such as SDTS or S-57.

Each field instance has a name, and consists of a series of subfields. A given field always has the same subfields in each field instance, and these subfields are defined in the DDR (DDFSubfieldDefn), in association with their field definition (DDFFieldDefn). A field may appear 0, 1, or many times in a DR.

Each subfield has a name, format (from the DDFSubfieldDefn) and actual subfield data for a particular DR. Some fields contain an *array* of their group of subfields. For instance a *coordinate field* may have X and Y subfields, and they may repeat many times within one coordinate field indicating a series of points.

This would be a real good place for a UML diagram of ISO 8211, and the corresponding library classes!

Development Information

The **iso8211.h** contains the definitions for all public ISO8211Lib classes, enumerations and other services.

To establish access to an ISO 8211 dataset, instantiate a DDFModule object, and then use the DDFModule::Open() method. This will read the DDR, and establish all the DDFFieldDefn, and DDFSubfieldDefn objects which can be queried off the DDFModule.

The use DDFModule::ReadRecord() to fetch data records (DDFRecord). When a record is read, a list of field objects (DDFField) on that record are created. They can be queried with various DDFRecord methods.

Data pointers for individual subfields of a DDFField can be fetched with DDFField::GetSubfieldData(). The interpreted value can then be extracted with the appropriate one of DDFSubfieldDefn::ExtractIntValue(), DDFSubfield↩Defn::ExtractStringValue(), or DDFSubfieldDefn::ExtractFloatValue(). Note that there is no object instantiated for individual subfields of a DDFField. Instead the application extracts a pointer to the subfields raw data, and then uses the DDFSubfieldDefn for that subfield to extract a usable value from the raw data.

Once the end of the file has been encountered (DDFModule::ReadRecord() returns NULL), the DDFModule should be deleted, which will close the file, and cleanup all records, definitions and related objects.

Class APIs

- DDFModule class.
- DDFFieldDefn class.
- DDFSubfieldDefn class.
- DDFRecord class.
- DDFField class.

A complete `Example Reader` should clarify simple use of ISO8211Lib.

Related Information

- The ISO 8211 standard can be ordered through `ISO`. It cost me about $200CDN.
- The `ISO/IEC 8211/DDFS Home Page` contains tutorials and some code by Dr. Alfred A. Brooks, one of the originators of the 8211 standard.
- The `ISO/IEC 8211 Home Page` has some python code for parsing 8211 files, and some other useful background.
- The `SDTS++` library from the USGS includes support for ISO 8211. It doesn't include some of the 1994 additions to ISO 8211, but it is sufficient for SDTS, and quite elegantly done. Also supports writing ISO 8211 files.
- The USGS also has an older `FIPS123` library which supports the older profile of ISO 8211 (to some extent).

Licensing

This library is offered as `Open Source`. In particular, it is offered under the X Consortium license which doesn't attempt to impose any copyleft, or credit requirements on users of the code.

The precise license text is:

Copyright (c) 1999, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN↩ CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Building the Source

1. First, fetch the source. The most recent source should be accessible at an url such as http://home.↩gdal.org/projects/iso8211/iso8211lib-1.4.zip.
2. Untar the source.

   ```
   % unzip iso8211lib-1.4.zip
   ```

3. On unix you can now type ``configure'' to establish configuration options.
4. On unix you can now type make to build libiso8211.a, and the sample mainline 8211view.

Windows developers will have to create their own makefile/project but can base it on the very simple Makefile.in provided. As well, you would need to copy cpl_config.h.in to cpl_config.h, and modify as needed. The default will likely work OK, but may result in some compiler warnings. Let me know if you are having difficulties, and I will prepare a VC++ makefile.

Author and Acknowledgements

The primary author of ISO8211Lib is Frank Warmerdam, and I can be reached at warmerdam@pobox.com.
I am eager to receive bug reports, and also open to praise or suggestions.

I would like to thank:

- Safe Software who funded development of this library, and agreed for it to be Open Source.
- Mark Colletti, a primary author of SDTS++ from which I derived most of what I know about ISO 8211 and who was very supportive, answering a variety of questions.
- Tony J Ibbs, author of the ISO/IEC 8211 home page who answered a number of questions, and collected a variety of very useful information.
- Rodney Jenson, for a detailed bug report related to repeating variable length fields (from S-57).

I would also like to dedicate this library to the memory of Sol Katz. Sol released a variety of SDTS (and hence ISO8211) translators, at substantial personal effort, to the GIS community along with the many other generous contributions he made to the community. His example has been an inspiration to me, and I hope similar efforts on my part will contribute to his memory.

Chapter 2

sdts_al_main

<title>SDTS Abstraction Library</title>

Introduction

SDTS_AL, the SDTS Abstraction Library, is intended to be an relatively easy to use library for reading vector from SDTS TVP (Topological Vector Profile) files, primary DLG data from the USGS. It also include support for reading raster data such as USGS DEMs in SDTS format. It consists of open source, easy to compile and integrate C++ code.

SDTS Background

The USGS SDTS Page at `http://mcmcweb.er.usgs.gov/sdts` is the definitive source of information on the SDTS format. The SDTS format is based on the `ISO 8211` encoding scheme for the underlying files, and the SDTS Abstraction Library uses `ISO8211Lib` library to decode them. All references to DDF∗ classes are from ISO8211Lib.

An SDTS Transfer is a grouping of ISO8211 encoded files (ending in the .DDF extension), normally with part of the basename in common. For instance a USGS DLG SDTS transfer might consists of many files matching the SC01????.DDF pattern. The key file in an SDTS transfer is the catalog file, such as SC01CATD.DDF.

Development Information

The `sdts_al.h` include file contains the definitions for all public SDTS classes, enumerations and other services.

The SDTSTransfer class is used to access a transfer as a whole. The SDTSTransfer::Open() method is passed the name of the catalog file, such as SC01CATD.DDF, to open.

The SDTSTransfer analyses the catalog, and some other aspects of the transfer, and builds a list of feature layers. This list can be accessed using the SDTSTransfer::GetLayerCount(), SDTSTransfer::GetLayerType(), and SDTSTransfer::GetLayerIndexedReader() methods. A typical TVP (Topological Vector Profile) transfer might include three point layers (of type SLTPoint), a line layer (of type SLTLine), a polygon layer (of type SLTPoly) as well as some additional attribute layers (of type SLTAttr). the SDTSTransfer::GetLayerIndexedReader() method can be used to instantiate a reader object for reading a particular layer. (NOTE: raster layers are handled differently).

Each type of SDTSIndexedReader (SDTSPointReader, SDTSLineReader, SDTSPolygonReader, and SDTSAttrReader) returns specific subclasses of SDTSIndexedFeature from the SDTSIndexedReader::GetNextFeature() method. These classes are SDTSRawPoint, SDTSRawLine, SDTSRawPolygon and SDTSAttrRecord. These classes can be investigated for details on the data available for each.

See the `SDTS_AL Tutorial` for more information on how to use this library.

Building the Source on Unix

1. First, fetch the source. The most recent source should be accessible at an url such as ftp://gdal.velocet.ca/pub/outgoing/sdts_1_3.tar.gz.
2. Unpack the source.

```
% gzip -d sdts_1_3.tar.gz
% tar xzvf sdts_1_3.tar.gz
```

3. Type ``configure'' to establish configuration options.
4. Type make to build sdts_al.a, and the sample mainline sdts2shp.

See the SDTS_AL Tutorial for more information on how to use this library.

Building the Source on Windows

1. First, fetch the source. The most recent source should be accessible at an url such as ftp://gdal.velocet.ca/pub/outgoing/sdts_1_3.zip.
2. Unpack the source.

```
C:> unzip sdts_1_3.zip
```

3. Build using makefile.vc with VC++. You will need the VC++ runtime environment variables (LIB/INCLUDE) set properly. This will build the library (sdts_al.lib), and the executables sdts2shp.exe, 8211view.exe and 8211dump.exe.

```
C:> nmake /f makefile.vc
```

See the SDTS_AL Tutorial for more information on how to use this library.

The sdts2shp Sample Program

The sdts2shp program distributed with this toolkit is primary intended to serve as an example of how to use the SDTS access library. However, it can be useful to translate SDTS datasets into ESRI Shapefile format.

```
Usage: sdts2shp CATD_filename [-o shapefile_name]
                [-m module_name] [-v]

Modules include 'LE01', 'PC01', 'NP01' and 'ARDF'
```

A typical session in which we inspect the contents of a transfer, and then extract polygon and line layers might look like this:

```
warmerda[134]% sdts2shp data/SC01CATD.DDF -v
Layers:
  ASCF: 'Attribute Primary         '
  AHDR: 'Attribute Primary         '
  NP01: 'Point-Node                '
  NA01: 'Point-Node                '
  NO01: 'Point-Node                '
  LE01: 'Line                      '
  PC01: 'Polygon                   '

warmerda[135]% sdts2shp data/SC01CATD.DDF -m PC01 -o pc01.shp
warmerda[136]% sdts2shp data/SC01CATD.DDF -m LE01 -o le01.shp
```

A prebuilt executable is available for Windows.

Licensing

This library is offered as Open Source. In particular, it is offered under the X Consortium license which doesn't attempt to impose any copyleft, or credit requirements on users of the code.

The precise license text is:

Copyright (c) 1999, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN↩ CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Author and Acknowledgements

The primary author of SDTS_AL is Frank Warmerdam, and I can be reached at warmerdam@pobox.com. I am eager to receive bug reports, and also open to praise or suggestions.

I would like to thank:

- Safe Software who funded development of this library, and agreed for it to be Open Source.
- Mark Colletti, a primary author of SDTS++ from which I derived most of what I know about SDTS and IS↩ O8211 and who was very supportive, answering a variety of questions.

I would also like to dedicate this library to the memory of Sol Katz. Sol released a variety of SDTS translators, at substantial personal effort, to the GIS community along with the many other generous contributions he made to the community. His example has been an inspiration to me, and I hope similar efforts on my part will contribute to his memory.

Chapter 3

SDTS_AL_TUT

<title>SDTS Abstraction Library Tutorial</title>

This page is a walk through of the polygon layer portion of the `sdts2shp.cpp` example application. It is should give sufficient information to utilize the SDTS_AL library to read SDTS files.

Opening the Transfer

The following statements will open an SDTS transfer. The filename passed to SDTSTransfer::Open() should be the name of the catalog file, such as `palo_alto/SC01CATD.DDF`. The Open() method returns FALSE if it fails for any reason. In addition to the message we print out ourselves, the SDTSTransfer::Open() method will also emit its own error message using CPLError(). See the cpl_error.h page for more information on how to capture and control CPLError() style error reporting.

```
#include "stds_al.h"

...

    SDTSTransfer oTransfer;

    if( !oTransfer.Open( pszCATDFilename ) )
    {
        fprintf( stderr,
                 "Failed to read CATD file `%s'\n",
                 pszCATDFilename );
        exit( 100 );
    }
```

Getting a Layer List

Once an SDTSTransfer has been opened, it is possible to establish what layers are available. The sdts2shp example problem includes a -v argument to dump a list of available layers. It isn't normally necessary to use the SDTS_CATD (catalog) from an application to access SDTS files; however, in this example we use it to fetch a module name, and description for each of the available layers.

In particular, the SDTSTransfer::GetLayerCount() method returns the number of feature layers in the transfer and the SDTSTransfer::GetLayerCATDEntry() is used to translate layer indexes into SDTS_CATD compatible CATD indexes.

```
printf( "Layers:\n" );
for( i = 0; i < oTransfer.GetLayerCount(); i++ )
{
    int     iCATDEntry = oTransfer.GetLayerCATDEntry(i);

    printf( "  %s: '%s'\n",
            oTransfer.GetCATD()->GetEntryModule(iCATDEntry),
            oTransfer.GetCATD()->GetEntryTypeDesc(iCATDEntry) );
}
printf( "\n" );
```

The following would be a typical layer list. Note that there are many other modules (files) registered with the catalog, but only these ones are considered to be feature layers by the SDTSTransfer object. The rest are supporting information, much of it, like data quality, is ignored by the SDTS_AL library.

```
warmerda-c[113]% sdts2shp data/SC01CATD.DDF -v
Layers:
  ASCF: 'Attribute Primary         '
  AHDR: 'Attribute Primary         '
  NP01: 'Point-Node                '
  NA01: 'Point-Node                '
  NO01: 'Point-Node                '
  LE01: 'Line                      '
  PC01: 'Polygon                   '
```

Getting a Reader

In order to read polygon features, it is necessary to instantiate a polygon reader on the desired layer. The sdts2shp.cpp program allow the user to select a module name (such as PC01, stored in pszMODN) to write to shape format. Other application might just search for, and operate on all known layers of a desired type.

The SDTSTransfer::GetLayerIndexedReader() method instantiates a reader of the desired type. In this case we know we are instantiating a SDTSPolygonReader so we can safely cast the returned SDTSIndexedReader pointer to the more specific type SDTSPolygonReader.

```
SDTSPolygonReader *poPolyReader;

poPolyReader = (SDTSPolygonReader *)
    poTransfer->GetLayerIndexedReader( poTransfer->FindLayer( pszMODN ) );

if( poPolyReader == NULL )
{
    fprintf( stderr, "Failed to open %s.\n",
             poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
    return;
}
```

Note that readers returned by SDTSTransfer::GetLayerIndexedReader() are managed by the SDTSTransfer, and should not be deleted by the application.

Collecting Polygon Geometry

The SDTS TVP format does not directly associate a polygons geometry (the points forming its boundary) with the polygon feature. Instead it is stored in separate line layers, and the lines contain references to the right, and left polygons that the lines border.

The SDTS_AL library provides a convenient method for forming the polygon geometry. Basically just call the SD↩ TSPolygonReader::AssemblePolygons() method. This method will scan all SLTLine layers in the transfer, indexing them and attaching their line work to the polygons. Then it assembles the line work into rings. It also ensures that the outer ring comes first, that the outer ring is counter-clockwise and that the inner ring(s) are clockwise.

```
poPolyReader->AssembleRings( poTransfer );
```

Upon completion the SDTSPolygonReader will have been "indexed". That means that all the polygon information will have been read from disk, and the polygon objects will now have information stored with them indicating the list of edges that form their border.

Identifying Attributes

In order to create the schema for the output shapefile dataset, it is necessary to identify the attributes associated with the polygons. There are two types of attributes which can occur. The first are hardcoded attributes specific to the feature type, and the second are generic user attributes stored in a separate primary attribute layer.

In the case of SDTSRawPolygon, there is only one attribute of interest, and that is the record number of the polygon. This is actually stored within the oModId data member of the SDTSIndexedFeature base class, as will be seen in later examples when we write it to disk. For now we create a DBF field for the record number. This record number is a unique identifier of the polygon within this module/layer.

```
nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
```

Identification of user attributes is more complicated. Any feature in a layer can have associates with 0, 1, 2 or potentially more attribute records in other primary attribute layers. In order to establish a schema for the layer it is necessary to build up a list of all attribute layers (tables) to which references appear. The SDTSIndexedReader::ScanModuleReferences() method can be used to scan a whole module for references to attribute modules via the ATID field. The return result is a list of referenced modules in the form of a string list. In a typical case this is one or two modules, such as "ASCF".

```
char  **papszModRefs = poPolyReader->ScanModuleReferences();
```

In sdts2shp.cpp, a subroutine (AddPrimaryAttrToDBFSchema()) is defined to add all the fields of all references attribute layers to the DBF file. For each module in the list the following steps are executed.

Fetch an Attribute Module Reader

The following code is similar to our code for create a polygon layer reader. It creates a reader on one of the attribute layers referenced. We explicitly rewind it since it may have been previously opened and read by another part of the application.

```
SDTSAttrReader  *poAttrReader;


poAttrReader = (SDTSAttrReader *)
    poTransfer->GetLayerIndexedReader(
        poTransfer->FindLayer( papszModuleList[iModule] ) );


if( poAttrReader == NULL )
{
    printf( "Unable to open attribute module %s, skipping.\n" ,
            papszModuleList[iModule] );
    continue;
}


poAttrReader->Rewind();
```

Get a Prototype Record

In order to get access to field definitions, and in order to establish some sort of reasonable default lengths for field without fixed lengths the sdts2shp program fetches a prototype record from the attribute module.

```
SDTSAttrRecord  *poAttrFeature;


poAttrFeature = (SDTSAttrRecord *) poAttrReader->GetNextFeature();
if( poAttrFeature == NULL )
{
    fprintf( stderr,
             "Didn't find any meaningful attribute records in %s.\n",
             papszModuleList[iModule] );


    continue;
}
```

When no longer needed, the attribute record may need to be explicitly deleted if it is not part of an indexed cached.

```
if( !poAttrReader->IsIndexed() )
    delete poAttrFeature;
```

Extract Field Definitions

The Shapefile DBF fields are defined based on the information available for each of the subfields of the attribute records ATTR DDFField (the poATTR data member). The following code loops over each of the subfields, getting a pointer to the DDBSubfieldDefn containing information about that subfield.

```
DDFFieldDefn    *poFDefn = poAttrFeature->poATTR->GetFieldDefn();
int     iSF;
DDFField    *poSR = poAttrFeature->poATTR;

for( iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
{
    DDFSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
```

Then each of the significant ISO8211 field types is translated to an appropriate DBF field type. In cases where the nWidth field is zero, indicating that the field is variable width, we use the length of the field in the prototype record. Ideally we would scan the whole file to find the longest value for each field, but that would be a significant amount of work.

```
    int     nWidth = poSFDefn->GetWidth();

    switch( poSFDefn->GetType() )
    {
      case DDFString:
        if( nWidth == 0 )
        {
            int     nMaxBytes;

            const char * pachData = poSR->GetSubfieldData(poSFDefn,
                                                          &nMaxBytes);

            nWidth = strlen(poSFDefn->ExtractStringData(pachData,
                                                        nMaxBytes, NULL ));
        }

        DBFAddField( hDBF, poSFDefn->GetName(), FTString, nWidth, 0 );
        break;

      case DDFInt:
        if( nWidth == 0 )
            nWidth = 9;

        DBFAddField( hDBF, poSFDefn->GetName(), FTInteger, nWidth, 0 );
        break;

      case DDFFloat:
        DBFAddField( hDBF, poSFDefn->GetName(), FTDouble, 18, 6 );
        break;

      default:
        fprintf( stderr,
                 "Dropping attribute `%s' of module `%s'.  "
                 "Type unsupported\n",
                 poSFDefn->GetName(),
                 papszModuleList[iModule] );
        break;
    }
}
```

Reading Polygon Features

With definition of the attribute schema out of the way, we return to the main event, reading polygons from the polygon layer. We have already instantiated the SDTSPolygonReader (poPolyReader), and now we loop reading features from it. Note that we Rewind() the reader to ensure we are starting at the beginning. After we are done process the polygon we delete it, if and only if the layer does not have an index cache.

```
SDTSRawPolygon  *poRawPoly;


poPolyReader->Rewind();
while( (poRawPoly = (SDTSRawPolygon *) poPolyReader->GetNextFeature())
       != NULL )
{
    ... process and write polygon ...


    if( !poPolyReader->IsIndexed() )
        delete poRawPoly;
}
```

Translate Geometry

In an earlier step we used the SDTSPolygonReader::AssembleRings() method to build ring geometry on the polygons from the linework in the line layers.

Coincidently (well, ok, maybe it isn't a coincidence) it so happens that the ring organization exactly matches what is needed for the shapefile api. The following call creates a polygon from the ring information in the SDTSRawPolygon. See the SDTSRawPolygon reference help for a fuller definition of the nRings, panRingStart, nVertices, and vertex fields.

```
psShape = SHPCreateObject( SHPT_POLYGON, -1, poRawPoly->nRings,
                           poRawPoly->panRingStart, NULL,
                           poRawPoly->nVertices,
                           poRawPoly->padfX,
                           poRawPoly->padfY,
                           poRawPoly->padfZ,
                           NULL );
```

Write Record Number

The following call is used to write out the record number of the polygon, fetched from the SDTSIndexedFeature::o↩ ModId data member. The szModule value in this data field will always match the module name for the whole layer. While not shown here, there is also an szOBRP field on oModId which have different values depending on whether the polygon is a universe or regular polygon.

```
DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                          poRawPoly->oModId.nRecord );
```

Fetch Associated User Records

In keeping with the setting up of the schema, accessing the user records is somewhat complicated. In sdts2shp, the primary attribute records associated with any feature (including SDTSRawPolygons) can be fetched with the WriteAttrRecordToDBF() function defined as follows.

In particular, the poFeature->nAttributes member indicates how many associated attribute records there are. The poFeature->aoATID[] array contains the SDTSModId's for each record. This SDTSModId can be passed to SDTSTransfer::GetAttr() to fetch the DDFField pointer for the user attributes. The WriteAttrRecordToDBF() method is specific to sdts2shp and will be define later.

```
    int     iAttrRecord;

    for( iAttrRecord = 0; iAttrRecord < poFeature->nAttributes; iAttrRecord++)
    {
        DDFField    *poSR;

        poSR = poTransfer->GetAttr( poFeature->aoATID+iAttrRecord );

        WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poSR );
    }
```

Write User Attributes

In a manner analogous to the definition of the fields from the prototype attribute record, the following code loops over the subfields, and fetches the data for each. The data extraction via poSR->GetSubfieldData() is a bit involved, and more information can be found on the DDFField reference page.

```
/* ------------------------------------------------------------------- */
/*      Process each subfield in the record.                           */
/* ------------------------------------------------------------------- */
    DDFFieldDefn    *poFDefn = poSR->GetFieldDefn();

    for( int iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
    {
        DDFSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
        int         iField;
        int         nMaxBytes;
        const char *    pachData = poSR->GetSubfieldData(poSFDefn,
                                                         &nMaxBytes);

/* ------------------------------------------------------------------- */
/*      Identify the related DBF field, if any.                        */
/* ------------------------------------------------------------------- */
        for( iField = 0; iField < hDBF->nFields; iField++ )
        {
            if( EQUALN(poSFDefn->GetName(),
                       hDBF->pszHeader+iField*32,10) )
                break;
        }

        if( iField == hDBF->nFields )
            iField = -1;
```

```
/* ------------------------------------------------------------------ */
/*      Handle each of the types.                                     */
/* ------------------------------------------------------------------ */
        switch( poSFDefn->GetType() )
        {
          case DDFString:
            const char  *pszValue;


            pszValue = poSFDefn->ExtractStringData(pachData, nMaxBytes,
                                                   NULL);


            if( iField != -1 )
                DBFWriteStringAttribute(hDBF, iRecord, iField, pszValue );
            break;


          case DDFFloat:
            double  dfValue;


            dfValue = poSFDefn->ExtractFloatData(pachData, nMaxBytes,
                                                 NULL);


            if( iField != -1 )
                DBFWriteDoubleAttribute( hDBF, iRecord, iField, dfValue );
            break;


          case DDFInt:
            int     nValue;


            nValue = poSFDefn->ExtractIntData(pachData, nMaxBytes, NULL);


            if( iField != -1 )
                DBFWriteIntegerAttribute( hDBF, iRecord, iField, nValue );
            break;


          default:
            break;
        }
    } /* next subfield */
```

Cleanup

In the case of sdts2shp, the SDTSTransfer is created on the stack. When it falls out of scope it is destroyed, and all the indexed readers, and their indexed features caches are also cleaned up.

Chapter 4

sdts2shp.cpp

<title>SDTS To Shape Example Application</title>

```
/* ******************************************************************************
 *
 * Project:  SDTS Translator
 * Purpose:  Mainline for converting to ArcView Shapefiles.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 1999, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 ******************************************************************************/

#include "sdts_al.h"
#include "shapefil.h"
#include "cpl_string.h"

CPL_CVSID("$Id: sdts2shp.cpp 3b0bbf7a8a012d69a783ee1f9cfeb5c52b370021 2017-06-27 20:57:02Z Even Rouault $")

static int  bVerbose = FALSE;

static void WriteLineShapefile( const char *, SDTSTransfer *,
                                const char * );
static void WritePointShapefile( const char *, SDTSTransfer *,
                                 const char * );
static void WriteAttributeDBF( const char *, SDTSTransfer *,
                               const char * );
static void WritePolygonShapefile( const char *, SDTSTransfer *,
                                   const char * );

static void
AddPrimaryAttrToDBFSchema( DBFHandle hDBF, SDTSTransfer * poTransfer,
                           char ** papszModuleList );
static void
WritePrimaryAttrToDBF( DBFHandle hDBF, int nRecord,
                       SDTSTransfer *, SDTSFeature * poFeature );
static void
WriteAttrRecordToDBF( DBFHandle hDBF, int nRecord,
                      SDTSTransfer *, DDFField * poAttributes );

/* ******************************************************************/
/*                              Usage()                             */
```

```
/* ************************************************************************/

static void Usage()

{
    printf( "Usage: sdts2shp CATD_filename [-o shapefile_name]\n" /*ok*/
            "                 [-m module_name] [-v]\n"
            "\n"
            "Modules include 'LE01', 'PC01', 'NP01' and 'ARDF'\n" );

    exit( 1 );
}

/* ************************************************************************/
/*                                main()                                  */
/* ************************************************************************/

int main( int nArgc, char ** papszArgv )

{
{
    int         i;
    const char  *pszCATDFilename = NULL;
    const char  *pszMODN = "LE01";
    char        *pszShapefile = "sdts_out.shp";
    SDTSTransfer oTransfer;

/* -------------------------------------------------------------------- */
/*      Interpret commandline switches.                                 */
/* -------------------------------------------------------------------- */
    if( nArgc < 2 )
        Usage();

    pszCATDFilename = papszArgv[1];

    for( i = 2; i < nArgc; i++ )
    {
        if( EQUAL(papszArgv[i],"-m") && i+1 < nArgc )
            pszMODN = papszArgv[++i];
        else if( EQUAL(papszArgv[i],"-o") && i+1 < nArgc )
            pszShapefile = papszArgv[++i];
        else if( EQUAL(papszArgv[i],"-v") )
            bVerbose = TRUE;
        else
        {
            printf( "Incomplete, or unsupported option '%s'\n\n",/*ok*/
                    papszArgv[i] );
            Usage();
        }
    }

/* -------------------------------------------------------------------- */
/*      Massage shapefile name to have no extension.                    */
/* -------------------------------------------------------------------- */
    pszShapefile = CPLStrdup(pszShapefile);
    for( i = strlen(pszShapefile)-1; i >= 0; i-- )
    {
        if( pszShapefile[i] == '.' )
        {
            pszShapefile[i] = '\0';
            break;
        }
        else if( pszShapefile[i] == '/' || pszShapefile[i] == '\\' )
            break;
    }

/* -------------------------------------------------------------------- */
/*      Open the transfer.                                              */
/* -------------------------------------------------------------------- */
    if( !oTransfer.Open( pszCATDFilename ) )
    {
        fprintf( stderr,
                 "Failed to read CATD file '%s'\n",
                 pszCATDFilename );
        exit( 100 );
    }

/* -------------------------------------------------------------------- */
/*      Dump available layer in verbose mode.                           */
/* -------------------------------------------------------------------- */
    if( bVerbose )
    {
        printf( "Layers:\n" );/*ok*/
        for( i = 0; i < oTransfer.GetLayerCount(); i++ )
        {
            int         iCATDEntry = oTransfer.GetLayerCATDEntry(i);
```

```
            printf( "  %s: `%s'\n",/*ok*/
                    oTransfer.GetCATD()->GetEntryModule(iCATDEntry),
                    oTransfer.GetCATD()->GetEntryTypeDesc(iCATDEntry) );
        }
        printf( "\n" );/*ok*/
    }

/* ------------------------------------------------------------------- */
/*      Check that module exists.                                      */
/* ------------------------------------------------------------------- */
    if( oTransfer.FindLayer( pszMODN ) == -1 )
    {
        fprintf( stderr, "Unable to identify module: %s\n", pszMODN );
        exit( 1 );
    }

/* ------------------------------------------------------------------- */
/*      If the module is an LE module, write it to an Arc file.        */
/* ------------------------------------------------------------------- */
    if( pszMODN[0] == 'L' || pszMODN[0] == 'l' )
    {
        WriteLineShapefile( pszShapefile, &oTransfer, pszMODN );
    }

/* ------------------------------------------------------------------- */
/*      If the module is an attribute primary one, dump to DBF.        */
/* ------------------------------------------------------------------- */
    else if( pszMODN[0] == 'A' || pszMODN[0] == 'a'
             || pszMODN[0] == 'B' || pszMODN[0] == 'b' )
    {
        WriteAttributeDBF( pszShapefile, &oTransfer, pszMODN );
    }

/* ------------------------------------------------------------------- */
/*      If the module is a point one, dump to Shapefile.               */
/* ------------------------------------------------------------------- */
    else if( pszMODN[0] == 'N' || pszMODN[0] == 'n' )
    {
        WritePointShapefile( pszShapefile, &oTransfer, pszMODN );
    }

/* ------------------------------------------------------------------- */
/*      If the module is a polygon one, dump to Shapefile.             */
/* ------------------------------------------------------------------- */
    else if( pszMODN[0] == 'P' || pszMODN[0] == 'p' )
    {
        WritePolygonShapefile( pszShapefile, &oTransfer, pszMODN );
    }

    else
    {
        fprintf( stderr, "Unrecognized module name: %s\n", pszMODN );
    }

    CPLFree( pszShapefile );
}
#ifdef DBMALLOC
    malloc_dump(1);
#endif
}

/* ********************************************************************/
/*                         WriteLineShapefile()                       */
/* ********************************************************************/

static void WriteLineShapefile( const char * pszShapefile,
                                SDTSTransfer * poTransfer,
                                const char * pszMODN )

{
/* ------------------------------------------------------------------- */
/*      Fetch a reference to the indexed Pointgon reader.               */
/* ------------------------------------------------------------------- */
    SDTSLineReader *poLineReader = (SDTSLineReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
      FindLayer( pszMODN ) );

    if( poLineReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
                 poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

    poLineReader->Rewind();

/* ------------------------------------------------------------------- */
```

```
/*      Create the Shapefile.                                           */
/* ---------------------------------------------------------------------- */
    SHPHandle   hSHP;

    hSHP = SHPCreate( pszShapefile, SHPT_ARC );
    if( hSHP == NULL )
    {
        fprintf( stderr, "Unable to create shapefile `%s'\n",
                 pszShapefile );
        return;
    }

/* ---------------------------------------------------------------------- */
/*      Create the database file, and our basic set of attributes.      */
/* ---------------------------------------------------------------------- */
    DBFHandle   hDBF;
    int         nLeftPolyField, nRightPolyField;
    int         nStartNodeField, nEndNodeField, nSDTSRecordField;
    char        szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for `%s'\n",
                 pszShapefile );
        return;
    }

    nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
    nLeftPolyField = DBFAddField( hDBF, "LeftPoly", FTString, 12, 0 );
    nRightPolyField = DBFAddField( hDBF, "RightPoly", FTString, 12, 0 );
    nStartNodeField = DBFAddField( hDBF, "StartNode", FTString, 12, 0 );
    nEndNodeField = DBFAddField( hDBF, "EndNode", FTString, 12, 0 );

    char  **papszModRefs = poLineReader->ScanModuleReferences();
    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
    CSLDestroy( papszModRefs );

/* ====================================================================== */
/*      Process all the line features in the module.                    */
/* ====================================================================== */
    SDTSRawLine *poRawLine = NULL;

    while( (poRawLine = poLineReader->GetNextLine()) != NULL )
    {
/* ---------------------------------------------------------------------- */
/*      Write out a shape with the vertices.                            */
/* ---------------------------------------------------------------------- */
        SHPObject *psShape =
            SHPCreateSimpleObject( SHPT_ARC, poRawLine->nVertices,
                                   poRawLine->padfX, poRawLine->padfY,
                                   poRawLine->padfZ );

        int iShape = SHPWriteObject( hSHP, -1, psShape );

        SHPDestroyObject( psShape );

/* ---------------------------------------------------------------------- */
/*      Write out the attributes.                                       */
/* ---------------------------------------------------------------------- */
        char    szID[13];

        DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                                  poRawLine->oModId.nRecord );

        sprintf( szID, "%s:%d",
                 poRawLine->oLeftPoly.szModule,
                 poRawLine->oLeftPoly.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nLeftPolyField, szID );

        sprintf( szID, "%s:%d",
                 poRawLine->oRightPoly.szModule,
                 poRawLine->oRightPoly.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nRightPolyField, szID );

        sprintf( szID, "%s:%d",
                 poRawLine->oStartNode.szModule,
                 poRawLine->oStartNode.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nStartNodeField, szID );

        sprintf( szID, "%s:%d",
                 poRawLine->oEndNode.szModule,
                 poRawLine->oEndNode.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nEndNodeField, szID );
```

```
        WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawLine );

        if( !poLineReader->IsIndexed() )
            delete poRawLine;
    }

/* ------------------------------------------------------------------- */
/*      Close, and cleanup.                                            */
/* ------------------------------------------------------------------- */
    DBFClose( hDBF );
    SHPClose( hSHP );
}

/* ********************************************************************/
/*                        WritePointShapefile()                       */
/* ********************************************************************/

static void WritePointShapefile( const char * pszShapefile,
                                 SDTSTransfer * poTransfer,
                                 const char * pszMODN )

{
/* ------------------------------------------------------------------- */
/*      Fetch a reference to the indexed Pointgon reader.               */
/* ------------------------------------------------------------------- */
    SDTSPointReader *poPointReader = (SDTSPointReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
      FindLayer( pszMODN ) );

    if( poPointReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
                 poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

    poPointReader->Rewind();

/* ------------------------------------------------------------------- */
/*      Create the Shapefile.                                          */
/* ------------------------------------------------------------------- */
    SHPHandle   hSHP;

    hSHP = SHPCreate( pszShapefile, SHPT_POINT );
    if( hSHP == NULL )
    {
        fprintf( stderr, "Unable to create shapefile `%s'\n",
                 pszShapefile );
        return;
    }

/* ------------------------------------------------------------------- */
/*      Create the database file, and our basic set of attributes.      */
/* ------------------------------------------------------------------- */
    DBFHandle   hDBF;
    int         nAreaField, nSDTSRecordField;
    char        szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for `%s'\n",
                 pszShapefile );
        return;
    }

    nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
    nAreaField = DBFAddField( hDBF, "AreaId", FTString, 12, 0 );

    char  **papszModRefs = poPointReader->ScanModuleReferences();
    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
    CSLDestroy( papszModRefs );

/* ================================================================== */
/*      Process all the line features in the module.                   */
/* ================================================================== */
    SDTSRawPoint *poRawPoint = NULL;

    while( (poRawPoint = poPointReader->GetNextPoint()) != NULL )
    {
/* ------------------------------------------------------------------- */
/*      Write out a shape with the vertices.                           */
/* ------------------------------------------------------------------- */
        SHPObject *psShape =
            SHPCreateSimpleObject( SHPT_POINT, 1,
```

```
                                &(poRawPoint->dfX),
                                &(poRawPoint->dfY),
                                &(poRawPoint->dfZ) );

        int iShape = SHPWriteObject( hSHP, -1, psShape );

        SHPDestroyObject( psShape );

/* ------------------------------------------------------------------- */
/*      Write out the attributes.                                      */
/* ------------------------------------------------------------------- */
        char    szID[13];

        DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                                  poRawPoint->oModId.nRecord );

        sprintf( szID, "%s:%d",
                 poRawPoint->oAreaId.szModule,
                 poRawPoint->oAreaId.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nAreaField, szID );

        WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawPoint );

        if( !poPointReader->IsIndexed() )
            delete poRawPoint;
    }

/* ------------------------------------------------------------------- */
/*      Close, and cleanup.                                            */
/* ------------------------------------------------------------------- */
    DBFClose( hDBF );
    SHPClose( hSHP );
}

/* ********************************************************************/
/*                         WriteAttributeDBF()                        */
/* ********************************************************************/

static void WriteAttributeDBF( const char * pszShapefile,
                               SDTSTransfer * poTransfer,
                               const char * pszMODN )

{
/* ------------------------------------------------------------------- */
/*      Fetch a reference to the indexed Pointgon reader.              */
/* ------------------------------------------------------------------- */
    SDTSAttrReader *poAttrReader = (SDTSAttrReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
      FindLayer( pszMODN ) );

    if( poAttrReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
                 poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

    poAttrReader->Rewind();

/* ------------------------------------------------------------------- */
/*      Create the database file, and our basic set of attributes.     */
/* ------------------------------------------------------------------- */
    DBFHandle   hDBF;
    char        szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for `%s'\n",
                 pszShapefile );
        return;
    }

    DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );

/* ------------------------------------------------------------------- */
/*      Prepare the schema.                                            */
/* ------------------------------------------------------------------- */
    char        **papszMODNList = CSLAddString( NULL, pszMODN );

    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszMODNList );

    CSLDestroy( papszMODNList );

/* ================================================================== */
```

```
/*      Process all the records in the module.                          */
/* ================================================================= */
    SDTSAttrRecord *poRecord = NULL;
    int iRecord = 0;

    while( (poRecord = (SDTSAttrRecord*)poAttrReader->
      GetNextFeature())
           != NULL )
    {
        DBFWriteIntegerAttribute( hDBF, iRecord, 0,
                                  poRecord->oModId.nRecord );

        WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poRecord->poATTR );

        if( !poAttrReader->IsIndexed() )
            delete poRecord;

        iRecord++;
    }

/* ----------------------------------------------------------------- */
/*      Close, and cleanup.                                             */
/* ----------------------------------------------------------------- */
    DBFClose( hDBF );
}

/* ***********************************************************************/
/*                        WritePolygonShapefile()                      */
/* ***********************************************************************/

static void WritePolygonShapefile( const char * pszShapefile,
                                   SDTSTransfer * poTransfer,
                                   const char * pszMODN )

{
/* ----------------------------------------------------------------- */
/*      Fetch a reference to the indexed polygon reader.                */
/* ----------------------------------------------------------------- */
    SDTSPolygonReader *poPolyReader = (SDTSPolygonReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
      FindLayer( pszMODN ) );

    if( poPolyReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
                 poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

/* ----------------------------------------------------------------- */
/*      Assemble polygon geometries from all the line layers.           */
/* ----------------------------------------------------------------- */
    poPolyReader->AssembleRings( poTransfer, poTransfer->FindLayer(pszMODN) );

/* ----------------------------------------------------------------- */
/*      Create the Shapefile.                                           */
/* ----------------------------------------------------------------- */
    SHPHandle   hSHP;

    hSHP = SHPCreate( pszShapefile, SHPT_POLYGON );
    if( hSHP == NULL )
    {
        fprintf( stderr, "Unable to create shapefile `%s'\n",
                 pszShapefile );
        return;
    }

/* ----------------------------------------------------------------- */
/*      Create the database file, and our basic set of attributes.      */
/* ----------------------------------------------------------------- */
    DBFHandle   hDBF;
    int         nSDTSRecordField;
    char        szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for `%s'\n",
                 pszShapefile );
        return;
    }

    nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );

    char  **papszModRefs = poPolyReader->ScanModuleReferences();
```

```
    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
    CSLDestroy( papszModRefs );

/* ==================================================================== */
/*      Process all the polygon features in the module.                 */
/* ==================================================================== */
    poPolyReader->Rewind();

    SDTSRawPolygon *poRawPoly = NULL;
    while( (poRawPoly = (SDTSRawPolygon *) poPolyReader->
      GetNextFeature())
           != NULL )
    {
/* -------------------------------------------------------------------- */
/*      Write out a shape with the vertices.                            */
/* -------------------------------------------------------------------- */
        SHPObject *psShape =
            SHPCreateObject( SHPT_POLYGON, -1, poRawPoly->nRings,
                             poRawPoly->panRingStart, NULL,
                             poRawPoly->nVertices,
                             poRawPoly->padfX,
                             poRawPoly->padfY,
                             poRawPoly->padfZ,
                             NULL );

        int iShape = SHPWriteObject( hSHP, -1, psShape );

        SHPDestroyObject( psShape );

/* -------------------------------------------------------------------- */
/*      Write out the attributes.                                       */
/* -------------------------------------------------------------------- */
        DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                                  poRawPoly->oModId.nRecord );
        WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawPoly );

        if( !poPolyReader->IsIndexed() )
            delete poRawPoly;
    }

/* -------------------------------------------------------------------- */
/*      Close, and cleanup.                                             */
/* -------------------------------------------------------------------- */
    DBFClose( hDBF );
    SHPClose( hSHP );
}

/* ********************************************************************/
/*                        AddPrimaryAttrToDBF()                        */
/*                                                                     */
/*      Add the fields from all the given primary attribute modules    */
/*      to the schema of the passed DBF file.                          */
/* ********************************************************************/

static void
AddPrimaryAttrToDBFSchema( DBFHandle hDBF, SDTSTransfer *poTransfer,
                           char ** papszModuleList )

{
    for( int iModule = 0;
         papszModuleList != NULL && papszModuleList[iModule] != NULL;
         iModule++ )
    {
/* -------------------------------------------------------------------- */
/*      Get a reader on the desired module.                             */
/* -------------------------------------------------------------------- */
        SDTSAttrReader *poAttrReader = (SDTSAttrReader *)
            poTransfer->GetLayerIndexedReader(
                poTransfer->FindLayer( papszModuleList[iModule] ) );

        if( poAttrReader == NULL )
        {
            printf( "Unable to open attribute module %s, skipping.\n" ,/*ok*/
                    papszModuleList[iModule] );
            continue;
        }

        poAttrReader->Rewind();

/* -------------------------------------------------------------------- */
/*      Read the first record so we can clone schema information off    */
/*      of it.                                                          */
/* -------------------------------------------------------------------- */
        SDTSAttrRecord *poAttrFeature =
            (SDTSAttrRecord *) poAttrReader->GetNextFeature();
        if( poAttrFeature == NULL )
        {
```

```
            fprintf( stderr,
                     "Didn't find any meaningful attribute records in %s.\n",
                     papszModuleList[iModule] );

            continue;
        }

/* ------------------------------------------------------------------- */
/*      Clone schema off the first record.  Eventually we need to      */
/*      get the information out of the DDR record, but it isn't        */
/*      clear to me how to accomplish that with the SDTS++ API.        */
/*                                                                     */
/*      The following approach may fail (dramatically) if some         */
/*      records do not include all subfields.  Furthermore, no         */
/*      effort is made to make DBF field names unique.  The SDTS       */
/*      attributes often have names much beyond the 14 character dbf   */
/*      limit which may result in non-unique attributes.               */
/* ------------------------------------------------------------------- */
        DDFFieldDefn    *poFDefn = poAttrFeature->poATTR->
      GetFieldDefn();
        int             iSF;
        DDFField        *poSR = poAttrFeature->poATTR;

        for( iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
        {
            DDFSubfieldDefn     *poSFDefn = poFDefn->GetSubfield( iSF );
            int         nWidth = poSFDefn->GetWidth();

            switch( poSFDefn->GetType() )
            {
              case DDFString:
                if( nWidth == 0 )
                {
                    int         nMaxBytes;

                    const char * pachData = poSR->GetSubfieldData(poSFDefn,
                                                                  &nMaxBytes);

                    nWidth = strlen(poSFDefn->ExtractStringData(pachData,
                                                                nMaxBytes, NULL ));
                }

                DBFAddField( hDBF, poSFDefn->GetName(), FTString, nWidth, 0 );
                break;

              case DDFInt:
                if( nWidth == 0 )
                    nWidth = 9;

                DBFAddField( hDBF, poSFDefn->GetName(), FTInteger, nWidth, 0 );
                break;

              case DDFFloat:
                DBFAddField( hDBF, poSFDefn->GetName(), FTDouble, 18, 6 );
                break;

              default:
                fprintf( stderr,
                         "Dropping attribute '%s' of module '%s'.  "
                         "Type unsupported\n",
                         poSFDefn->GetName(),
                         papszModuleList[iModule] );
                break;
            }
        }

        if( !poAttrReader->IsIndexed() )
            delete poAttrFeature;
    } /* next module */
}

/* ***********************************************************************/
/*                        WritePrimaryAttrToDBF()                      */
/* ***********************************************************************/

static void
WritePrimaryAttrToDBF( DBFHandle hDBF, int iRecord,
                       SDTSTransfer * poTransfer, SDTSFeature * poFeature )

{
/* ------------------------------------------------------------------- */
/*      Loop over all the attribute records linked to this feature.    */
/* ------------------------------------------------------------------- */
    for( int iAttrRecord = 0;
         iAttrRecord < poFeature->nAttributes;
         iAttrRecord++ )
    {
```

```
        DDFField *poSR = poTransfer->GetAttr( poFeature->paoATID+iAttrRecord );

        WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poSR );
    }
}

/* ************************************************************************/
/*                        WriteAttrRecordToDBF()                          */
/* ************************************************************************/

static void
WriteAttrRecordToDBF( DBFHandle hDBF, int iRecord,
                      SDTSTransfer * poTransfer, DDFField * poSR )

{
/* ------------------------------------------------------------------- */
/*      Process each subfield in the record.                           */
/* ------------------------------------------------------------------- */
    DDFFieldDefn        *poFDefn = poSR->GetFieldDefn();

    for( int iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
    {
        DDFSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
        int                     iField;
        int                     nMaxBytes;
        const char *    pachData = poSR->GetSubfieldData(poSFDefn,
                                                         &nMaxBytes);

/* ------------------------------------------------------------------- */
/*      Identify the related DBF field, if any.                        */
/* ------------------------------------------------------------------- */
        for( iField = 0; iField < hDBF->nFields; iField++ )
        {
            if( EQUALN(poSFDefn->GetName(),
                       hDBF->pszHeader+iField*32,10) )
                break;
        }

        if( iField == hDBF->nFields )
            iField = -1;

/* ------------------------------------------------------------------- */
/*      Handle each of the types.                                      */
/* ------------------------------------------------------------------- */
        switch( poSFDefn->GetType() )
        {
          case DDFString:
            const char *pszValue
                = poSFDefn->ExtractStringData(pachData, nMaxBytes, NULL);

            if( iField != -1 )
                DBFWriteStringAttribute(hDBF, iRecord, iField, pszValue );
            break;

          case DDFFloat:
            double      dfValue;

            dfValue = poSFDefn->ExtractFloatData(pachData, nMaxBytes,
                                                 NULL);

            if( iField != -1 )
                DBFWriteDoubleAttribute( hDBF, iRecord, iField, dfValue );
            break;

          case DDFInt:
            int         nValue;

            nValue = poSFDefn->ExtractIntData(pachData, nMaxBytes, NULL);

            if( iField != -1 )
                DBFWriteIntegerAttribute( hDBF, iRecord, iField, nValue );
            break;

          default:
            break;
        }
    } /* next subfield */
}
```

Chapter 5

ISO8211_Example

```
/* ******************************************************************************
 *
 * Project:  SDTS Translator
 * Purpose:  Example program dumping data in 8211 data to stdout.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 1999, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 ******************************************************************************/

#include <stdio.h>
#include "iso8211.h"

#include <algorithm>

CPL_CVSID("$Id: 8211view.cpp 7e07230bbff24eb333608de4dbd460b7312839d0 2017-12-11 19:08:47Z Even Rouault $")

static void ViewRecordField( DDFField * poField );
static int ViewSubfield( DDFSubfieldDefn *poSFDefn,
                         const char * pachFieldData,
                         int nBytesRemaining );

/* ************************************************************************/
/*                                main()                                  */
/* ************************************************************************/

int main( int nArgc, char ** papszArgv )

{
    DDFModule   oModule;
    const char  *pszFilename = nullptr;
    int         bFSPTHack = FALSE;

    for( int iArg = 1; iArg < nArgc; iArg++ )
    {
        if( EQUAL(papszArgv[iArg],"-fspt_repeating") )
            bFSPTHack = TRUE;
        else
            pszFilename = papszArgv[iArg];
    }

    if( pszFilename == nullptr )
    {
```

```
        printf( "Usage: 8211view filename\n" );
        exit( 1 );
    }

/* ------------------------------------------------------------------- */
/*      Open the file.  Note that by default errors are reported to    */
/*      stderr, so we don't bother doing it ourselves.                 */
/* ------------------------------------------------------------------- */
    if( !oModule.Open( pszFilename ) )
    {
        exit( 1 );
    }

    if( bFSPTHack )
    {
        DDFFieldDefn *poFSPT = oModule.FindFieldDefn( "FSPT" );

        if( poFSPT == nullptr )
            fprintf( stderr,
                     "unable to find FSPT field to set repeating flag.\n" );
        else
            poFSPT->SetRepeatingFlag( TRUE );
    }

/* ------------------------------------------------------------------- */
/*      Loop reading records till there are none left.                 */
/* ------------------------------------------------------------------- */
    DDFRecord *poRecord = nullptr;
    int iRecord = 0;

    while( (poRecord = oModule.ReadRecord()) != nullptr )
    {
        printf( "Record %d (%d bytes)\n",
                ++iRecord, poRecord->GetDataSize() );

        /* ------------------------------------------------------------ */
        /*      Loop over each field in this particular record.         */
        /* ------------------------------------------------------------ */
        for( int iField = 0; iField < poRecord->GetFieldCount(); iField++ )
        {
            DDFField    *poField = poRecord->GetField( iField );

            ViewRecordField( poField );
        }
    }
}

/* ***********************************************************************/
/*                           ViewRecordField()                         */
/*                                                                     */
/*      Dump the contents of a field instance in a record.             */
/* ***********************************************************************/

static void ViewRecordField( DDFField * poField )

{
    DDFFieldDefn *poFieldDefn = poField->GetFieldDefn();

    // Report general information about the field.
    printf( "    Field %s: %s\n",
            poFieldDefn->GetName(), poFieldDefn->GetDescription() );

    // Get pointer to this fields raw data.  We will move through
    // it consuming data as we report subfield values.

    const char  *pachFieldData = poField->GetData();
    int nBytesRemaining = poField->GetDataSize();

    /* ------------------------------------------------------ */
    /*      Loop over the repeat count for this fields        */
    /*      subfields.  The repeat count will almost          */
    /*      always be one.                                    */
    /* ------------------------------------------------------ */
    for( int iRepeat = 0; iRepeat < poField->GetRepeatCount(); iRepeat++ )
    {

        /* ------------------------------------------------------ */
        /*   Loop over all the subfields of this field, advancing */
        /*   the data pointer as we consume data.                 */
        /* ------------------------------------------------------ */
        for( int iSF = 0; iSF < poFieldDefn->GetSubfieldCount(); iSF++ )
        {
            DDFSubfieldDefn *poSFDefn = poFieldDefn->GetSubfield( iSF );
            int nBytesConsumed =
                ViewSubfield( poSFDefn, pachFieldData, nBytesRemaining );

            nBytesRemaining -= nBytesConsumed;
```

```
            pachFieldData += nBytesConsumed;
        }
    }
}

/* ************************************************************************/
/*                            ViewSubfield()                              */
/* ************************************************************************/

static int ViewSubfield( DDFSubfieldDefn *poSFDefn,
                         const char * pachFieldData,
                         int nBytesRemaining )

{
    int         nBytesConsumed = 0;

    switch( poSFDefn->GetType() )
    {
      case DDFInt:
        if( poSFDefn->GetBinaryFormat() == DDFSubfieldDefn::UInt )
            printf( "        %s = %u\n",
                    poSFDefn->GetName(),
                    static_cast<unsigned int>(
                        poSFDefn->ExtractIntData( pachFieldData, nBytesRemaining,
                                                  &nBytesConsumed )) );
        else
            printf( "        %s = %d\n",
                    poSFDefn->GetName(),
                    poSFDefn->ExtractIntData( pachFieldData, nBytesRemaining,
                                              &nBytesConsumed ) );
        break;

      case DDFFloat:
        printf( "        %s = %f\n",
                poSFDefn->GetName(),
                poSFDefn->ExtractFloatData( pachFieldData, nBytesRemaining,
                                            &nBytesConsumed ) );
        break;

      case DDFString:
        printf( "        %s = '%s'\n",
                poSFDefn->GetName(),
                poSFDefn->ExtractStringData( pachFieldData, nBytesRemaining,
                                             &nBytesConsumed ) );
        break;

      case DDFBinaryString:
      {
          int   i;
          //rjensen 19-Feb-2002 5 integer variables to decode NAME and LNAM
          int vrid_rcnm=0;
          int vrid_rcid=0;
          int foid_agen=0;
          int foid_find=0;
          int foid_fids=0;

          GByte *pabyBString = (GByte *)
              poSFDefn->ExtractStringData( pachFieldData, nBytesRemaining,
                                           &nBytesConsumed );

          printf( "        %s = 0x", poSFDefn->GetName() );
          for( i = 0; i < std::min(nBytesConsumed, 24); i++ )
              printf( "%02X", pabyBString[i] );

          if( nBytesConsumed > 24 )
              printf( "%s", "..." );

          // rjensen 19-Feb-2002 S57 quick hack. decode NAME and LNAM bitfields
          if ( EQUAL(poSFDefn->GetName(),"NAME") )
          {
              vrid_rcnm=pabyBString[0];
              vrid_rcid=pabyBString[1] + (pabyBString[2]*256)+
                  (pabyBString[3]*65536)+ (pabyBString[4]*16777216);
              printf("\tVRID RCNM = %d,RCID = %d",vrid_rcnm,vrid_rcid);
          }
          else if ( EQUAL(poSFDefn->GetName(),"LNAM") )
          {
              foid_agen=pabyBString[0] + (pabyBString[1]*256);
              foid_find=pabyBString[2] + (pabyBString[3]*256)+
                  (pabyBString[4]*65536)+ (pabyBString[5]*16777216);
              foid_fids=pabyBString[6] + (pabyBString[7]*256);
              printf("\tFOID AGEN = %d,FIDN = %d,FIDS = %d",
                     foid_agen,foid_find,foid_fids);
          }

          printf( "\n" );
      }
```

```
      break;
    }

    return nBytesConsumed;
}
```

Chapter 6

Deprecated List

Member CPL_LSBINT16PTR (x)

Use rather CPL_LSBSINT16PTR or CPL_LSBUINT16PTR for explicit signedness.

Member CPL_LSBINT32PTR (x)

Use rather CPL_LSBSINT32PTR or CPL_LSBUINT32PTR for explicit signedness.

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

_CPL_SHA256Context . . . 43
_CPLHashSet . . . 43
_CPLList . . . 43
_CPLLock . . . 44
_CPLQuadTree . . . 44
_CPLSpawnedProcess . . . 44
_CPLSpinLock . . . 44
_MutexLinkedElt . . . 45
_QuadTreeNode . . . 45
CPL_SHA1Context . . . 45
CPLErrorContext . . . 45
CPLHTTPResult . . . 45
CPLJSONDocument . . . 53
CPLJSONObject . . . 57
    CPLJSONArray . . . 47
CPLMD5Context . . . 73
CPLMimePart . . . 73
CPLODBCDriverInstaller . . . 74
CPLODBCSession . . . 76
CPLODBCStatement . . . 79
CPLRectObj . . . 92
CPLSharedFileInfo . . . 93
CPLSharedFileInfoExtra . . . 94
CPLStdCallThreadInfo . . . 94
CPLVirtualMem . . . 94
CPLWorkerThread . . . 95
CPLWorkerThreadJob . . . 95
CPLWorkerThreadPool . . . 95
CPLXMLNode . . . 98
CPLZip . . . 99
ctb . . . 99
curfile_info . . . 100
DDFField . . . 100
DDFFieldDefn . . . 102
DDFModule . . . 105

DDFRecord . . . 109
DDFSubfieldDefn . . . 118
DefaultCSVFileNameTLS . . . 124
errHandler . . . 124
file_in_zip_read_info_s . . . 125
FindFileTLS . . . 125
GDALPamDataset
  SDTSDataset . . . 131
GDALPamRasterBand
  SDTSRasterBand . . . 139
GDALScaledProgressInfo . . . 125
linkedlist_data_s . . . 125
linkedlist_datablock_internal_s . . . 125
ParseContext . . . 125
SDTS_CATD . . . 125
SDTS_CATDEntry . . . 127
SDTS_IREF . . . 127
SDTS_XREF . . . 128
SDTSFeature . . . 131
  SDTSAttrRecord . . . 129
  SDTSRawLine . . . 142
  SDTSRawPoint . . . 145
  SDTSRawPolygon . . . 146
SDTSIndexedReader . . . 133
  SDTSAttrReader . . . 129
  SDTSLineReader . . . 135
  SDTSPointReader . . . 137
  SDTSPolygonReader . . . 138
SDTSModId . . . 136
SDTSRasterReader . . . 139
SDTSTransfer . . . 149
SFRegion . . . 154
StackContext . . . 155
tm_unz_s . . . 155
tm_zip_s . . . 155
TupleEnvVarOptionName . . . 155
unz_file_info_internal_s . . . 155
unz_file_info_s . . . 155
unz_file_pos_s . . . 156
unz_global_info_s . . . 156
unz_s . . . 156
VSIArchiveContent . . . 156
VSIArchiveEntry . . . 156
VSIArchiveEntryFileOffset . . . 156
VSIArchiveReader . . . 157
VSIDIR . . . 157
VSIErrorContext . . . 158
VSIFileManager . . . 158
VSIFilesystemHandler . . . 158
  VSIArchiveFilesystemHandler . . . 157
  VSIDummyCryptFilesystemHandler . . . 157
  VSISparseFileFilesystemHandler . . . 158
  VSISubFileFilesystemHandler . . . 162
VSIReadDirRecursiveTask . . . 158
VSIVirtualHandle . . . 166
  VSISparseFileHandle . . . 159
  VSISubFileHandle . . . 163

zip_fileinfo 172
zip_internal 173
zlib_filefunc_def_s 173

Chapter 8

Class Index

8.1 Class List

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

_CPL_SHA256Context . . . 43
_CPLHashSet . . . 43
_CPLList . . . 43
_CPLLock . . . 44
_CPLQuadTree . . . 44
_CPLSpawnedProcess . . . 44
_CPLSpinLock . . . 44
_MutexLinkedElt . . . 45
_QuadTreeNode . . . 45
CPL_SHA1Context . . . 45
CPLErrorContext . . . 45
CPLHTTPResult . . . 45
CPLJSONArray
The JSONArray class JSON array from JSONDocument . . . 47
CPLJSONDocument
Wrapper class around json-c library . . . 53
CPLJSONObject
The CPLJSONArray class holds JSON object from CPLJSONDocument . . . 57
CPLMD5Context . . . 73
CPLMimePart . . . 73
CPLODBCDriverInstaller . . . 74
CPLODBCSession . . . 76
CPLODBCStatement . . . 79
CPLRectObj . . . 92
CPLSharedFileInfo . . . 93
CPLSharedFileInfoExtra . . . 94
CPLStdCallThreadInfo . . . 94
CPLVirtualMem . . . 94
CPLWorkerThread . . . 95
CPLWorkerThreadJob . . . 95
CPLWorkerThreadPool . . . 95
CPLXMLNode . . . 98
CPLZip . . . 99
ctb . . . 99
curfile_info . . . 100

DDFField . . . 100
DDFFieldDefn . . . 102
DDFModule . . . 105
DDFRecord . . . 109
DDFSubfieldDefn . . . 118
DefaultCSVFileNameTLS . . . 124
errHandler . . . 124
file_in_zip_read_info_s . . . 125
FindFileTLS . . . 125
GDALScaledProgressInfo . . . 125
linkedlist_data_s . . . 125
linkedlist_datablock_internal_s . . . 125
ParseContext . . . 125
SDTS_CATD . . . 125
SDTS_CATDEntry . . . 127
SDTS_IREF . . . 127
SDTS_XREF . . . 128
SDTSAttrReader . . . 129
SDTSAttrRecord . . . 129
SDTSDataset . . . 131
SDTSFeature . . . 131
SDTSIndexedReader . . . 133
SDTSLineReader . . . 135
SDTSModId . . . 136
SDTSPointReader . . . 137
SDTSPolygonReader . . . 138
SDTSRasterBand . . . 139
SDTSRasterReader . . . 139
SDTSRawLine . . . 142
SDTSRawPoint . . . 145
SDTSRawPolygon . . . 146
SDTSTransfer . . . 149
SFRegion . . . 154
StackContext . . . 155
tm_unz_s . . . 155
tm_zip_s . . . 155
TupleEnvVarOptionName . . . 155
unz_file_info_internal_s . . . 155
unz_file_info_s . . . 155
unz_file_pos_s . . . 156
unz_global_info_s . . . 156
unz_s . . . 156
VSIArchiveContent . . . 156
VSIArchiveEntry . . . 156
VSIArchiveEntryFileOffset . . . 156
VSIArchiveFilesystemHandler . . . 157
VSIArchiveReader . . . 157
VSIDIR . . . 157
VSIDummyCryptFilesystemHandler . . . 157
VSIErrorContext . . . 158
VSIFileManager . . . 158
VSIFilesystemHandler . . . 158
VSIReadDirRecursiveTask . . . 158
VSISparseFileFilesystemHandler . . . 158
VSISparseFileHandle . . . 159
VSISubFileFilesystemHandler . . . 162
VSISubFileHandle . . . 163
VSIVirtualHandle . . . 166

zip_fileinfo 172
zip_internal 173
zlib_filefunc_def_s 173

Chapter 9

File Index

9.1 File List

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

cpl_alibaba_oss.h . . . . . ??
cpl_atomic_ops.h . . . . . ??
cpl_aws.h . . . . . ??
cpl_azure.h . . . . . ??
cpl_config.h . . . . . ??
cpl_config_extras.h . . . . . ??
cpl_conv.h 175
cpl_cpu_features.h . . . . . ??
cpl_csv.h . . . . . ??
cpl_error.h 213
cpl_google_cloud.h . . . . . ??
cpl_hash_set.h 224
cpl_http.h 232
cpl_json.h 240
cpl_json_header.h . . . . . ??
cpl_json_streaming_parser.h . . . . . ??
cpl_list.h 240
cpl_md5.h . . . . . ??
cpl_mem_cache.h . . . . . ??
cpl_minixml.h 245
cpl_minizip_ioapi.h . . . . . ??
cpl_minizip_unzip.h . . . . . ??
cpl_minizip_zip.h . . . . . ??
cpl_multiproc.h . . . . . ??
cpl_odbc.h 256
cpl_port.h 256
cpl_progress.h . . . . . ??
cpl_quad_tree.h 269
cpl_sha1.h . . . . . ??
cpl_sha256.h . . . . . ??
cpl_spawn.h . . . . . ??
cpl_string.h 275
cpl_swift.h . . . . . ??
cpl_time.h . . . . . ??
cpl_virtualmem.h 301

cpl_vsi.h 312
cpl_vsi_error.h . . . . . **??**
cpl_vsi_virtual.h . . . . . **??**
cpl_vsil_curl_priv.h . . . . . **??**
cpl_worker_thread_pool.h 348
cplkeywordparser.h . . . . . **??**
gdal_csv.h . . . . . **??**
iso8211.h . . . . . **??**
sdts_al.h . . . . . **??**
sdtsdataset.cpp 349

Chapter 10

Class Documentation

10.1 _CPL_SHA256Context Struct Reference

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

- cpl_sha256.h

10.2 _CPLHashSet Struct Reference

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

- cpl_hash_set.cpp

10.3 _CPLList Struct Reference

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

Public Attributes

- void * pData
- struct _CPLList * psNext

10.3.1 Detailed Description

List element structure.

10.3.2 Member Data Documentation

10.3.2.1 pData

```
void* _CPLList::pData
```

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

10.3.2.2 psNext

```
struct _CPLList* _CPLList::psNext
```

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

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

- cpl_list.h

10.4 _CPLLock Struct Reference

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

- cpl_multiproc.cpp

10.5 _CPLQuadTree Struct Reference

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

- cpl_quad_tree.cpp

10.6 _CPLSpawnedProcess Struct Reference

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

- cpl_spawn.cpp

10.7 _CPLSpinLock Struct Reference

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

- cpl_multiproc.cpp

10.8 _MutexLinkedElt Struct Reference

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

- cpl_multiproc.cpp

10.9 _QuadTreeNode Struct Reference

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

- cpl_quad_tree.cpp

10.10 CPL_SHA1Context Struct Reference

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

- cpl_sha1.cpp

10.11 CPLErrorContext Struct Reference

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

- cpl_error.cpp

10.12 CPLHTTPResult Struct Reference

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

Public Attributes

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

10.12.1 Detailed Description

Describe the result of a CPLHTTPFetch() call

10.12.2 Member Data Documentation

10.12.2.1 nDataAlloc

```
int CPLHTTPResult::nDataAlloc
```

Allocated size of the pabyData buffer

10.12.2.2 nDataLen

```
int CPLHTTPResult::nDataLen
```

Length of the pabyData buffer

10.12.2.3 nMimePartCount

```
int CPLHTTPResult::nMimePartCount
```

Number of parts in a multipart message

10.12.2.4 nStatus

```
int CPLHTTPResult::nStatus
```

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

10.12.2.5 pabyData

```
GByte* CPLHTTPResult::pabyData
```

Buffer with downloaded data

10.12.2.6 papszHeaders

```
char** CPLHTTPResult::papszHeaders
```

Headers returned

10.12.2.7 pasMimePart

```
CPLMimePart* CPLHTTPResult::pasMimePart
```

Array of parts (resolved by CPLHTTPParseMultipartMime())

10.12.2.8 pszContentType

```
char* CPLHTTPResult::pszContentType
```

Content-Type of the response

10.12.2.9 pszErrBuf

```
char* CPLHTTPResult::pszErrBuf
```

Error message from curl, or NULL

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

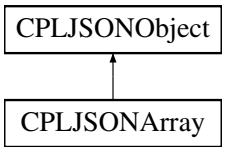
- cpl_http.h

10.13 CPLJSONArray Class Reference

The JSONArray class JSON array from JSONDocument.

```
#include <cpl_json.h>
```

Inheritance diagram for CPLJSONArray:



Public Member Functions

- int Size () const
- void Add (const CPLJSONObject &oValue)
- void Add (const std::string &osValue)
- void Add (const char *pszValue)
- void Add (double dfValue)
- void Add (int nValue)
- void Add (GInt64 nValue)
- void Add (bool bValue)
- CPLJSONObject operator[ ] (int nIndex)
- const CPLJSONObject operator[ ] (int nIndex) const

Friends

- class **CPLJSONObject**
- class **CPLJSONDocument**

Additional Inherited Members

10.13.1 Detailed Description

The JSONArray class JSON array from JSONDocument.

10.13.2 Member Function Documentation

10.13.2.1 Add() [1/7]

```
void CPLJSONArray::Add (
            const CPLJSONObject & oValue )
```

Add json object to array.

Parameters

oValue	Json array.

Since

GDAL 2.3

10.13.2.2 Add() [2/7]

```
void CPLJSONArray::Add (
            const std::string & osValue )
```

Add value to array

Parameters

osValue	Value to add.

Since

GDAL 2.3

10.13.2.3 Add() [3/7]

```
void CPLJSONArray::Add (
            const char * pszValue )
```

Add value to array

Parameters

pszValue	Value to add.

Since

GDAL 2.3

10.13.2.4 Add() [4/7]

```
void CPLJSONArray::Add (
            double dfValue )
```

Add value to array

Parameters

dfValue	Value to add.

Since

GDAL 2.3

10.13.2.5 Add() [5/7]

```
void CPLJSONArray::Add (
            int nValue )
```

Add value to array

Parameters

nValue	Value to add.

Since

GDAL 2.3

10.13.2.6 Add() [6/7]

```
void CPLJSONArray::Add (
            GInt64 nValue )
```

Add value to array

Parameters

nValue	Value to add.

Since

GDAL 2.3

10.13.2.7 Add() [7/7]

```
void CPLJSONArray::Add (
            bool bValue )
```

Add value to array

Parameters

bValue	Value to add.

Since

GDAL 2.3

10.13.2.8 operator[]() [1/2]

```
CPLJSONObject CPLJSONArray::operator[] (
            int nIndex )
```

Get array item by index.

Parameters

nIndex	Item index.

Returns

Json object.

Since

GDAL 2.3

10.13.2.9 operator[]() [2/2]

```
const CPLJSONObject CPLJSONArray::operator[] (
            int nIndex ) const
```

Get array const item by index.

Parameters

nIndex	Item index.

Returns

Json object.

Since

GDAL 2.3

10.13.2.10 Size()

```
int CPLJSONArray::Size ( ) const
```

Get array size.

Returns

Array size.

Since

GDAL 2.3

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

- cpl_json.h
- cpl_json.cpp

10.14 CPLJSONDocument Class Reference

The CPLJSONDocument class Wrapper class around json-c library.

```
#include <cpl_json.h>
```

Public Member Functions

- bool Save (const std::string &osPath)
- std::string SaveAsString ()
- CPLJSONObject GetRoot ()
- bool Load (const std::string &osPath)
- bool LoadMemory (const std::string &osStr)
- bool LoadMemory (const GByte ∗pabyData, int nLength=-1)
- bool LoadChunks (const std::string &osPath, size_t nChunkSize=16384, GDALProgressFunc pfn↩ Progress=nullptr, void ∗pProgressArg=nullptr)
- bool LoadUrl (const std::string &osUrl, char ∗∗papszOptions, GDALProgressFunc pfnProgress=nullptr, void ∗pProgressArg=nullptr)

10.14.1 Detailed Description

The CPLJSONDocument class Wrapper class around json-c library.

10.14.2 Member Function Documentation

10.14.2.1 GetRoot()

```
CPLJSONObject CPLJSONDocument::GetRoot ( )
```

Get json document root object

Returns

CPLJSONObject class instance

Since

GDAL 2.3

10.14.2.2 Load()

```
bool CPLJSONDocument::Load (
            const std::string & osPath )
```

Load json document from file by provided path

Parameters

osPath	Path to json file.

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.3 LoadChunks()

```
bool CPLJSONDocument::LoadChunks (
            const std::string & osPath,
            size_t nChunkSize = 16384,
            GDALProgressFunc pfnProgress = nullptr,
            void * pProgressArg = nullptr )
```

Load json document from file using small chunks of data.

Parameters

osPath	Path to json document file.
nChunkSize	Chunk size.
pfnProgress	a function to report progress of the json data loading.
pProgressArg	application data passed into progress function.

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.4 LoadMemory() [1/2]

```
bool CPLJSONDocument::LoadMemory (
            const std::string & osStr )
```

Load json document from memory buffer.

Parameters

osStr	String

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.5 LoadMemory() [2/2]

```
bool CPLJSONDocument::LoadMemory (
            const GByte * pabyData,
            int nLength = -1 )
```

Load json document from memory buffer.

Parameters

pabyData	Buffer.data.
nLength	Buffer size.

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.6 LoadUrl()

```
bool CPLJSONDocument::LoadUrl (
            const std::string & osUrl,
            char ** papszOptions,
            GDALProgressFunc pfnProgress = nullptr,
            void * pProgressArg = nullptr )
```

Load json document from web.

Parameters

osUrl	Url to json document.
papszOptions	Option list as a NULL-terminated array of strings. May be NULL. The available keys are same for CPLHTTPFetch method. Additional key JSON_DEPTH define json parse depth. Default is 10.
pfnProgress	a function to report progress of the json data loading.
pProgressArg	application data passed into progress function.

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.7 Save()

```
bool CPLJSONDocument::Save (
            const std::string & osPath )
```

Save json document at specified path

Parameters

osPath	Path to save json document

Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

Since

GDAL 2.3

10.14.2.8 SaveAsString()

```
std::string CPLJSONDocument::SaveAsString ( )
```

Return the json document as a serialized string.

Returns

serialized document.

Since

GDAL 2.3

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

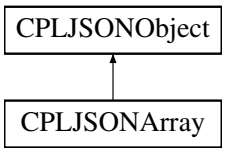
- cpl_json.h
- cpl_json.cpp

10.15 CPLJSONObject Class Reference

The CPLJSONArray class holds JSON object from CPLJSONDocument.

```
#include <cpl_json.h>
```

Inheritance diagram for CPLJSONObject:



Public Types

- enum Type
- enum PrettyFormat { Plain, Spaced, Pretty }

Public Member Functions

- void Add (const std::string &osName, const std::string &osValue)
- void Add (const std::string &osName, const char *pszValue)
- void Add (const std::string &osName, double dfValue)
- void Add (const std::string &osName, int nValue)
- void Add (const std::string &osName, GInt64 nValue)
- void Add (const std::string &osName, const CPLJSONArray &oValue)
- void Add (const std::string &osName, const CPLJSONObject &oValue)
- void Add (const std::string &osName, bool bValue)
- void AddNull (const std::string &osName)
- void Set (const std::string &osName, const std::string &osValue)
- void Set (const std::string &osName, const char *pszValue)
- void Set (const std::string &osName, double dfValue)
- void Set (const std::string &osName, int nValue)
- void Set (const std::string &osName, GInt64 nValue)
- void Set (const std::string &osName, bool bValue)
- void SetNull (const std::string &osName)

- std::string GetString (const std::string &osName, const std::string &osDefault="") const
- double GetDouble (const std::string &osName, double dfDefault=0.0) const
- int GetInteger (const std::string &osName, int nDefault=0) const
- GInt64 GetLong (const std::string &osName, GInt64 nDefault=0) const
- bool GetBool (const std::string &osName, bool bDefault=false) const
- std::string ToString (const std::string &osDefault="") const
- double ToDouble (double dfDefault=0.0) const
- int ToInteger (int nDefault=0) const
- GInt64 ToLong (GInt64 nDefault=0) const
- bool ToBool (bool bDefault=false) const
- CPLJSONArray ToArray () const
- std::string Format (enum PrettyFormat eFormat) const
- void Delete (const std::string &osName)
- CPLJSONArray GetArray (const std::string &osName) const
- CPLJSONObject GetObj (const std::string &osName) const
- CPLJSONObject operator[] (const std::string &osName) const
- enum Type GetType () const
- std::vector< CPLJSONObject > GetChildren () const

  Get json object children.
- bool IsValid () const
- void Deinit ()

Friends

- class **CPLJSONArray**
- class **CPLJSONDocument**

10.15.1 Detailed Description

The CPLJSONArray class holds JSON object from CPLJSONDocument.

10.15.2 Member Enumeration Documentation

10.15.2.1 PrettyFormat

```
enum CPLJSONObject::PrettyFormat
```

Json object format to string options

Enumerator

Plain	No extra whitespace or formatting applied.
Spaced	Minimal whitespace inserted.
Pretty	Formatted output.

10.15.2.2 Type

```
enum CPLJSONObject::Type
```

Json object types

10.15.3 Member Function Documentation

10.15.3.1 Add() [1/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            const std::string & osValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
osValue	String value.

Since

GDAL 2.3

10.15.3.2 Add() [2/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            const char * pszValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
pszValue	String value.

Since

GDAL 2.3

10.15.3.3 Add() [3/8]

```
CPL_C_END void CPLJSONObject::Add (
            const std::string & osName,
            double dfValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
dfValue	Double value.

Since

GDAL 2.3

10.15.3.4 Add() [4/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            int nValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
nValue	Integer value.

Since

GDAL 2.3

10.15.3.5 Add() [5/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            GInt64 nValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
nValue	Long value.

Since

GDAL 2.3

10.15.3.6 Add() [6/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            const CPLJSONArray & oValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
oValue	Array value.

Since

GDAL 2.3

10.15.3.7 Add() [7/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            const CPLJSONObject & oValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
oValue	Json object value.

Since

GDAL 2.3

10.15.3.8 Add() [8/8]

```
void CPLJSONObject::Add (
            const std::string & osName,
            bool bValue )
```

Add new key - value pair to json object.

Parameters

osName	Key name.
bValue	Boolean value.

Since

GDAL 2.3

10.15.3.9 AddNull()

```
void CPLJSONObject::AddNull (
            const std::string & osName )
```

Add new key - null pair to json object.

Parameters

osName	Key name.

Since

GDAL 2.3

10.15.3.10 Deinit()

```
void CPLJSONObject::Deinit ( )
```

Decrement reference counter and make pointer NULL. A json object will become invalid.

Since

GDAL 2.3

10.15.3.11 Delete()

```
void CPLJSONObject::Delete (
            const std::string & osName )
```

Delete json object by key.

Parameters

osName	Key name.

Since

GDAL 2.3

10.15.3.12 Format()

```
std::string CPLJSONObject::Format (
            enum PrettyFormat eFormat ) const
```

Stringify object to json format.

Parameters

eFormat	Format type,

Returns

A string in JSON format.

Since

GDAL 2.3

10.15.3.13 GetArray()

```
CPLJSONArray CPLJSONObject::GetArray (
            const std::string & osName ) const
```

Get value by key.

Parameters

osName	Key name.

Returns

Json array object.

Since

GDAL 2.3

10.15.3.14 GetBool()

```
bool CPLJSONObject::GetBool (
            const std::string & osName,
            bool bDefault = false ) const
```

Get value by key.

Parameters

osName	Key name.
bDefault	Default value.

Returns

Boolean value.

Since

GDAL 2.3

10.15.3.15 GetChildren()

```
std::vector< CPLJSONObject > CPLJSONObject::GetChildren ( ) const
```

Get json object children.

This function is useful when keys is not know and need to iterate over json object items and get keys and values.

Returns

Array of CPLJSONObject class instance.

Since

GDAL 2.3

10.15.3.16 GetDouble()

```
double CPLJSONObject::GetDouble (
            const std::string & osName,
            double dfDefault = 0.0 ) const
```

Get value by key.

Parameters

osName	Key name.
dfDefault	Default value.

Returns

Double value.

Since

GDAL 2.3

10.15.3.17 GetInteger()

```
int CPLJSONObject::GetInteger (
            const std::string & osName,
            int nDefault = 0 ) const
```

Get value by key.

Parameters

osName	Key name.
nDefault	Default value.

Returns

Integer value.

Since

GDAL 2.3

10.15.3.18 GetLong()

```
GInt64 CPLJSONObject::GetLong (
            const std::string & osName,
            GInt64 nDefault = 0 ) const
```

Get value by key.

Parameters

osName	Key name.
nDefault	Default value.

Returns

Long value.

Since

GDAL 2.3

10.15.3.19 GetObj()

```
CPLJSONObject CPLJSONObject::GetObj (
            const std::string & osName ) const
```

Get value by key.

Parameters

osName	Key name.

Returns

Json object.

Since

GDAL 2.3

10.15.3.20 GetString()

```
std::string CPLJSONObject::GetString (
            const std::string & osName,
            const std::string & osDefault = "" ) const
```

Get value by key.

Parameters

osName	Key name.
osDefault	Default value.

Returns

String value.

Since

GDAL 2.3

10.15.3.21 GetType()

```
CPLJSONObject::Type CPLJSONObject::GetType ( ) const
```

Get json object type.

Returns

Json object type.

Since

GDAL 2.3

10.15.3.22 IsValid()

```
bool CPLJSONObject::IsValid ( ) const
```

Check if json object valid.

Returns

true if json object valid.

Since

GDAL 2.3

10.15.3.23 operator[]()

```
CPLJSONObject CPLJSONObject::operator[] (
            const std::string & osName ) const
```

Get value by key.

Parameters

osName	Key name.

Returns

Json object.

Since

GDAL 2.3

10.15.3.24 Set() [1/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            const std::string & osValue )
```

Change value by key.

Parameters

osName	Key name.
osValue	String value.

Since

GDAL 2.3

10.15.3.25 Set() [2/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            const char * pszValue )
```

Change value by key.

Parameters

osName	Key name.
pszValue	String value.

Since

GDAL 2.3

10.15.3.26 Set() [3/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            double dfValue )
```

Change value by key.

Parameters

osName	Key name.
dfValue	Double value.

Since

GDAL 2.3

10.15.3.27 Set() [4/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            int nValue )
```

Change value by key.

Parameters

osName	Key name.
nValue	Integer value.

Since

GDAL 2.3

10.15.3.28 Set() [5/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            GInt64 nValue )
```

Change value by key.

Parameters

osName	Key name.
nValue	Long value.

Since

GDAL 2.3

10.15.3.29 Set() [6/6]

```
void CPLJSONObject::Set (
            const std::string & osName,
            bool bValue )
```

Change value by key.

Parameters

osName	Key name.
bValue	Boolean value.

Since

GDAL 2.3

10.15.3.30 SetNull()

```
void CPLJSONObject::SetNull (
            const std::string & osName )
```

Change value by key.

Parameters

osName	Key name.

Since

GDAL 2.3

10.15.3.31 ToArray()

```
CPLJSONArray CPLJSONObject::ToArray ( ) const
```

Get value.

Returns

Array

Since

GDAL 2.3

10.15.3.32 ToBool()

```
bool CPLJSONObject::ToBool (
            bool bDefault = false ) const
```

Get value.

Parameters

bDefault	Default value.

Returns

Boolean value.

Since

GDAL 2.3

10.15.3.33 ToDouble()

```
double CPLJSONObject::ToDouble (
            double dfDefault = 0.0 ) const
```

Get value

Parameters

dfDefault	Default value.

Returns

Double value.

Since

GDAL 2.3

10.15.3.34 ToInteger()

```
int CPLJSONObject::ToInteger (
            int nDefault = 0 ) const
```

Get value.

Parameters

nDefault	Default value.

Returns

Integer value.

Since

GDAL 2.3

10.15.3.35 ToLong()

```
GInt64 CPLJSONObject::ToLong (
            GInt64 nDefault = 0 ) const
```

Get value.

Parameters

nDefault	Default value.

Returns

Long value.

Since

GDAL 2.3

10.15.3.36 ToString()

```
std::string CPLJSONObject::ToString (
            const std::string & osDefault = "" ) const
```

Get value.

Parameters

osDefault	Default value.

Returns

String value.

Since

GDAL 2.3

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

- cpl_json.h
- cpl_json.cpp

10.16 CPLMD5Context Struct Reference

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

- cpl_md5.h

10.17 CPLMimePart Struct Reference

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

Public Attributes

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

10.17.1 Detailed Description

Describe a part of a multipart message

10.17.2 Member Data Documentation

10.17.2.1 nDataLen

```
int CPLMimePart::nDataLen
```

Buffer length

10.17.2.2 pabyData

```
GByte* CPLMimePart::pabyData
```

Buffer with data of the part

10.17.2.3 papszHeaders

```
char** CPLMimePart::papszHeaders
```

NULL terminated array of headers

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

- cpl_http.h

10.18 CPLODBCDriverInstaller Class Reference

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

Public Member Functions

- int InstallDriver (const char *pszDriver, const char *pszPathIn, WORD fRequest=ODBC_INSTALL_COMP↩LETE)
- int RemoveDriver (const char *pszDriverName, int fRemoveDSN=FALSE)
- int GetUsageCount () const
- const char * GetPathOut () const
- const char * GetLastError () const
- DWORD GetLastErrorCode () const

10.18.1 Detailed Description

A class providing functions to install or remove ODBC driver.

10.18.2 Member Function Documentation

10.18.2.1 GetLastError()

```
const char* CPLODBCDriverInstaller::GetLastError ( ) const  [inline]
```

If InstallDriver returns FALSE, then GetLastError then error message can be obtained by calling this function. Internally, it calls ODBC's SQLInstallerError function.

10.18.2.2 GetLastErrorCode()

```
DWORD CPLODBCDriverInstaller::GetLastErrorCode ( ) const  [inline]
```

If InstallDriver returns FALSE, then GetLastErrorCode then error code can be obtained by calling this function. Internally, it calls ODBC's SQLInstallerError function. See ODBC API Reference for possible error flags.

10.18.2.3 GetPathOut()

```
const char* CPLODBCDriverInstaller::GetPathOut ( ) const  [inline]
```

Path of the target directory where the driver should be installed. For details, see ODBC API Reference and lpsz↩ PathOut parameter of SQLInstallDriverEx

10.18.2.4 GetUsageCount()

```
int CPLODBCDriverInstaller::GetUsageCount ( ) const  [inline]
```

The usage count of the driver after this function has been called

10.18.2.5 InstallDriver()

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

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

Parameters

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

Returns

TRUE indicates success, FALSE if it fails.

10.18.2.6 RemoveDriver()

```
int CPLODBCDriverInstaller::RemoveDriver (
            const char * pszDriverName,
            int fRemoveDSN = FALSE )
```

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

Parameters

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

Returns

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

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

- cpl_odbc.h
- cpl_odbc.cpp

10.19 CPLODBCSession Class Reference

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

Public Member Functions

- CPLODBCSession ()
- ~CPLODBCSession ()
- int EstablishSession (const char *pszDSN, const char *pszUserid, const char *pszPassword)
- const char * GetLastError ()
- int ClearTransaction ()
- int BeginTransaction ()
- int CommitTransaction ()
- int RollbackTransaction ()
- int IsInTransaction ()
- int CloseSession ()
- int Failed (int, HSTMT=nullptr)
- HDBC GetConnection ()
- HENV GetEnvironment ()

10.19.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 CPLODBCSession()

```
CPLODBCSession::CPLODBCSession ( )
```

Constructor

10.19.2.2 ~CPLODBCSession()

```
CPLODBCSession::~CPLODBCSession ( )
```

Destructor

10.19.3 Member Function Documentation

10.19.3.1 BeginTransaction()

```
int CPLODBCSession::BeginTransaction ( )
```

Begin transaction

10.19.3.2 ClearTransaction()

```
int CPLODBCSession::ClearTransaction ( )
```

Clear transaction

10.19.3.3 CloseSession()

```
int CPLODBCSession::CloseSession ( )
```

Close session

10.19.3.4 CommitTransaction()

```
int CPLODBCSession::CommitTransaction ( )
```

Commit transaction

10.19.3.5 EstablishSession()

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

Connect to database and logon.

Parameters

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

Returns

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

10.19.3.6 Failed()

```
int CPLODBCSession::Failed (
            int nRetCode,
            HSTMT hStmt = nullptr )
```

Test if a return code indicates failure, return TRUE if that is the case. Also update error text.

10.19.3.7 GetConnection()

```
HDBC CPLODBCSession::GetConnection ( ) [inline]
```

Return connection handle

10.19.3.8 GetEnvironment()

```
HENV CPLODBCSession::GetEnvironment ( ) [inline]
```

Return GetEnvironment handle

10.19.3.9 GetLastError()

```
const char * CPLODBCSession::GetLastError ( )
```

Returns the last ODBC error message.

Returns

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

10.19.3.10 IsInTransaction()

```
int CPLODBCSession::IsInTransaction ( ) [inline]
```

Returns whether a transaction is active

10.19.3.11 RollbackTransaction()

```
int CPLODBCSession::RollbackTransaction ( )
```

Rollback transaction

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

- cpl_odbc.h
- cpl_odbc.cpp

10.20 CPLODBCStatement Class Reference

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

Public Member Functions

- CPLODBCStatement (CPLODBCSession ∗)
- ∼CPLODBCStatement ()
- HSTMT GetStatement ()
- void Clear ()
- void AppendEscaped (const char ∗)
- void Append (const char ∗)
- void Append (int)
- void Append (double)
- int Appendf (CPL_FORMAT_STRING(const char ∗),...) CPL_PRINT_FUNC_FORMAT(2
- int const char ∗ GetCommand ()
- int ExecuteSQL (const char ∗=nullptr)
- int Fetch (int nOrientation=SQL_FETCH_NEXT, int nOffset=0)
- void ClearColumnData ()
- int GetColCount ()
- const char ∗ GetColName (int)
- short GetColType (int)
- const char ∗ GetColTypeName (int)
- short GetColSize (int)
- short GetColPrecision (int)
- short GetColNullable (int)
- const char ∗ GetColColumnDef (int)
- int GetColId (const char ∗)
- const char ∗ GetColData (int, const char ∗=nullptr)
- const char ∗ GetColData (const char ∗, const char ∗=nullptr)
- int GetColDataLength (int)
- int GetRowCountAffected ()
- int GetColumns (const char ∗pszTable, const char ∗pszCatalog=nullptr, const char ∗pszSchema=nullptr)
- int GetPrimaryKeys (const char ∗pszTable, const char ∗pszCatalog=nullptr, const char ∗pszSchema=nullptr)
- int GetTables (const char ∗pszCatalog=nullptr, const char ∗pszSchema=nullptr)
- void DumpResult (FILE ∗fp, int bShowSchema=FALSE)
- int CollectResultsInfo ()

Static Public Member Functions

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

10.20.1 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.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 CPLODBCStatement()

```
CPLODBCStatement::CPLODBCStatement (
            CPLODBCSession * poSession )  [explicit]
```

Constructor

10.20.2.2 ~CPLODBCStatement()

```
CPLODBCStatement::~CPLODBCStatement ( )
```

Destructor

10.20.3 Member Function Documentation

10.20.3.1 Append() [1/3]

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

Append text to internal command.

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

Parameters

pszText	text to append.

10.20.3.2 Append() [2/3]

```
void CPLODBCStatement::Append (
            int nValue )
```

Append to internal command.

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

Parameters

nValue	value to append to the command.

10.20.3.3 Append() [3/3]

```
void CPLODBCStatement::Append (
            double dfValue )
```

Append to internal command.

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

Parameters

dfValue	value to append to the command.

10.20.3.4 AppendEscaped()

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

Append text to internal command.

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

Parameters

pszText	text to append.

10.20.3.5 Appendf()

```
int CPLODBCStatement::Appendf (
            CPL_FORMAT_STRING(const char *) ,
             ...  )
```

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() methods.

Parameters

pszFormat	printf() style format string.

Returns

FALSE if formatting fails due to result being too large.

10.20.3.6 Clear()

```
void CPLODBCStatement::Clear ( )
```

Clear internal command text and result set definitions.

10.20.3.7 ClearColumnData()

```
void CPLODBCStatement::ClearColumnData ( )
```

ClearColumnData

10.20.3.8 CollectResultsInfo()

```
int CPLODBCStatement::CollectResultsInfo ( )
```

CollectResultsInfo

10.20.3.9 DumpResult()

```
void CPLODBCStatement::DumpResult (
            FILE * fp,
            int bShowSchema = FALSE )
```

Dump resultset to file.

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

Parameters

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

10.20.3.10 ExecuteSQL()

```
int CPLODBCStatement::ExecuteSQL (
            const char * pszStatement = nullptr )
```

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() or Appendf() calls will be used.

Parameters

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

Returns

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

10.20.3.11 Fetch()

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

Fetch a new record.

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

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

Parameters

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

Returns

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

10.20.3.12 GetColColumnDef()

```
const char * CPLODBCStatement::GetColColumnDef (
            int iCol )
```

Fetch a column default value.

Returns the default value of a column.

Parameters

iCol	the zero based column index.

Returns

NULL if the default value is not dpecified or the internal copy of the default value.

10.20.3.13 GetColCount()

```
int CPLODBCStatement::GetColCount ( )
```

Fetch the resultset column count.

Returns

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

10.20.3.14 GetColData() [1/2]

```
const char * CPLODBCStatement::GetColData (
            int iCol,
            const char * pszDefault = nullptr )
```

Fetch column data.

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

Parameters

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

Returns

pointer to internal column data or NULL on failure.

10.20.3.15 GetColData() [2/2]

```
const char * CPLODBCStatement::GetColData (
            const char * pszColName,
            const char * pszDefault = nullptr )
```

Fetch column data.

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

Parameters

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

Returns

pointer to internal column data or NULL on failure.

10.20.3.16 GetColDataLength()

```
int CPLODBCStatement::GetColDataLength (
            int iCol )
```

GetColDataLength

10.20.3.17 GetColId()

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

Fetch column index.

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

Parameters

pszColName	the name to search for.

Returns

the column index, or -1 if not found.

10.20.3.18 GetColName()

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

Fetch a column name.

Parameters

iCol	the zero based column index.

Returns

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

10.20.3.19 GetColNullable()

```
short CPLODBCStatement::GetColNullable (
            int iCol )
```

Fetch the column nullability.

Parameters

iCol	the zero based column index.

Returns

TRUE if the column may contains or FALSE otherwise.

10.20.3.20 GetColPrecision()

```
short CPLODBCStatement::GetColPrecision (
            int iCol )
```

Fetch the column precision.

Parameters

iCol	the zero based column index.

Returns

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

10.20.3.21 GetColSize()

```
short CPLODBCStatement::GetColSize (
            int iCol )
```

Fetch the column width.

Parameters

iCol	the zero based column index.

Returns

column width, zero for unknown width columns.

10.20.3.22 GetColType()

```
short CPLODBCStatement::GetColType (
            int iCol )
```

Fetch a column data type.

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

Parameters

iCol	the zero based column index.

Returns

type code or -1 if the column is illegal.

10.20.3.23 GetColTypeName()

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

Fetch a column data type name.

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

Parameters

iCol	the zero based column index.

Returns

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

10.20.3.24 GetColumns()

```
int CPLODBCStatement::GetColumns (
            const char * pszTable,
            const char * pszCatalog = nullptr,
            const char * pszSchema = nullptr )
```

Fetch column definitions for a table.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

10.20.3.25 GetCommand()

```
int const char* CPLODBCStatement::GetCommand ( )  [inline]
```

Return statement string

10.20.3.26 GetPrimaryKeys()

```
int CPLODBCStatement::GetPrimaryKeys (
            const char * pszTable,
            const char * pszCatalog = nullptr,
            const char * pszSchema = nullptr )
```

Fetch primary keys for a table.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

10.20.3.27 GetRowCountAffected()

```
int CPLODBCStatement::GetRowCountAffected ( )
```

GetRowCountAffected

10.20.3.28 GetStatement()

```
HSTMT CPLODBCStatement::GetStatement ( )  [inline]
```

Return statement handle

10.20.3.29 GetTables()

```
int CPLODBCStatement::GetTables (
            const char * pszCatalog = nullptr,
            const char * pszSchema = nullptr )
```

Fetch tables in database.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

10.20.3.30 GetTypeMapping()

```
SQLSMALLINT CPLODBCStatement::GetTypeMapping (
            SQLSMALLINT nTypeCode )  [static]
```

Get appropriate C data type for SQL column type.

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

Parameters

nTypeCode	the SQL_ code, such as SQL_CHAR.

Returns

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

10.20.3.31 GetTypeName()

```
CPLString CPLODBCStatement::GetTypeName (
            int nTypeCode )  [static]
```

Get name for SQL column type.

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

Parameters

nTypeCode	the SQL_ code, such as SQL_CHAR.

Returns

internal string, "UNKNOWN" if code not recognised.

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

- cpl_odbc.h
- cpl_odbc.cpp

10.21 CPLRectObj Struct Reference

```
#include <cpl_quad_tree.h>
```

Public Attributes

- double minx
- double miny
- double maxx
- double maxy

10.21.1 Detailed Description

Describe a rectangle

10.21.2 Member Data Documentation

10.21.2.1 maxx

```
double CPLRectObj::maxx
```

Maximum x

10.21.2.2 maxy

```
double CPLRectObj::maxy
```

Maximum y

10.21.2.3 minx

```
double CPLRectObj::minx
```

Minimum x

10.21.2.4 miny

```
double CPLRectObj::miny
```

Minimum y

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

- cpl_quad_tree.h

10.22 CPLSharedFileInfo Struct Reference

```
#include <cpl_conv.h>
```

Public Attributes

- FILE * fp
- int nRefCount
- int bLarge
- char * pszFilename
- char * pszAccess

10.22.1 Detailed Description

Information on a shared file

10.22.2 Member Data Documentation

10.22.2.1 bLarge

```
int CPLSharedFileInfo::bLarge
```

Whether fp must be interpreted as VSIFILE*

10.22.2.2 fp

```
FILE* CPLSharedFileInfo::fp
```

File pointer

10.22.2.3 nRefCount

```
int CPLSharedFileInfo::nRefCount
```

Reference counter

10.22.2.4 pszAccess

```
char* CPLSharedFileInfo::pszAccess
```

Access mode

10.22.2.5 pszFilename

```
char* CPLSharedFileInfo::pszFilename
```

Filename

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

- cpl_conv.h

10.23 CPLSharedFileInfoExtra Struct Reference

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

- cpl_conv.cpp

10.24 CPLStdCallThreadInfo Struct Reference

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

- cpl_multiproc.cpp

10.25 CPLVirtualMem Struct Reference

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

- cpl_virtualmem.cpp

10.26 CPLWorkerThread Struct Reference

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

- cpl_worker_thread_pool.h

10.27 CPLWorkerThreadJob Struct Reference

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

- cpl_worker_thread_pool.h

10.28 CPLWorkerThreadPool Class Reference

```
#include <cpl_worker_thread_pool.h>
```

Public Member Functions

- CPLWorkerThreadPool ()
- ~CPLWorkerThreadPool ()
- bool Setup (int nThreads, CPLThreadFunc pfnInitFunc, void **pasInitData)
- bool SubmitJob (CPLThreadFunc pfnFunc, void *pData)
- bool SubmitJobs (CPLThreadFunc pfnFunc, const std::vector< void *> &apData)
- void WaitCompletion (int nMaxRemainingJobs=0)
- int GetThreadCount () const

10.28.1 Detailed Description

Pool of worker threads

10.28.2 Constructor & Destructor Documentation

10.28.2.1 CPLWorkerThreadPool()

```
CPLWorkerThreadPool::CPLWorkerThreadPool ( )
```

Instantiate a new pool of worker threads.

The pool is in an uninitialized state after this call. The Setup() method must be called.

10.28.2.2 ~CPLWorkerThreadPool()

```
CPLWorkerThreadPool::~CPLWorkerThreadPool ( )
```

Destroys a pool of worker threads.

Any still pending job will be completed before the destructor returns.

10.28.3 Member Function Documentation

10.28.3.1 GetThreadCount()

```
int CPLWorkerThreadPool::GetThreadCount ( ) const  [inline]
```

Return the number of threads setup

10.28.3.2 Setup()

```
bool CPLWorkerThreadPool::Setup (
            int nThreads,
            CPLThreadFunc pfnInitFunc,
            void ** pasInitData )
```

Setup the pool.

Parameters

nThreads	Number of threads to launch
pfnInitFunc	Initialization function to run in each thread. May be NULL
pasInitData	Array of initialization data. Its length must be nThreads, or it should be NULL.

Returns

true if initialization was successful.

10.28.3.3 SubmitJob()

```
bool CPLWorkerThreadPool::SubmitJob (
            CPLThreadFunc pfnFunc,
            void * pData )
```

Queue a new job.

Parameters

pfnFunc	Function to run for the job.
pData	User data to pass to the job function.

Returns

true in case of success.

10.28.3.4 SubmitJobs()

```
bool CPLWorkerThreadPool::SubmitJobs (
            CPLThreadFunc pfnFunc,
            const std::vector< void *> & apData )
```

Queue several jobs

Parameters

pfnFunc	Function to run for the job.
apData	User data instances to pass to the job function.

Returns

true in case of success.

10.28.3.5 WaitCompletion()

```
void CPLWorkerThreadPool::WaitCompletion (
            int nMaxRemainingJobs = 0 )
```

Wait for completion of part or whole jobs.

Parameters

nMaxRemainingJobs	Maximum number of pendings jobs that are allowed in the queue after this method has completed. Might be 0 to wait for all jobs.

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

- cpl_worker_thread_pool.h
- cpl_worker_thread_pool.cpp

10.29 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.

10.29.1 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(). 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 hierarchical tree structure for a document can be represented as a tree of CPLXMLNode structures.

10.29.2 Member Data Documentation

10.29.2.1 eType

```
CPLXMLNodeType CPLXMLNode::eType
```

Node type.

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

10.29.2.2 psChild

```
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.

10.29.2.3 psNext

```
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.

10.29.2.4 pszValue

```
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 "-->").

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

- cpl_minixml.h

10.30 CPLZip Struct Reference

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

- cpl_minizip_zip.cpp

10.31 ctb Struct Reference

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

- cpl_csv.cpp

10.32 curfile_info Struct Reference

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

- cpl_minizip_zip.cpp

10.33 DDFField Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- void Dump (FILE *fp)
- const char * GetSubfieldData (DDFSubfieldDefn *, int *=nullptr, int=0)
- const char * GetInstanceData (int nInstance, int *pnSize)
- const char * GetData () const
- int GetDataSize () const
- int GetRepeatCount ()
- DDFFieldDefn * GetFieldDefn ()

10.33.1 Detailed Description

This object represents one field in a DDFRecord. This models an instance of the fields data, rather than its data definition, which is handled by the DDFFieldDefn class. Note that a DDFField doesn't have DDFSubfield children as you would expect. To extract subfield values use GetSubfieldData() to find the right data pointer and then use ExtractIntData(), ExtractFloatData() or ExtractStringData().

10.33.2 Member Function Documentation

10.33.2.1 Dump()

```
void DDFField::Dump (
            FILE * fp )
```

Write out field contents to debugging file.

A variety of information about this field, and all its subfields is written to the given debugging file handle. Note that field definition information (ala DDFFieldDefn) isn't written.

Parameters

fp	The standard IO file handle to write to. i.e. stderr

10.33.2.2 GetData()

```
const char* DDFField::GetData ( ) const  [inline]
```

Return the pointer to the entire data block for this record. This is an internal copy, and should not be freed by the application.

10.33.2.3 GetDataSize()

```
int DDFField::GetDataSize ( ) const  [inline]
```

Return the number of bytes in the data block returned by GetData().

10.33.2.4 GetFieldDefn()

```
DDFFieldDefn* DDFField::GetFieldDefn ( )  [inline]
```

Fetch the corresponding DDFFieldDefn.

10.33.2.5 GetInstanceData()

```
const char * DDFField::GetInstanceData (
            int nInstance,
            int * pnInstanceSize )
```

Get field instance data and size.

The returned data pointer and size values are suitable for use with DDFRecord::SetFieldRaw().

Parameters

nInstance	a value from 0 to GetRepeatCount()-1.
pnInstanceSize	a location to put the size (in bytes) of the field instance data returned. This size will include the unit terminator (if any), but not the field terminator. This size pointer may be NULL if not needed.

Returns

the data pointer, or NULL on error.

10.33.2.6 GetRepeatCount()

```
int DDFField::GetRepeatCount ( )
```

How many times do the subfields of this record repeat? This will always be one for non-repeating fields.

Returns

The number of times that the subfields of this record occur in this record. This will be one for non-repeating fields.

See also

`8211view example program` for a demonstration of handling repeated fields properly.

10.33.2.7 GetSubfieldData()

```
const char * DDFField::GetSubfieldData (
            DDFSubfieldDefn * poSFDefn,
            int * pnMaxBytes = nullptr,
            int iSubfieldIndex = 0 )
```

Fetch raw data pointer for a particular subfield of this field.

The passed DDFSubfieldDefn (poSFDefn) should be acquired from the DDFFieldDefn corresponding with this field. This is normally done once before reading any records. This method involves a series of calls to DDFSubfield::↩ GetDataLength() in order to track through the DDFField data to that belonging to the requested subfield. This can be relatively expensive.

Parameters

poSFDefn	The definition of the subfield for which the raw data pointer is desired.
pnMaxBytes	The maximum number of bytes that can be accessed from the returned data pointer is placed in this int, unless it is NULL.
iSubfieldIndex	The instance of this subfield to fetch. Use zero (the default) for the first instance.

Returns

A pointer into the DDFField's data that belongs to the subfield. This returned pointer is invalidated by the next record read (DDFRecord::ReadRecord()) and the returned pointer should not be freed by the application.

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

- iso8211.h
- ddffield.cpp

10.34 DDFFieldDefn Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- void Dump (FILE *fp)
- const char * GetName () const
- const char * GetDescription () const
- int GetSubfieldCount () const
- DDFSubfieldDefn * GetSubfield (int i)
- DDFSubfieldDefn * FindSubfieldDefn (const char *)
- int GetFixedWidth () const
- int IsRepeating () const
- void SetRepeatingFlag (int n)
- char * GetDefaultValue (int *pnSize)

10.34.1 Detailed Description

Information from the DDR defining one field. Note that just because a field is defined for a DDFModule doesn't mean that it actually occurs on any records in the module. DDFFieldDefns are normally just significant as containers of the DDFSubfieldDefns.

10.34.2 Member Function Documentation

10.34.2.1 Dump()

```
void DDFFieldDefn::Dump (
            FILE * fp )
```

Write out field definition info to debugging file.

A variety of information about this field definition, and all its subfields is written to the give debugging file handle.

Parameters

fp	The standard IO file handle to write to. i.e. stderr

10.34.2.2 FindSubfieldDefn()

```
DDFSubfieldDefn * DDFFieldDefn::FindSubfieldDefn (
            const char * pszMnemonic )
```

Find a subfield definition by its mnemonic tag.

Parameters

pszMnemonic	The name of the field.

Returns

The subfield pointer, or NULL if there isn't any such subfield.

10.34.2.3 GetDefaultValue()

```
char * DDFFieldDefn::GetDefaultValue (
            int * pnSize )
```

Return default data for field instance.

10.34.2.4 GetDescription()

```
const char* DDFFieldDefn::GetDescription ( ) const  [inline]
```

Fetch a longer description of this field.

Returns

this is an internal copy and should not be freed.

10.34.2.5 GetFixedWidth()

```
int DDFFieldDefn::GetFixedWidth ( ) const  [inline]
```

Get the width of this field. This function isn't normally used by applications.

Returns

The width of the field in bytes, or zero if the field is not apparently of a fixed width.

10.34.2.6 GetName()

```
const char* DDFFieldDefn::GetName ( ) const  [inline]
```

Fetch a pointer to the field name (tag).

Returns

this is an internal copy and should not be freed.

10.34.2.7 GetSubfield()

```
DDFSubfieldDefn * DDFFieldDefn::GetSubfield (
            int i )
```

Fetch a subfield by index.

Parameters

i	The index subfield index. (Between 0 and GetSubfieldCount()-1)

Returns

The subfield pointer, or NULL if the index is out of range.

10.34.2.8 GetSubfieldCount()

```
int DDFFieldDefn::GetSubfieldCount ( ) const  [inline]
```

Get the number of subfields.

10.34.2.9 IsRepeating()

```
int DDFFieldDefn::IsRepeating ( ) const  [inline]
```

Fetch repeating flag.

See also

DDFField::GetRepeatCount()

Returns

TRUE if the field is marked as repeating.

10.34.2.10 SetRepeatingFlag()

```
void DDFFieldDefn::SetRepeatingFlag (
            int n )  [inline]
```

this is just for an S-57 hack for swedish data

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

- iso8211.h
- ddffielddefn.cpp

10.35 DDFModule Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- DDFModule ()
- ~DDFModule ()
- int Open (const char *pszFilename, int bFailQuietly=FALSE)
- void Close ()
- void Dump (FILE *fp)
- DDFRecord * ReadRecord ()
- void Rewind (long nOffset=-1)
- DDFFieldDefn * FindFieldDefn (const char *)
- int GetFieldCount () const
- DDFFieldDefn * GetField (int)
- void AddField (DDFFieldDefn *poNewFDefn)

10.35.1 Detailed Description

The primary class for reading ISO 8211 files. This class contains all the information read from the DDR record, and is used to read records from the file.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 DDFModule()

```
DDFModule::DDFModule ( )
```

The constructor.

10.35.2.2 ~DDFModule()

```
DDFModule::~DDFModule ( )
```

The destructor.

10.35.3 Member Function Documentation

10.35.3.1 AddField()

```
void DDFModule::AddField (
            DDFFieldDefn * poNewFDefn )
```

Add new field definition.

Field definitions may only be added to DDFModules being used for writing, not those being used for reading. Ownership of the DDFFieldDefn object is taken by the DDFModule.

Parameters

poNewFDefn	definition to be added to the module.

10.35.3.2 Close()

```
void DDFModule::Close ( )
```

Close an ISO 8211 file.

10.35.3.3 Dump()

```
void DDFModule::Dump (
            FILE * fp )
```

Write out module info to debugging file.

A variety of information about the module is written to the debugging file. This includes all the field and subfield definitions read from the header.

Parameters

fp	The standard IO file handle to write to. i.e. stderr.

10.35.3.4 FindFieldDefn()

```
DDFFieldDefn * DDFModule::FindFieldDefn (
            const char * pszFieldName )
```

Fetch the definition of the named field.

This function will scan the DDFFieldDefn's on this module, to find one with the indicated field name.

Parameters

pszFieldName	The name of the field to search for. The comparison is case insensitive.

Returns

A pointer to the request DDFFieldDefn object is returned, or NULL if none matching the name are found. The return object remains owned by the DDFModule, and should not be deleted by application code.

10.35.3.5 GetField()

```
DDFFieldDefn * DDFModule::GetField (
            int i )
```

Fetch a field definition by index.

Parameters

i	(from 0 to GetFieldCount() - 1.

Returns

the returned field pointer or NULL if the index is out of range.

10.35.3.6 GetFieldCount()

```
int DDFModule::GetFieldCount ( ) const  [inline]
```

Fetch the number of defined fields.

10.35.3.7 Open()

```
int DDFModule::Open (
            const char * pszFilename,
            int bFailQuietly = FALSE )
```

Open a ISO 8211 (DDF) file for reading.

If the open succeeds the data descriptive record (DDR) will have been read, and all the field and subfield definitions will be available.

Parameters

pszFilename	The name of the file to open.
bFailQuietly	If FALSE a CPL Error is issued for non-8211 files, otherwise quietly return NULL.

Returns

FALSE if the open fails or TRUE if it succeeds. Errors messages are issued internally with CPLError().

10.35.3.8 ReadRecord()

```
DDFRecord * DDFModule::ReadRecord ( )
```

Read one record from the file.

Returns

A pointer to a DDFRecord object is returned, or NULL if a read error, or end of file occurs. The returned record is owned by the module, and should not be deleted by the application. The record is only valid until the next ReadRecord() at which point it is overwritten.

10.35.3.9 Rewind()

```
void DDFModule::Rewind (
            long nOffset = -1 )
```

Return to first record.

The next call to ReadRecord() will read the first data record in the file.

Parameters

nOffset	the offset in the file to return to. By default this is -1, a special value indicating that reading should return to the first data record. Otherwise it is an absolute byte offset in the file.

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

- iso8211.h
- ddfmodule.cpp

10.36 DDFRecord Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- DDFRecord * Clone ()
- DDFRecord * CloneOn (DDFModule *)
- void Dump (FILE *)
- int GetFieldCount () const
- DDFField * FindField (const char *, int=0)
- DDFField * GetField (int)
- int GetIntSubfield (const char *, int, const char *, int, int *=nullptr)
- double GetFloatSubfield (const char *, int, const char *, int, int *=nullptr)
- const char * GetStringSubfield (const char *, int, const char *, int, int *=nullptr)
- int SetIntSubfield (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, int n↩ Value)
- int SetStringSubfield (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, const char *pszValue, int nValueLength=-1)
- int SetFloatSubfield (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, double dfNewValue)
- int GetDataSize () const

- const char * GetData () const
- DDFModule * GetModule ()
- int ResizeField (DDFField *poField, int nNewDataSize)
- int DeleteField (DDFField *poField)
- DDFField * AddField (DDFFieldDefn *)
- int CreateDefaultFieldInstance (DDFField *poField, int iIndexWithinField)
- int SetFieldRaw (DDFField *poField, int iIndexWithinField, const char *pachRawData, int nRawDataSize)
- int Write ()

10.36.1 Detailed Description

Contains instance data from one data record (DR). The data is contained as a list of DDFField instances partitioning the raw data into fields.

10.36.2 Member Function Documentation

10.36.2.1 AddField()

```
DDFField * DDFRecord::AddField (
            DDFFieldDefn * poDefn )
```

Add a new field to record.

Add a new zero sized field to the record. The new field is always added at the end of the record.

NOTE: This method doesn't currently update the header information for the record to include the field information for this field, so the resulting record image isn't suitable for writing to disk. However, everything else about the record state should be updated properly to reflect the new field.

Parameters

poDefn	the definition of the field to be added.

Returns

the field object on success, or NULL on failure.

10.36.2.2 Clone()

```
DDFRecord * DDFRecord::Clone ( )
```

Make a copy of a record.

This method is used to make a copy of a record that will become (mostly) the properly of application. However, it is automatically destroyed if the DDFModule it was created relative to is destroyed, as its field and subfield definitions relate to that DDFModule. However, it does persist even when the record returned by DDFModule::ReadRecord() is invalidated, such as when reading a new record. This allows an application to cache whole DDFRecords.

Returns

A new copy of the DDFRecord. This can be delete'd by the application when no longer needed, otherwise it will be cleaned up when the DDFModule it relates to is destroyed or closed.

10.36.2.3 CloneOn()

```
DDFRecord * DDFRecord::CloneOn (
            DDFModule * poTargetModule )
```

Recreate a record referencing another module.

Works similarly to the DDFRecord::Clone() method, but creates the new record with reference to a different DDFModule. All DDFFieldDefn references are transcribed onto the new module based on field names. If any fields don't have a similarly named field on the target module the operation will fail. No validation of field types and properties is done, but this operation is intended only to be used between modules with matching definitions of all affected fields.

The new record will be managed as a clone by the target module in a manner similar to regular clones.

Parameters

poTargetModule	the module on which the record copy should be created.

Returns

NULL on failure or a pointer to the cloned record.

10.36.2.4 CreateDefaultFieldInstance()

```
int DDFRecord::CreateDefaultFieldInstance (
            DDFField * poField,
            int iIndexWithinField )
```

Initialize default instance.

This method is normally only used internally by the AddField() method to initialize the new field instance with default subfield values. It installs default data for one instance of the field in the record using the DDFFieldDefn::GetDefaultValue() method and DDFRecord::SetFieldRaw().

Parameters

poField	the field within the record to be assign a default instance.
iIndexWithinField	the instance to set (may not have been tested with values other than 0).

Returns

TRUE on success or FALSE on failure.

10.36.2.5 DeleteField()

```
int DDFRecord::DeleteField (
            DDFField * poTarget )
```

Delete a field instance from a record.

Remove a field from this record, cleaning up the data portion and repacking the fields list. We don't try to reallocate the data area of the record to be smaller.

NOTE: This method doesn't actually remove the header information for this field from the record tag list yet. This should be added if the resulting record is even to be written back to disk!

Parameters

poTarget	the field instance on this record to delete.

Returns

TRUE on success, or FALSE on failure. Failure can occur if poTarget isn't really a field on this record.

10.36.2.6 Dump()

```
void DDFRecord::Dump (
            FILE * fp )
```

Write out record contents to debugging file.

A variety of information about this record, and all its fields and subfields is written to the given debugging file handle. Note that field definition information (ala DDFFieldDefn) isn't written.

Parameters

fp	The standard IO file handle to write to. i.e. stderr

10.36.2.7 FindField()

```
DDFField * DDFRecord::FindField (
            const char * pszName,
            int iFieldIndex = 0 )
```

Find the named field within this record.

Parameters

pszName	The name of the field to fetch. The comparison is case insensitive.
iFieldIndex	The instance of this field to fetch. Use zero (the default) for the first instance.

Returns

Pointer to the requested DDFField. This pointer is to an internal object, and should not be freed. It remains valid until the next record read.

10.36.2.8 GetData()

```
const char* DDFRecord::GetData ( ) const  [inline]
```

Fetch the raw data for this record. The returned pointer is effectively to the data for the first field of the record, and is of size GetDataSize().

10.36.2.9 GetDataSize()

```
int DDFRecord::GetDataSize ( ) const  [inline]
```

Fetch size of records raw data (GetData()) in bytes.

10.36.2.10 GetField()

```
DDFField * DDFRecord::GetField (
            int i )
```

Fetch field object based on index.

Parameters

i	The index of the field to fetch. Between 0 and GetFieldCount()-1.

Returns

A DDFField pointer, or NULL if the index is out of range.

10.36.2.11 GetFieldCount()

```
int DDFRecord::GetFieldCount ( ) const  [inline]
```

Get the number of DDFFields on this record.

10.36.2.12 GetFloatSubfield()

```
double DDFRecord::GetFloatSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            int * pnSuccess = nullptr )
```

Fetch value of a subfield as a float (double). This is a convenience function for fetching a subfield of a field within this record.

Parameters

pszField	The name of the field containing the subfield.
iFieldIndex	The instance of this field within the record. Use zero for the first instance of this field.
pszSubfield	The name of the subfield within the selected field.
iSubfieldIndex	The instance of this subfield within the record. Use zero for the first instance.
pnSuccess	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or zero if it failed for some reason.

10.36.2.13 GetIntSubfield()

```
int DDFRecord::GetIntSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            int * pnSuccess = nullptr )
```

Fetch value of a subfield as an integer. This is a convenience function for fetching a subfield of a field within this record.

Parameters

pszField	The name of the field containing the subfield.
iFieldIndex	The instance of this field within the record. Use zero for the first instance of this field.
pszSubfield	The name of the subfield within the selected field.
iSubfieldIndex	The instance of this subfield within the record. Use zero for the first instance.
pnSuccess	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or zero if it failed for some reason.

10.36.2.14 GetModule()

```
DDFModule* DDFRecord::GetModule ( ) [inline]
```

Fetch the DDFModule with which this record is associated.

10.36.2.15 GetStringSubfield()

```
const char * DDFRecord::GetStringSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            int * pnSuccess = nullptr )
```

Fetch value of a subfield as a string. This is a convenience function for fetching a subfield of a field within this record.

Parameters

pszField	The name of the field containing the subfield.
iFieldIndex	The instance of this field within the record. Use zero for the first instance of this field.
pszSubfield	The name of the subfield within the selected field.
iSubfieldIndex	The instance of this subfield within the record. Use zero for the first instance.
pnSuccess	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or NULL if it failed for some reason. The returned pointer is to internal data and should not be modified or freed by the application.

10.36.2.16 ResizeField()

```
int DDFRecord::ResizeField (
            DDFField * poField,
            int nNewDataSize )
```

Alter field data size within record.

This method will rearrange a DDFRecord altering the amount of space reserved for one of the existing fields. All following fields will be shifted accordingly. This includes updating the DDFField infos, and actually moving stuff within the data array after reallocating to the desired size.

Parameters

poField	the field to alter.
nNewDataSize	the number of data bytes to be reserved for the field.

Returns

TRUE on success or FALSE on failure.

10.36.2.17 SetFieldRaw()

```
int DDFRecord::SetFieldRaw (
            DDFField * poField,
            int iIndexWithinField,
            const char * pachRawData,
            int nRawDataSize )
```

Set the raw contents of a field instance.

Parameters

poField	the field to set data within.
iIndexWithinField	The instance of this field to replace. Must be a value between 0 and GetRepeatCount(). If GetRepeatCount() is used, a new instance of the field is appended.
pachRawData	the raw data to replace this field instance with.
nRawDataSize	the number of bytes pointed to by pachRawData.

Returns

TRUE on success or FALSE on failure.

10.36.2.18 SetFloatSubfield()

```
int DDFRecord::SetFloatSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            double dfNewValue )
```

Set a float subfield in record.

The value of a given subfield is replaced with a new float value formatted appropriately.

Parameters

pszField	the field name to operate on.
iFieldIndex	the field index to operate on (zero based).
pszSubfield	the subfield name to operate on.
iSubfieldIndex	the subfield index to operate on (zero based).
dfNewValue	the new value to place in the subfield.

Returns

TRUE if successful, and FALSE if not.

10.36.2.19 SetIntSubfield()

```
int DDFRecord::SetIntSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            int nNewValue )
```

Set an integer subfield in record.

The value of a given subfield is replaced with a new integer value formatted appropriately.

Parameters

pszField	the field name to operate on.
iFieldIndex	the field index to operate on (zero based).
pszSubfield	the subfield name to operate on.
iSubfieldIndex	the subfield index to operate on (zero based).
nNewValue	the new value to place in the subfield.

Returns

TRUE if successful, and FALSE if not.

10.36.2.20 SetStringSubfield()

```
int DDFRecord::SetStringSubfield (
            const char * pszField,
            int iFieldIndex,
            const char * pszSubfield,
            int iSubfieldIndex,
            const char * pszValue,
            int nValueLength = -1 )
```

Set a string subfield in record.

The value of a given subfield is replaced with a new string value formatted appropriately.

Parameters

pszField	the field name to operate on.
iFieldIndex	the field index to operate on (zero based).
pszSubfield	the subfield name to operate on.
iSubfieldIndex	the subfield index to operate on (zero based).
pszValue	the new string to place in the subfield. This may be arbitrary binary bytes if nValueLength is specified.
nValueLength	the number of valid bytes in pszValue, may be -1 to internally fetch with strlen().

Returns

TRUE if successful, and FALSE if not.

10.36.2.21 Write()

```
int DDFRecord::Write ( )
```

Write record out to module.

This method writes the current record to the module to which it is attached. Normally this would be at the end of the file, and only used for modules newly created with DDFModule::Create(). Rewriting existing records is not supported at this time. Calling Write() multiple times on a DDFRecord will result it multiple copies being written at the end of the module.

Returns

TRUE on success or FALSE on failure.

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

- iso8211.h
- ddfrecord.cpp

10.37 DDFSubfieldDefn Class Reference

```
#include <iso8211.h>
```

Public Types

- enum DDFBinaryFormat

Public Member Functions

- const char * GetName () const
- const char * GetFormat () const
- DDFDataType GetType () const
- double ExtractFloatData (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- int ExtractIntData (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- const char * ExtractStringData (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- int GetDataLength (const char *, int, int *)
- void DumpData (const char *pachData, int nMaxBytes, FILE *fp)
- int FormatStringValue (char *pachData, int nBytesAvailable, int *pnBytesUsed, const char *pszValue, int nValueLength=-1) const
- int FormatIntValue (char *pachData, int nBytesAvailable, int *pnBytesUsed, int nNewValue) const
- int FormatFloatValue (char *pachData, int nBytesAvailable, int *pnBytesUsed, double dfNewValue) const
- int GetWidth () const
- int GetDefaultValue (char *pachData, int nBytesAvailable, int *pnBytesUsed) const
- void Dump (FILE *fp)

10.37.1 Detailed Description

Information from the DDR record describing one subfield of a DDFFieldDefn. All subfields of a field will occur in each occurrence of that field (as a DDFField) in a DDFRecord. Subfield's actually contain formatted data (as instances within a record).

10.37.2 Member Enumeration Documentation

10.37.2.1 DDFBinaryFormat

```
enum DDFSubfieldDefn::DDFBinaryFormat
```

Binary format: this is the digit immediately following the B or b for binary formats.

10.37.3 Member Function Documentation

10.37.3.1 Dump()

```
void DDFSubfieldDefn::Dump (
            FILE * fp )
```

Write out subfield definition info to debugging file.

A variety of information about this field definition is written to the give debugging file handle.

Parameters

fp	The standard IO file handle to write to. i.e. stderr

10.37.3.2 DumpData()

```
void DDFSubfieldDefn::DumpData (
            const char * pachData,
            int nMaxBytes,
            FILE * fp )
```

Dump subfield value to debugging file.

Parameters

pachData	Pointer to data for this subfield.
nMaxBytes	Maximum number of bytes available in pachData.
fp	File to write report to.

10.37.3.3 ExtractFloatData()

```
double DDFSubfieldDefn::ExtractFloatData (
            const char * pachSourceData,
            int nMaxBytes,
            int * pnConsumedBytes )
```

Extract a subfield value as a float. Given a pointer to the data for this subfield (from within a DDFRecord) this method will return the floating point data for this subfield. The number of bytes consumed as part of this field can also be fetched. This method may be called for any type of subfield, and will return zero if the subfield is not numeric.

Parameters

pachSourceData	The pointer to the raw data for this field. This may have come from DDFRecord::GetData(), taking into account skip factors over previous subfields data.
nMaxBytes	The maximum number of bytes that are accessible after pachSourceData.
pnConsumedBytes	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

The subfield's numeric value (or zero if it isn't numeric).

See also

ExtractIntData(), ExtractStringData()

10.37.3.4 ExtractIntData()

```
int DDFSubfieldDefn::ExtractIntData (
            const char * pachSourceData,
            int nMaxBytes,
            int * pnConsumedBytes )
```

Extract a subfield value as an integer. Given a pointer to the data for this subfield (from within a DDFRecord) this method will return the int data for this subfield. The number of bytes consumed as part of this field can also be fetched. This method may be called for any type of subfield, and will return zero if the subfield is not numeric.

Parameters

pachSourceData	The pointer to the raw data for this field. This may have come from DDFRecord::GetData(), taking into account skip factors over previous subfields data.
nMaxBytes	The maximum number of bytes that are accessible after pachSourceData.
pnConsumedBytes	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

The subfield's numeric value (or zero if it isn't numeric).

See also

ExtractFloatData(), ExtractStringData()

10.37.3.5 ExtractStringData()

```
const char * DDFSubfieldDefn::ExtractStringData (
            const char * pachSourceData,
            int nMaxBytes,
            int * pnConsumedBytes )
```

Extract a zero terminated string containing the data for this subfield. Given a pointer to the data for this subfield (from within a DDFRecord) this method will return the data for this subfield. The number of bytes consumed as part of this field can also be fetched. This number may be one longer than the string length if there is a terminator character used.

This function will return the raw binary data of a subfield for types other than DDFString, including data past zero chars. This is the standard way of extracting DDFBinaryString subfields for instance.

Parameters

pachSourceData	The pointer to the raw data for this field. This may have come from DDFRecord::GetData(), taking into account skip factors over previous subfields data.
nMaxBytes	The maximum number of bytes that are accessible after pachSourceData.
pnConsumedBytes	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

A pointer to a buffer containing the data for this field. The returned pointer is to an internal buffer which is invalidated on the next ExtractStringData() call on this DDFSubfieldDefn(). It should not be freed by the application.

See also

ExtractIntData(), ExtractFloatData()

10.37.3.6 FormatFloatValue()

```
int DDFSubfieldDefn::FormatFloatValue (
            char * pachData,
            int nBytesAvailable,
            int * pnBytesUsed,
            double dfNewValue ) const
```

Format float subfield value.

Returns a buffer with the passed in float value reformatted in a way suitable for storage in a DDFField for this subfield.

10.37.3.7 FormatIntValue()

```
int DDFSubfieldDefn::FormatIntValue (
            char * pachData,
            int nBytesAvailable,
            int * pnBytesUsed,
            int nNewValue ) const
```

Format int subfield value.

Returns a buffer with the passed in int value reformatted in a way suitable for storage in a DDFField for this subfield.

10.37.3.8 FormatStringValue()

```
int DDFSubfieldDefn::FormatStringValue (
            char * pachData,
            int nBytesAvailable,
            int * pnBytesUsed,
            const char * pszValue,
            int nValueLength = -1 ) const
```

Format string subfield value.

Returns a buffer with the passed in string value reformatted in a way suitable for storage in a DDFField for this subfield.

10.37.3.9 GetDataLength()

```
int DDFSubfieldDefn::GetDataLength (
            const char * pachSourceData,
            int nMaxBytes,
            int * pnConsumedBytes )
```

Scan for the end of variable length data. Given a pointer to the data for this subfield (from within a DDFRecord) this method will return the number of bytes which are data for this subfield. The number of bytes consumed as part of this field can also be fetched. This number may be one longer than the length if there is a terminator character used.

This method is mainly for internal use, or for applications which want the raw binary data to interpret themselves. Otherwise use one of ExtractStringData(), ExtractIntData() or ExtractFloatData().

Parameters

pachSourceData	The pointer to the raw data for this field. This may have come from DDFRecord::GetData(), taking into account skip factors over previous subfields data.
nMaxBytes	The maximum number of bytes that are accessible after pachSourceData.
pnConsumedBytes	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore.

Returns

The number of bytes at pachSourceData which are actual data for this record (not including unit, or field terminator).

10.37.3.10 GetDefaultValue()

```
int DDFSubfieldDefn::GetDefaultValue (
            char * pachData,
            int nBytesAvailable,
            int * pnBytesUsed ) const
```

Get default data.

Returns the default subfield data contents for this subfield definition. For variable length numbers this will normally be "0<unit-terminator>". For variable length strings it will be "<unit-terminator>". For fixed length numbers it is zero filled. For fixed length strings it is space filled. For binary numbers it is binary zero filled.

Parameters

pachData	the buffer into which the returned default will be placed. May be NULL if just querying default size.
nBytesAvailable	the size of pachData in bytes.
pnBytesUsed	will receive the size of the subfield default data in bytes.

Returns

TRUE on success or FALSE on failure or if the passed buffer is too small to hold the default.

10.37.3.11 GetFormat()

```
const char* DDFSubfieldDefn::GetFormat ( ) const  [inline]
```

Get pointer to subfield format string

10.37.3.12 GetName()

```
const char* DDFSubfieldDefn::GetName ( ) const  [inline]
```

Get pointer to subfield name.

10.37.3.13 GetType()

```
DDFDataType DDFSubfieldDefn::GetType ( ) const  [inline]
```

Get the general type of the subfield. This can be used to determine which of ExtractFloatData(), ExtractIntData() or ExtractStringData() should be used.

Returns

The subfield type. One of DDFInt, DDFFloat, DDFString or DDFBinaryString.

10.37.3.14 GetWidth()

```
int DDFSubfieldDefn::GetWidth ( ) const  [inline]
```

Get the subfield width (zero for variable).

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

- iso8211.h
- ddfsubfielddefn.cpp

10.38 DefaultCSVFileNameTLS Struct Reference

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

- cpl_csv.cpp

10.39 errHandler Struct Reference

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

- cpl_error.cpp

10.40 file_in_zip_read_info_s Struct Reference

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

- cpl_minizip_unzip.cpp

10.41 FindFileTLS Struct Reference

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

- cpl_findfile.cpp

10.42 GDALScaledProgressInfo Struct Reference

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

- cpl_progress.cpp

10.43 linkedlist_data_s Struct Reference

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

- cpl_minizip_zip.cpp

10.44 linkedlist_datablock_internal_s Struct Reference

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

- cpl_minizip_zip.cpp

10.45 ParseContext Struct Reference

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

- cpl_minixml.cpp

10.46 SDTS_CATD Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- const char * GetEntryTypeDesc (int) const
- const char * GetEntryFilePath (int) const
- SDTSLayerType GetEntryType (int) const

10.46.1 Detailed Description

Class for accessing the CATD (Catalog Directory) file containing a list of all other files (modules) in the transfer.

10.46.2 Member Function Documentation

10.46.2.1 GetEntryFilePath()

```
const char * SDTS_CATD::GetEntryFilePath (
            int iEntry ) const
```

Fetch the full filename of the requested module.

Parameters

iEntry	The module index within the CATD catalog. A number from zero to GetEntryCount()-1.

Returns

A pointer to an internal string containing the filename. This string should not be altered, or freed by the application.

10.46.2.2 GetEntryType()

```
SDTSLayerType SDTS_CATD::GetEntryType (
            int iEntry ) const
```

Fetch the enumerated type of a module in the catalog.

Parameters

iEntry	The module index within the CATD catalog. A number from zero to GetEntryCount()-1.

Returns

A value from the SDTSLayerType enumeration indicating the type of the module, and indicating the corresponding type of reader.

- SLTPoint: Read with SDTSPointReader, underlying type of `Point-Node`.
- SLTLine: Read with SDTSLineReader, underlying type of `Line`.
- SLTAttr: Read with SDTSAttrReader, underlying type of `Attribute Primary` or `Attribute Secondary`.
- SLTPolygon: Read with SDTSPolygonReader, underlying type of `Polygon`.

10.46.2.3 GetEntryTypeDesc()

```
const char * SDTS_CATD::GetEntryTypeDesc (
            int iEntry ) const
```

Fetch the type description of a module in the catalog.

Parameters

iEntry	The module index within the CATD catalog. A number from zero to GetEntryCount()-1.

Returns

A pointer to an internal string with the type description for this module. This is from the CATD file (subfield TYPE of field CATD), and will be something like "Attribute Primary ".

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

- sdts_al.h
- sdtscatd.cpp

10.47 SDTS_CATDEntry Class Reference

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

- sdtscatd.cpp

10.48 SDTS_IREF Class Reference

```
#include <sdts_al.h>
```

10.48.1 Detailed Description

Class holding SDTS IREF (internal reference) information, internal coordinate system format, scaling and resolution. This object isn't normally needed by applications.

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

- sdts_al.h
- sdtsiref.cpp

10.49 SDTS_XREF Class Reference

```
#include <sdts_al.h>
```

Public Attributes

- char * pszSystemName
- char * pszDatum
- int nZone

10.49.1 Detailed Description

Class for reading the XREF (external reference) module containing the data projection definition.

10.49.2 Member Data Documentation

10.49.2.1 nZone

```
int SDTS_XREF::nZone
```

Zone number for UTM and SPCS projections, from the ZONE field.

10.49.2.2 pszDatum

```
char* SDTS_XREF::pszDatum
```

Horizontal datum name, from the HDAT field. One of NAS, NAX, WGA, WGB, WGC, WGE.

10.49.2.3 pszSystemName

```
char* SDTS_XREF::pszSystemName
```

Projection system name, from the RSNM field. One of GEO, SPCS, UTM, UPS, OTHR, UNSP.

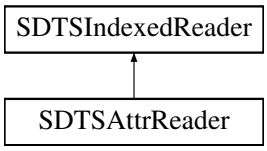
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtsxref.cpp

10.50 SDTSAttrReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSAttrReader:



Public Member Functions

- int IsSecondary () const

10.50.1 Detailed Description

Class for reading SDTSAttrRecord features from a primary or secondary attribute module.

10.50.2 Member Function Documentation

10.50.2.1 IsSecondary()

```
int SDTSAttrReader::IsSecondary ( ) const  [inline]
```

Returns TRUE if this is a Attribute Secondary layer rather than an Attribute Primary layer.

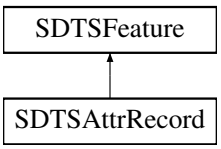
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtsattrreader.cpp

10.51 SDTSAttrRecord Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSAttrRecord:



Public Member Functions

- virtual void Dump (FILE ∗) override

Public Attributes

- DDFRecord ∗ poWholeRecord
- DDFField ∗ poATTR

10.51.1 Detailed Description

SDTS attribute record feature, as read from A∗ modules by SDTSAttrReader.

Note that even though SDTSAttrRecord is derived from SDTSFeature, there are never any attribute records associated with attribute records using the aoATID[] mechanism. SDTSFeature::nAttributes will always be zero.

10.51.2 Member Function Documentation

10.51.2.1 Dump()

```
void SDTSAttrRecord::Dump (
            FILE *  )  [override], [virtual]
```

Dump readable description of feature to indicated stream.

Implements SDTSFeature.

10.51.3 Member Data Documentation

10.51.3.1 poATTR

```
DDFField* SDTSAttrRecord::poATTR
```

The ATTR DDFField with the user attribute. Each subfield is a attribute value.

10.51.3.2 poWholeRecord

```
DDFRecord* SDTSAttrRecord::poWholeRecord
```

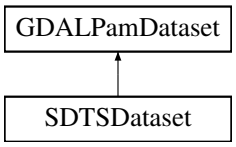
The entire DDFRecord read from the file.

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

- sdts_al.h
- sdtsattrreader.cpp

10.52 SDTSDataset Class Reference

Inheritance diagram for SDTSDataset:



Friends

- class **SDTSRasterBand**

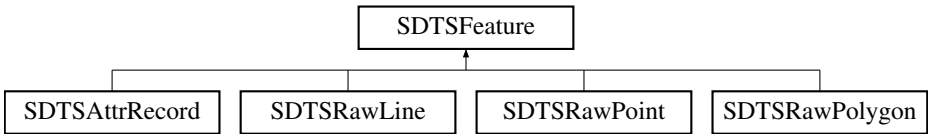
The documentation for this class was generated from the following file:

- sdtsdataset.cpp

10.53 SDTSFeature Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSFeature:



Public Member Functions

- virtual void Dump (FILE *)=0

Public Attributes

- SDTSModId oModId
- int nAttributes
- SDTSModId * paoATID

10.53.1 Detailed Description

Base class for various SDTS features classes, providing a generic module identifier, and list of attribute references.

10.53.2 Member Function Documentation

10.53.2.1 Dump()

```
virtual void SDTSFeature::Dump (
            FILE *  )  [pure virtual]
```

Dump readable description of feature to indicated stream.

Implemented in SDTSRawPolygon, SDTSRawPoint, SDTSAttrRecord, and SDTSRawLine.

10.53.3 Member Data Documentation

10.53.3.1 nAttributes

```
int SDTSFeature::nAttributes
```

Number of attribute links (aoATID[]) on this feature.

10.53.3.2 oModId

```
SDTSModId SDTSFeature::oModId
```

Unique identifier for this record/feature within transfer.

10.53.3.3 paoATID

```
SDTSModId* SDTSFeature::paoATID
```

List of nAttributes attribute record identifiers related to this feature.

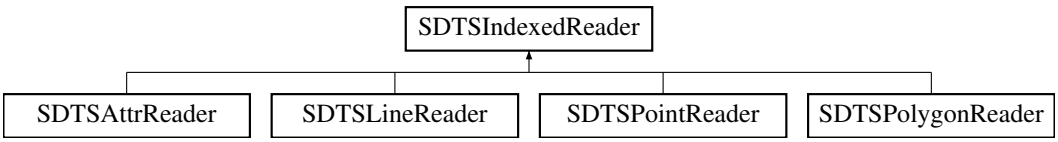
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtslib.cpp

10.54 SDTSIndexedReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSIndexedReader:



Public Member Functions

- SDTSFeature * GetNextFeature ()
- virtual void Rewind ()
- void FillIndex ()
- void ClearIndex ()
- int IsIndexed () const
- SDTSFeature * GetIndexedFeatureRef (int)
- char ** ScanModuleReferences (const char *="ATID")

10.54.1 Detailed Description

Base class for all the SDTSFeature type readers. Provides feature caching semantics and fetching based on a record number.

10.54.2 Member Function Documentation

10.54.2.1 ClearIndex()

```
void SDTSIndexedReader::ClearIndex ( )
```

Free all features in the index (if filled).

After this the reader is considered to not be indexed, and IsIndexed() will return FALSE until the index is forcibly filled again.

10.54.2.2 FillIndex()

```
void SDTSIndexedReader::FillIndex ( )
```

Read all features into a memory indexed cached.

The ClearIndex() method can be used to free all indexed features. FillIndex() does nothing, if an index has already been built.

10.54.2.3 GetIndexedFeatureRef()

```
SDTSFeature * SDTSIndexedReader::GetIndexedFeatureRef (
            int iRecordId )
```

Fetch a feature based on its record number.

This method will forcibly fill the feature cache, reading all the features in the file into memory, if they haven't already been loaded. The ClearIndex() method can be used to flush this cache when no longer needed.

Parameters

i↩ RecordId	the record to fetch, normally based on the nRecord field of an SDTSModId.

Returns

a pointer to an internal feature (not to be deleted) or NULL if there is no matching feature.

10.54.2.4 GetNextFeature()

```
SDTSFeature * SDTSIndexedReader::GetNextFeature ( )
```

Fetch the next available feature from this reader.

The returned SDTSFeature * is to an internal indexed object if the IsIndexed() method returns TRUE, otherwise the returned feature becomes the responsibility of the caller to destroy with delete.

Note that the Rewind() method can be used to start over at the beginning of the modules feature list.

Returns

next feature, or NULL if no more are left. Please review above ownership/delete semantics.

10.54.2.5 IsIndexed()

```
int SDTSIndexedReader::IsIndexed ( ) const
```

Returns TRUE if the module is indexed, otherwise it returns FALSE.

If the module is indexed all the feature have already been read into memory, and searches based on the record number can be performed efficiently.

10.54.2.6 Rewind()

```
void SDTSIndexedReader::Rewind ( )  [virtual]
```

Rewind so that the next feature returned by GetNextFeature() will be the first in the module.

10.54.2.7 ScanModuleReferences()

```
char ** SDTSIndexedReader::ScanModuleReferences (
            const char * pszFName = "ATID" )
```

Scan an entire SDTS module for record references with the given field name.

The fields are required to have a MODN subfield from which the module is extracted.

This method is normally used to find all the attribute modules referred to by a point, line or polygon module to build a unified schema.

This method will have the side effect of rewinding unindexed readers because the scanning operation requires reading all records in the module from disk.

Parameters

pszFName	the field name to search for. By default "ATID" is used.

Returns

a NULL terminated list of module names. Free with CSLDestroy().

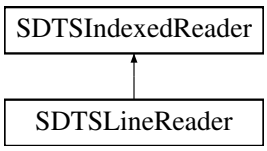
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtsindexedreader.cpp

10.55 SDTSLineReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSLineReader:



Public Member Functions

- void AttachToPolygons (SDTSTransfer *, int iPolyLayer)

10.55.1 Detailed Description

Reader for SDTS line modules.

Returns SDTSRawLine features. Normally readers are instantiated with the SDTSTransfer::GetIndexedReader() method.

10.55.2 Member Function Documentation

10.55.2.1 AttachToPolygons()

```
void SDTSLineReader::AttachToPolygons (
            SDTSTransfer * poTransfer,
            int iTargetPolyLayer )
```

Attach lines in this module to their polygons as the first step in polygon formation.

See also the SDTSRawPolygon::AssembleRings() method.

Parameters

poTransfer	the SDTSTransfer of this SDTSLineReader, and from which the related SDTSPolygonReader will be instantiated.
iTargetPolyLayer	the polygon reader instance number, used to avoid processing lines for other layers.

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

- sdts_al.h
- sdtslinereader.cpp

10.56 SDTSModId Class Reference

```
#include <sdts_al.h>
```

Public Attributes

- char szModule [8]
- int nRecord
- char szOBRP [8]
- char szName [20]

10.56.1 Detailed Description

Object representing a unique module/record identifier within an SDTS transfer.

10.56.2 Member Data Documentation

10.56.2.1 nRecord

```
int SDTSModId::nRecord
```

The record within the module referred to. This is -1 for unused SDTSModIds.

10.56.2.2 szModule

```
char SDTSModId::szModule[8]
```

The module name, such as PC01, containing the indicated record.

10.56.2.3 szName

```
char SDTSModId::szName[20]
```

String "szModule:nRecord"

10.56.2.4 szOBRP

```
char SDTSModId::szOBRP[8]
```

The "role" of this record within the module. This is normally empty for references, but set in the oModId member of a feature.

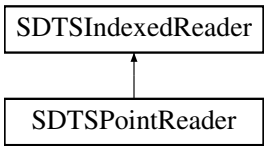
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtslib.cpp

10.57 SDTSPointReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSPointReader:



Additional Inherited Members

10.57.1 Detailed Description

Class for reading SDTSRawPoint features from a point module (type NA, NO or NP).

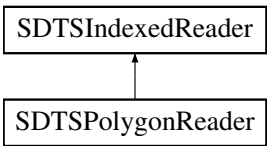
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtspointreader.cpp

10.58 SDTSPolygonReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSPolygonReader:



Public Member Functions

- void AssembleRings (SDTSTransfer *, int iPolyLayer)

10.58.1 Detailed Description

Class for reading SDTSRawPolygon features from a polygon (PC*) module.

10.58.2 Member Function Documentation

10.58.2.1 AssembleRings()

```
void SDTSPolygonReader::AssembleRings (
            SDTSTransfer * poTransfer,
            int iPolyLayer )
```

Assemble geometry for a polygon transfer.

This method takes care of attaching lines from all the line layers in this transfer to this polygon layer, assembling the lines into rings on the polygons, and then cleaning up unnecessary intermediate results.

Currently this method will leave the line layers rewound to the beginning but indexed, and the polygon layer rewound but indexed. In the future it may restore reading positions, and possibly flush line indexes if they were not previously indexed.

This method does nothing if the rings have already been assembled on this layer using this method.

See SDTSRawPolygon::AssembleRings() for more information on how the lines are assembled into rings.

Parameters

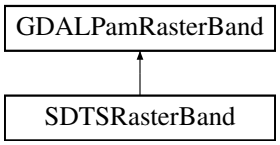
poTransfer	the SDTSTransfer that this reader is a part of. Used to get a list of line layers that might be needed.
iPolyLayer	the polygon reader instance number, used to avoid processing lines for other layers.

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

- sdts_al.h
- sdtspolygonreader.cpp

10.59 SDTSRasterBand Class Reference

Inheritance diagram for SDTSRasterBand:



Friends

- class **SDTSDataset**

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

- sdtsdataset.cpp

10.60 SDTSRasterReader Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- int GetRasterType ()
- int GetTransform (double ∗)
- int GetMinMax (double ∗pdfMin, double ∗pdfMax, double dfNoData)
- int GetXSize () const
- int GetYSize () const
- int GetBlockXSize () const
- int GetBlockYSize () const
- int GetBlock (int nXOffset, int nYOffset, void ∗pData)

10.60.1 Detailed Description

Class for reading raster data from a raster layer.

This class is somewhat unique among the reader classes in that it isn't derived from SDTSIndexedFeature, and it doesn't return "features". Instead it is used to read raster blocks, in the natural block size of the dataset.

10.60.2 Member Function Documentation

10.60.2.1 GetBlock()

```
int SDTSRasterReader::GetBlock (
            int nXOffset,
            int nYOffset,
            void * pData )
```

Read a block of raster data from the file.

Parameters

nXOffset	X block offset into the file. Normally zero for scanline organized raster files.
nYOffset	Y block offset into the file. Normally the scanline offset from top of raster for scanline organized raster files.
pData	pointer to GInt16 (signed short) buffer of data into which to read the raster.

Returns

TRUE on success and FALSE on error.

10.60.2.2 GetBlockXSize()

```
int SDTSRasterReader::GetBlockXSize ( ) const  [inline]
```

Fetch the width of a source block (usually same as raster width).

10.60.2.3 GetBlockYSize()

```
int SDTSRasterReader::GetBlockYSize ( ) const  [inline]
```

Fetch the height of a source block (usually one).

10.60.2.4 GetMinMax()

```
int SDTSRasterReader::GetMinMax (
            double * pdfMin,
            double * pdfMax,
            double dfNoData )
```

Fetch the minimum and maximum raster values that occur in the file.

Note this operation current results in a scan of the entire file.

Parameters

pdfMin	variable in which the minimum value encountered is returned.
pdfMax	variable in which the maximum value encountered is returned.
dfNoData	a value to ignore when computing min/max, defaults to -32766.

Returns

TRUE on success, or FALSE if an error occurs.

10.60.2.5 GetRasterType()

```
int SDTSRasterReader::GetRasterType ( )
```

Fetch the pixel data type.

Returns one of SDTS_RT_INT16 (1) or SDTS_RT_FLOAT32 (6) indicating the type of buffer that should be passed to GetBlock().

10.60.2.6 GetTransform()

```
int SDTSRasterReader::GetTransform (
            double * padfTransformOut )
```

Fetch the transformation between pixel/line coordinates and georeferenced coordinates.

Parameters

padfTransformOut	pointer to an array of six doubles which will be filled with the georeferencing transform.

Returns

TRUE is returned, indicating success.

The padfTransformOut array consists of six values. The pixel/line coordinate (Xp,Yp) can be related to a georeferenced coordinate (Xg,Yg) or (Easting, Northing).

```
Xg = padfTransformOut[0] + Xp * padfTransform[1] + Yp * padfTransform[2]
Yg = padfTransformOut[3] + Xp * padfTransform[4] + Yp * padfTransform[5]
```

In other words, for a north up image the top left corner of the top left pixel is at georeferenced coordinate (padf↩Transform[0],padfTransform[3]) the pixel width is padfTransform[1], the pixel height is padfTransform[5] and padf↩Transform[2] and padfTransform[4] will be zero.

10.60.2.7 GetXSize()

```
int SDTSRasterReader::GetXSize ( ) const  [inline]
```

Fetch the raster width.

Returns

the width in pixels.

10.60.2.8 GetYSize()

```
int SDTSRasterReader::GetYSize ( ) const  [inline]
```

Fetch the raster height.

Returns

the height in pixels.

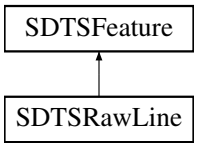
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtsrasterreader.cpp

10.61 SDTSRawLine Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSRawLine:



Public Member Functions

- void Dump (FILE *) override

Public Attributes

- int nVertices
- double * padfX
- double * padfY
- double * padfZ
- SDTSModId oLeftPoly
- SDTSModId oRightPoly
- SDTSModId oStartNode
- SDTSModId oEndNode

10.61.1 Detailed Description

SDTS line feature, as read from LE* modules by SDTSLineReader.

10.61.2 Member Function Documentation

10.61.2.1 Dump()

```
void SDTSRawLine::Dump (
            FILE *  )  [override], [virtual]
```

Dump readable description of feature to indicated stream.

Implements SDTSFeature.

10.61.3 Member Data Documentation

10.61.3.1 nVertices

```
int SDTSRawLine::nVertices
```

Number of vertices in the padfX, padfY and padfZ arrays.

10.61.3.2 oEndNode

```
SDTSModId SDTSRawLine::oEndNode
```

Identifier for the end node of this line. This is the SDTS ENID subfield.

10.61.3.3 oLeftPoly

```
SDTSModId SDTSRawLine::oLeftPoly
```

Identifier of polygon to left of this line. This is the SDTS PIDL subfield.

10.61.3.4 oRightPoly

```
SDTSModId SDTSRawLine::oRightPoly
```

Identifier of polygon to right of this line. This is the SDTS PIDR subfield.

10.61.3.5 oStartNode

```
SDTSModId SDTSRawLine::oStartNode
```

Identifier for the start node of this line. This is the SDTS SNID subfield.

10.61.3.6 padfX

```
double* SDTSRawLine::padfX
```

List of nVertices X coordinates.

10.61.3.7 padfY

```
double* SDTSRawLine::padfY
```

List of nVertices Y coordinates.

10.61.3.8 padfZ

```
double* SDTSRawLine::padfZ
```

List of nVertices Z coordinates - currently always zero.

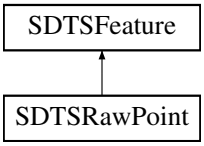
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtslinereader.cpp

10.62 SDTSRawPoint Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSRawPoint:



Public Member Functions

- virtual void Dump (FILE *) override

Public Attributes

- double dfX
- double dfY
- double dfZ
- SDTSModId oAreaId

10.62.1 Detailed Description

Object containing a point feature (type NA, NO or NP).

10.62.2 Member Function Documentation

10.62.2.1 Dump()

```
void SDTSRawPoint::Dump (
            FILE *  )  [override], [virtual]
```

Dump readable description of feature to indicated stream.

Implements SDTSFeature.

10.62.3 Member Data Documentation

10.62.3.1 dfX

```
double SDTSRawPoint::dfX
```

X coordinate of point.

10.62.3.2 dfY

```
double SDTSRawPoint::dfY
```

Y coordinate of point.

10.62.3.3 dfZ

```
double SDTSRawPoint::dfZ
```

Z coordinate of point.

10.62.3.4 oAreaId

```
SDTSModId SDTSRawPoint::oAreaId
```

Optional identifier of area marked by this point (i.e. PC01:27).

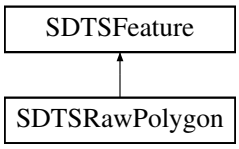
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtspointreader.cpp

10.63 SDTSRawPolygon Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSRawPolygon:



Public Member Functions

- int AssembleRings ()
- virtual void Dump (FILE *) override

Public Attributes

- int nRings
- int nVertices
- int * panRingStart
- double * padfX
- double * padfY
- double * padfZ

10.63.1 Detailed Description

Class for holding information about a polygon feature.

When directly read from a polygon module, the polygon has no concept of its geometry. Just its ID, and references to attribute records. However, if the SDTSLineReader::AttachToPolygons() method is called on the module containing the lines forming the polygon boundaries, then the nEdges/papoEdges information on the SDTSRawPolygon will be filled in.

Once this is complete the AssembleRings() method can be used to fill in the nRings/nVertices/panRingStart/padf↩X/padfY/padfZ information defining the ring geometry.

Note that the rings may not appear in any particular order, nor with any meaningful direction (clockwise or counter-clockwise).

10.63.2 Member Function Documentation

10.63.2.1 AssembleRings()

```
int SDTSRawPolygon::AssembleRings ( )
```

This method will assemble the edges associated with a polygon into rings, returning FALSE if problems are encountered during assembly.

Form border lines (arcs) into outer and inner rings.

See SDTSPolygonReader::AssemblePolygons() for a simple one step process to assembling geometry for all polygons in a transfer.

This method will assemble the lines attached to a polygon into an outer ring, and zero or more inner rings. Before calling it is necessary that all the lines associated with this polygon have already been attached. Normally this is accomplished by calling SDTSLineReader::AttachToPolygons() on all line layers that might contain edges related to this layer.

This method then forms the lines into rings. Rings are formed by:

1. Take a previously unconsumed line, and start a ring with it. Mark it as consumed, and keep track of its start and end node ids as being the start and end node ids of the ring.
2. If the rings start id is the same as the end node id then this ring is completely formed, return to step 1.

3. Search all unconsumed lines for a line with the same start or end node id as the rings current node id. If none are found then the assembly has failed. Return to step 1 but report failure on completion.
4. Once found, add the line to the current ring, dropping the duplicated vertex and reverse order if necessary. Mark the line as consumed, and update the rings end node id accordingly.
5. go to step 2.

Once ring assembly from lines is complete, another pass is made to order the rings such that the exterior ring is first, the first ring has counter-clockwise vertex ordering and the inner rings have clockwise vertex ordering. This is accomplished based on the assumption that the outer ring has the largest area, and using the +/- sign of area to establish direction of rings.

Returns

TRUE if all rings assembled without problems or FALSE if a problem occurred. If a problem occurs rings are still formed from all lines, but some of the rings will not be closed, and rings will have no particular order or direction.

10.63.2.2 Dump()

```
void SDTSRawPolygon::Dump (
            FILE *  )  [override], [virtual]
```

Dump readable description of feature to indicated stream.

Implements SDTSFeature.

10.63.3 Member Data Documentation

10.63.3.1 nRings

```
int SDTSRawPolygon::nRings
```

Number of rings in assembled polygon.

10.63.3.2 nVertices

```
int SDTSRawPolygon::nVertices
```

Total number of vertices in all rings of assembled polygon.

10.63.3.3 padfX

```
double* SDTSRawPolygon::padfX
```

List of nVertices X coordinates for the polygon (split over multiple rings via panRingStart.

10.63.3.4 padfY

```
double* SDTSRawPolygon::padfY
```

List of nVertices Y coordinates for the polygon (split over multiple rings via panRingStart.

10.63.3.5 padfZ

```
double* SDTSRawPolygon::padfZ
```

List of nVertices Z coordinates for the polygon (split over multiple rings via panRingStart. The values are almost always zero.

10.63.3.6 panRingStart

```
int* SDTSRawPolygon::panRingStart
```

Offsets into padfX/padfY/padfZ for the beginning of each ring in the polygon. This array is nRings long.

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

- sdts_al.h
- sdtspolygonreader.cpp

10.64 SDTSTransfer Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- int Open (const char *)
- int FindLayer (const char *)
- SDTSLayerType GetLayerType (int) const
- int GetLayerCATDEntry (int) const
- SDTSRasterReader * GetLayerRasterReader (int)
- SDTSIndexedReader * GetLayerIndexedReader (int)
- SDTS_CATD * GetCATD ()
- SDTS_XREF * GetXREF ()
- DDFField * GetAttr (SDTSModId *)
- int GetBounds (double *pdfMinX, double *pdfMinY, double *pdfMaxX, double *pdfMaxY)

10.64.1 Detailed Description

Master class representing an entire SDTS transfer.

This class is used to open the transfer, to get a list of available feature layers, and to instantiate readers for those layers.

10.64.2 Member Function Documentation

10.64.2.1 FindLayer()

```
int SDTSTransfer::FindLayer (
            const char * pszModule )
```

Fetch the SDTSTransfer layer number corresponding to a module name.

Parameters

pszModule	the name of the module to search for, such as "PC01".

Returns

the layer number (between 0 and GetLayerCount()-1 corresponding to the module, or -1 if it doesn't correspond to a layer.

10.64.2.2 GetAttr()

```
DDFField * SDTSTransfer::GetAttr (
            SDTSModId * poModId )
```

Fetch the attribute fields given a particular module/record id.

Parameters

po↩ ModId	an attribute record identifier, normally taken from the aoATID[] array of an SDTSIndexedFeature.

Returns

a pointer to the DDFField containing the user attribute values as subfields.

10.64.2.3 GetBounds()

```
int SDTSTransfer::GetBounds (
            double * pdfMinX,
            double * pdfMinY,
            double * pdfMaxX,
            double * pdfMaxY )
```

Fetch approximate bounds for a transfer by scanning all point layers and raster layers.

For TVP datasets (where point layers are scanned) the results can, in theory miss some lines that go outside the bounds of the point layers. However, this isn't common since most TVP sets contain a bounding rectangle whose corners will define the most extreme extents.

Parameters

pdfMinX	western edge of dataset
pdfMinY	southern edge of dataset
pdfMaxX	eastern edge of dataset
pdfMaxY	northern edge of dataset

Returns

TRUE if success, or FALSE on a failure.

10.64.2.4 GetCATD()

```
SDTS_CATD* SDTSTransfer::GetCATD ( ) [inline]
```

Fetch the catalog object for this transfer.

Returns

pointer to the internally managed SDTS_CATD for the transfer.

10.64.2.5 GetLayerCATDEntry()

```
int SDTSTransfer::GetLayerCATDEntry (
            int iEntry ) const
```

Fetch the CATD module index for a layer. This can be used to fetch details about the layer/module from the SDTS_CATD object, such as its filename, and description.

Parameters

iEntry	the layer index from 0 to GetLayerCount()-1.

Returns

the module index suitable for use with the various SDTS_CATD methods.

10.64.2.6 GetLayerIndexedReader()

```
SDTSIndexedReader * SDTSTransfer::GetLayerIndexedReader (
            int iEntry )
```

Returns a pointer to a reader of the appropriate type to the requested layer.

Notes:

- The returned reader remains owned by the SDTSTransfer, and will be destroyed when the SDTSTransfer is destroyed. It should not be destroyed by the application.
- If an indexed reader was already created for this layer using GetLayerIndexedReader(), it will be returned instead of creating a new reader. Among other things this means that the returned reader may not be positioned to read from the beginning of the module, and may already have its index filled.
- The returned reader will be of a type appropriate to the layer. See SDTSTransfer::GetLayerType() to see what reader classes correspond to what layer types, so it can be cast accordingly (if necessary).

Parameters

iEntry	the index of the layer to instantiate a reader for. A value between 0 and GetLayerCount()-1.

Returns

a pointer to an appropriate reader or NULL if the method fails.

10.64.2.7 GetLayerRasterReader()

```
SDTSRasterReader * SDTSTransfer::GetLayerRasterReader (
            int iEntry )
```

Instantiate an SDTSRasterReader for the indicated layer.

Parameters

iEntry	the index of the layer to instantiate a reader for. A value between 0 and GetLayerCount()-1.

Returns

a pointer to a new SDTSRasterReader object, or NULL if the method fails.

NOTE: The reader returned from GetLayerRasterReader() becomes the responsibility of the caller to delete, and isn't automatically deleted when the SDTSTransfer is destroyed. This method is different from the GetLayerIndexedReader() method in this regard.

10.64.2.8 GetLayerType()

```
SDTSLayerType SDTSTransfer::GetLayerType (
            int iEntry ) const
```

Fetch type of requested feature layer.

Parameters

iEntry	the index of the layer to fetch information on. A value from zero to GetLayerCount()-1.

Returns

the layer type.

- SLTPoint: A point layer. An SDTSPointReader is returned by SDTSTransfer::GetLayerIndexedReader().
- SLTLine: A line layer. An SDTSLineReader is returned by SDTSTransfer::GetLayerIndexedReader().
- SLTAttr: An attribute primary or secondary layer. An SDTSAttrReader is returned by SDTSTransfer::GetLayerIndexedReader().

- SLTPoly: A polygon layer. An SDTSPolygonReader is returned by SDTSTransfer::GetLayerIndexedReader().
- SLTRaster: A raster layer. SDTSTransfer::GetLayerIndexedReader() is not implemented. Use SDTSTransfer::GetLayerRasterReader() instead.

10.64.2.9 GetXREF()

```
SDTS_XREF* SDTSTransfer::GetXREF ( ) [inline]
```

Fetch the external reference object for this transfer.

Returns

pointer to the internally managed SDTS_XREF for the transfer.

10.64.2.10 Open()

```
int SDTSTransfer::Open (
            const char * pszFilename )
```

Open an SDTS transfer, and establish a list of data layers in the transfer.

Parameters

pszFilename	The name of the CATD file within the transfer.

Returns

TRUE if the open success, or FALSE if it fails.

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

- sdts_al.h
- sdtstransfer.cpp

10.65 SFRegion Class Reference

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

- cpl_vsil_sparsefile.cpp

10.66 StackContext Struct Reference

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

- cpl_minixml.cpp

10.67 tm_unz_s Struct Reference

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

- cpl_minizip_unzip.h

10.68 tm_zip_s Struct Reference

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

- cpl_minizip_zip.h

10.69 TupleEnvVarOptionName Struct Reference

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

- cpl_http.cpp

10.70 unz_file_info_internal_s Struct Reference

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

- cpl_minizip_unzip.cpp

10.71 unz_file_info_s Struct Reference

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

- cpl_minizip_unzip.h

10.72 unz_file_pos_s Struct Reference

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

- cpl_minizip_unzip.h

10.73 unz_global_info_s Struct Reference

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

- cpl_minizip_unzip.h

10.74 unz_s Struct Reference

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

- cpl_minizip_unzip.cpp

10.75 VSIArchiveContent Class Reference

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

- cpl_vsi_virtual.h

10.76 VSIArchiveEntry Struct Reference

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

- cpl_vsi_virtual.h

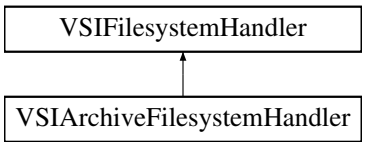
10.77 VSIArchiveEntryFileOffset Class Reference

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

- cpl_vsi_virtual.h

10.78 VSIArchiveFilesystemHandler Class Reference

Inheritance diagram for VSIArchiveFilesystemHandler:



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

- cpl_vsi_virtual.h

10.79 VSIArchiveReader Class Reference

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

- cpl_vsi_virtual.h

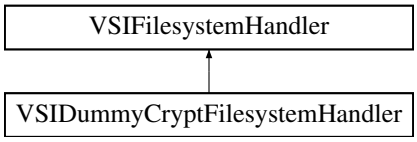
10.80 VSIDIR Struct Reference

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

- vsipreload.cpp

10.81 VSIDummyCryptFilesystemHandler Class Reference

Inheritance diagram for VSIDummyCryptFilesystemHandler:



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

- cpl_vsil_crypt.cpp

10.82 VSIErrorContext Struct Reference

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

- cpl_vsi_error.cpp

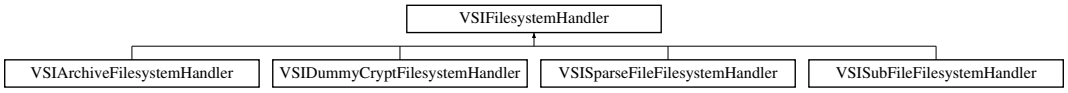
10.83 VSIFileManager Class Reference

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

- cpl_vsi_virtual.h
- cpl_vsil.cpp

10.84 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:



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

- cpl_vsi_virtual.h
- cpl_vsil.cpp

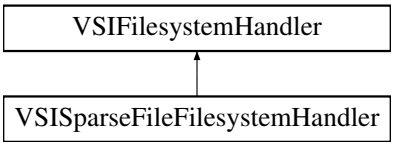
10.85 VSIReadDirRecursiveTask Struct Reference

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

- cpl_vsil.cpp

10.86 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

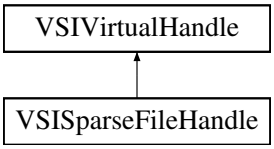


The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

10.87 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:



Public Member Functions

- int Seek (vsi_l_offset nOffset, int nWhence) override

  Seek to requested offset.
- vsi_l_offset Tell () override

  Tell current file offset.
- size_t Read (void *pBuffer, size_t nSize, size_t nMemb) override

  Read bytes from file.
- size_t Write (const void *pBuffer, size_t nSize, size_t nMemb) override

  Write bytes to file.
- int Eof () override

  Test for end of file.
- int Close () override

  Close file.

10.87.1 Member Function Documentation

10.87.1.1 Close()

```
int VSISparseFileHandle::Close ( )  [override], [virtual]
```

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.

Returns

0 on success or -1 on failure.

Implements VSIVirtualHandle.

10.87.1.2 Eof()

```
int VSISparseFileHandle::Eof ( ) [override], [virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful VSIFSeekL() call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Returns

TRUE if at EOF else FALSE.

Implements VSIVirtualHandle.

10.87.1.3 Read()

```
size_t VSISparseFileHandle::Read (
            void * pBuffer,
            size_t nSize,
            size_t nCount ) [override], [virtual]
```

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount $*$ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully read.

Implements VSIVirtualHandle.

10.87.1.4 Seek()

```
int VSISparseFileHandle::Seek (
            vsi_l_offset nOffset,
            int nWhence )  [override], [virtual]
```

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Caution: vsi_l_offset is a unsigned type, so SEEK_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek(handle->Tell() + negative_offset, SEEK_SET).

Parameters

nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

Implements VSIVirtualHandle.

10.87.1.5 Tell()

```
vsi_l_offset VSISparseFileHandle::Tell ( )  [override], [virtual]
```

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.

Returns

file offset in bytes.

Implements VSIVirtualHandle.

10.87.1.6 Write()

```
size_t VSISparseFileHandle::Write (
            const void * pBuffer,
            size_t nSize,
            size_t nCount )  [override], [virtual]
```

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters

pBuffer	the buffer from which the data should be written (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully written.

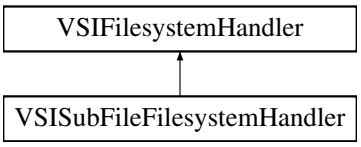
Implements VSIVirtualHandle.

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

- cpl_vsil_sparsefile.cpp

10.88 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

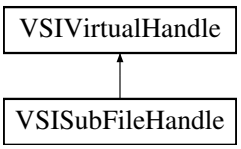


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

10.89 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:



Public Member Functions

- int Seek (vsi_l_offset nOffset, int nWhence) override

  Seek to requested offset.
- vsi_l_offset Tell () override

  Tell current file offset.
- size_t Read (void *pBuffer, size_t nSize, size_t nMemb) override

  Read bytes from file.
- size_t Write (const void *pBuffer, size_t nSize, size_t nMemb) override

  Write bytes to file.
- int Eof () override

  Test for end of file.
- int Close () override

  Close file.

10.89.1 Member Function Documentation

10.89.1.1 Close()

```
int VSISubFileHandle::Close ( )  [override], [virtual]
```

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.

Returns

0 on success or -1 on failure.

Implements VSIVirtualHandle.

10.89.1.2 Eof()

```
int VSISubFileHandle::Eof ( ) [override], [virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful VSIFSeekL() call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Returns

TRUE if at EOF else FALSE.

Implements VSIVirtualHandle.

10.89.1.3 Read()

```
size_t VSISubFileHandle::Read (
            void * pBuffer,
            size_t nSize,
            size_t nCount ) [override], [virtual]
```

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount $*$ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully read.

Implements VSIVirtualHandle.

10.89.1.4 Seek()

```
int VSISubFileHandle::Seek (
            vsi_l_offset nOffset,
            int nWhence )  [override], [virtual]
```

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Caution: vsi_l_offset is a unsigned type, so SEEK_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek(handle->Tell() + negative_offset, SEEK_SET).

Parameters

nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

Implements VSIVirtualHandle.

10.89.1.5 Tell()

```
vsi_l_offset VSISubFileHandle::Tell ( )  [override], [virtual]
```

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.

Returns

file offset in bytes.

Implements VSIVirtualHandle.

10.89.1.6 Write()

```
size_t VSISubFileHandle::Write (
            const void * pBuffer,
            size_t nSize,
            size_t nCount )  [override], [virtual]
```

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters

pBuffer	the buffer from which the data should be written (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully written.

Implements VSIVirtualHandle.

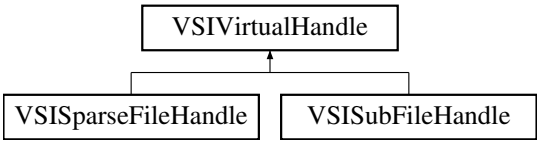
The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

10.90 VSIVirtualHandle Class Reference

```
#include <cpl_vsi_virtual.h>
```

Inheritance diagram for VSIVirtualHandle:



Public Member Functions

- virtual int Seek (vsi_l_offset nOffset, int nWhence)=0

  Seek to requested offset.
- virtual vsi_l_offset Tell ()=0

  Tell current file offset.
- virtual size_t Read (void *pBuffer, size_t nSize, size_t nCount)=0

  Read bytes from file.
- virtual int ReadMultiRange (int nRanges, void **ppData, const vsi_l_offset *panOffsets, const size_t *pan↩ Sizes)

  Read several ranges of bytes from file.
- virtual size_t Write (const void *pBuffer, size_t nSize, size_t nCount)=0

  Write bytes to file.
- virtual int Eof ()=0

  Test for end of file.
- virtual int Flush ()

  Flush pending writes to disk.
- virtual int Close ()=0

  Close file.
- virtual int Truncate (vsi_l_offset nNewSize)

  Truncate/expand the file to the specified size.
- virtual void * GetNativeFileDescriptor ()

  Returns the "native" file descriptor for the virtual handle.
- virtual VSIRangeStatus GetRangeStatus (CPL_UNUSED vsi_l_offset nOffset, CPL_UNUSED vsi_l_offset nLength)

  Return if a given file range contains data or holes filled with zeroes.

10.90.1 Detailed Description

Virtual file handle

10.90.2 Member Function Documentation

10.90.2.1 Close()

```
VSIVirtualHandle::Close ( )  [pure virtual]
```

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.

Returns

0 on success or -1 on failure.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

10.90.2.2 Eof()

```
VSIVirtualHandle::Eof ( ) [pure virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful VSIFSeekL() call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Returns

TRUE if at EOF else FALSE.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

10.90.2.3 Flush()

```
VSIVirtualHandle::Flush ( ) [inline], [virtual]
```

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Returns

0 on success or -1 on error.

10.90.2.4 GetNativeFileDescriptor()

```
VSIVirtualHandle::GetNativeFileDescriptor ( ) [inline], [virtual]
```

Returns the "native" file descriptor for the virtual handle.

This will only return a non-NULL value for "real" files handled by the operating system (to be opposed to GDAL virtual file systems).

On POSIX systems, this will be a integer value ("fd") cast as a void∗. On Windows systems, this will be the HANDLE.

Returns

the native file descriptor, or NULL.

10.90.2.5 GetRangeStatus()

```
VSIVirtualHandle::GetRangeStatus (
            CPL_UNUSED vsi_l_offset nOffset,
            CPL_UNUSED vsi_l_offset nLength )  [inline], [virtual]
```

Return if a given file range contains data or holes filled with zeroes.

This uses the filesystem capabilities of querying which regions of a sparse file are allocated or not. This is currently only implemented for Linux (and no other Unix derivatives) and Windows.

Note: A return of VSI_RANGE_STATUS_DATA doesn't exclude that the extent is filled with zeroes! It must be interpreted as "may contain non-zero data".

Parameters

nOffset	offset of the start of the extent.
nLength	extent length.

Returns

extent status: VSI_RANGE_STATUS_UNKNOWN, VSI_RANGE_STATUS_DATA or VSI_RANGE_STATU↩S_HOLE

Since

GDAL 2.2

10.90.2.6 Read()

```
VSIVirtualHandle::Read (
            void * pBuffer,
            size_t nSize,
            size_t nCount )  [pure virtual]
```

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully read.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

10.90.2.7 ReadMultiRange()

```
int VSIVirtualHandle::ReadMultiRange (
            int nRanges,
            void ** ppData,
            const vsi_l_offset * panOffsets,
            const size_t * panSizes )  [virtual]
```

Read several ranges of bytes from file.

Reads nRanges objects of panSizes[i] bytes from the indicated file at the offset panOffsets[i] into the buffer ppData[i].

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

Parameters

nRanges	number of ranges to read.
ppData	array of nRanges buffer into which the data should be read (ppData[i] must be at list panSizes[i] bytes).
panOffsets	array of nRanges offsets at which the data should be read.
panSizes	array of nRanges sizes of objects to read (in bytes).

Returns

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

10.90.2.8 Seek()

```
int VSIVirtualHandle::Seek (
            vsi_l_offset nOffset,
            int nWhence )  [pure virtual]
```

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Caution: vsi_l_offset is a unsigned type, so SEEK_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek(handle->Tell() + negative_offset, SEEK_SET).

Parameters

nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

10.90.2.9 Tell()

```
VSIVirtualHandle::Tell ( ) [pure virtual]
```

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.

Returns

file offset in bytes.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

10.90.2.10 Truncate()

```
int VSIVirtualHandle::Truncate (
            vsi_l_offset nNewSize ) [virtual]
```

Truncate/expand the file to the specified size.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

Parameters

nNewSize	new size in bytes.

Returns

0 on success

Since

GDAL 1.9.0

10.90.2.11 Write()

```
VSIVirtualHandle::Write (
            const void * pBuffer,
            size_t nSize,
            size_t nCount )  [pure virtual]
```

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters

pBuffer	the buffer from which the data should be written (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.

Returns

number of objects successfully written.

Implemented in VSISparseFileHandle, and VSISubFileHandle.

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

- cpl_vsi_virtual.h
- cpl_vsil.cpp

10.91 zip_fileinfo Struct Reference

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

- cpl_minizip_zip.h

10.92 zip_internal Struct Reference

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

- cpl_minizip_zip.cpp

10.93 zlib_filefunc_def_s Struct Reference

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

- cpl_minizip_ioapi.h

Chapter 11

File Documentation

11.1 cpl_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

Classes

- struct CPLSharedFileInfo

Macros

- #define CPLFree VSIFree

Typedefs

- typedef const char *(* CPLFileFinder) (const char *, const char *)

Functions

- CPL_C_START const char CPL_DLL *CPL_STDCALL CPLGetConfigOption (const char *, const char *) CPL_WARN_UNUSED_RESULT
- const char CPL_DLL *CPL_STDCALL CPLGetThreadLocalConfigOption (const char *, const char *) CPL_WARN_UNUSED_RESULT
- void CPL_DLL CPL_STDCALL CPLSetConfigOption (const char *, const char *)
- void CPL_DLL CPL_STDCALL CPLSetThreadLocalConfigOption (const char *pszKey, const char *pszValue)
- char CPL_DLL ** CPLGetConfigOptions (void)
- void CPL_DLL CPLSetConfigOptions (const char *const *papszConfigOptions)
- char CPL_DLL ** CPLGetThreadLocalConfigOptions (void)
- void CPL_DLL CPLSetThreadLocalConfigOptions (const char *const *papszConfigOptions)
- void CPL_DLL * CPLMalloc (size_t) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * CPLCalloc (size_t, size_t) CPL_WARN_UNUSED_RESULT

- void CPL_DLL * CPLRealloc (void *, size_t) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * CPLStrdup (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- char CPL_DLL * CPLStrlwr (char *)
- char CPL_DLL * CPLFGets (char *, int, FILE *)
- const char CPL_DLL * CPLReadLine (FILE *)
- const char CPL_DLL * CPLReadLineL (VSILFILE *)
- double CPL_DLL CPLAtof (const char *)
- double CPL_DLL CPLAtofDelim (const char *, char)
- double CPL_DLL CPLStrtod (const char *, char **)
- double CPL_DLL CPLStrtodDelim (const char *, char **, char)
- float CPL_DLL CPLStrtof (const char *, char **)
- float CPL_DLL CPLStrtofDelim (const char *, char **, char)
- double CPL_DLL CPLAtofM (const char *)
- char CPL_DLL * CPLScanString (const char *, int, int, int)
- double CPL_DLL CPLScanDouble (const char *, int)
- long CPL_DLL CPLScanLong (const char *, int)
- unsigned long CPL_DLL CPLScanULong (const char *, int)
- GUIntBig CPL_DLL CPLScanUIntBig (const char *, int)
- GIntBig CPL_DLL CPLAtoGIntBig (const char *pszString)
- GIntBig CPL_DLL CPLAtoGIntBigEx (const char *pszString, int bWarn, int *pbOverflow)
- void CPL_DLL * CPLScanPointer (const char *, int)
- int CPL_DLL CPLPrintString (char *, const char *, int)
- int CPL_DLL CPLPrintStringFill (char *, const char *, int)
- int CPL_DLL CPLPrintInt32 (char *, GInt32, int)
- int CPL_DLL CPLPrintUIntBig (char *, GUIntBig, int)
- int CPL_DLL CPLPrintTime (char *, int, const char *, const struct tm *, const char *)
- int CPL_DLL CPLPrintPointer (char *, void *, int)
- void CPL_DLL * CPLGetSymbol (const char *, const char *)
- const char CPL_DLL * CPLGetPath (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLGetDirname (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLGetFilename (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLGetBasename (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLGetExtension (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- char CPL_DLL * CPLGetCurrentDir (void)
- const char CPL_DLL * CPLFormFilename (const char *pszPath, const char *pszBasename, const char *pszExtension) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLFormCIFilename (const char *pszPath, const char *pszBasename, const char *pszExtension) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLResetExtension (const char *, const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLProjectRelativeFilename (const char *pszProjectDir, const char *pszSecondary↩ Filename) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- int CPL_DLL CPLIsFilenameRelative (const char *pszFilename)
- const char CPL_DLL * CPLExtractRelativePath (const char *, const char *, int *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLCleanTrailingSlash (const char *) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- char CPL_DLL ** CPLCorrespondingPaths (const char *pszOldFilename, const char *pszNewFilename, char **papszFileList) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CPLCheckForFile (char *pszFilename, char **papszSiblingList)
- const char CPL_DLL * CPLGenerateTempFilename (const char *pszStem) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLExpandTilde (const char *pszFilename) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- const char CPL_DLL * CPLGetHomeDir (void) CPL_WARN_UNUSED_RESULT
- const char CPL_DLL * CPLFindFile (const char *pszClass, const char *pszBasename)

- const char CPL_DLL * CPLDefaultFindFile (const char *pszClass, const char *pszBasename)
- void CPL_DLL CPLPushFileFinder (CPLFileFinder pfnFinder)
- CPLFileFinder CPL_DLL CPLPopFileFinder (void)
- void CPL_DLL CPLPushFinderLocation (const char *)
- void CPL_DLL CPLPopFinderLocation (void)
- void CPL_DLL CPLFinderClean (void)
- int CPL_DLL CPLStat (const char *, VSIStatBuf *) CPL_WARN_UNUSED_RESULT
- FILE CPL_DLL * CPLOpenShared (const char *, const char *, int)
- void CPL_DLL CPLCloseShared (FILE *)
- CPLSharedFileInfo CPL_DLL * CPLGetSharedList (int *)
- void CPL_DLL CPLDumpSharedList (FILE *)
- double CPL_DLL CPLDMSToDec (const char *is)
- const char CPL_DLL * CPLDecToDMS (double dfAngle, const char *pszAxis, int nPrecision)
- double CPL_DLL CPLPackedDMSToDec (double)
- double CPL_DLL CPLDecToPackedDMS (double dfDec)
- void CPL_DLL CPLStringToComplex (const char *pszString, double *pdfReal, double *pdfImag)
- int CPL_DLL CPLUnlinkTree (const char *)
- int CPL_DLL CPLCopyFile (const char *pszNewPath, const char *pszOldPath)
- int CPL_DLL CPLCopyTree (const char *pszNewPath, const char *pszOldPath)
- int CPL_DLL CPLMoveFile (const char *pszNewPath, const char *pszOldPath)
- int CPL_DLL CPLSymlink (const char *pszOldPath, const char *pszNewPath, CSLConstList papszOptions)
- void CPL_DLL * CPLCreateZip (const char *pszZipFilename, char **papszOptions)
- CPLErr CPL_DLL CPLCreateFileInZip (void *hZip, const char *pszFilename, char **papszOptions)
- CPLErr CPL_DLL CPLWriteFileInZip (void *hZip, const void *pBuffer, int nBufferSize)
- CPLErr CPL_DLL CPLCloseFileInZip (void *hZip)
- CPLErr CPL_DLL CPLCloseZip (void *hZip)
- void CPL_DLL * CPLZLibInflate (const void *ptr, size_t nBytes, void *outptr, size_t nOutAvailableBytes, size_t *pnOutBytes)

  Uncompress a buffer compressed with ZLib DEFLATE compression.
- char * CPLsetlocale (int category, const char *locale)
- int CPL_DLL CPLIsPowerOfTwo (unsigned int i)

11.1.1 Detailed Description

Various convenience functions for CPL.

11.1.2 Macro Definition Documentation

11.1.2.1 CPLFree

```
#define CPLFree VSIFree
```

Alias of VSIFree()

11.1.3 Typedef Documentation

11.1.3.1 CPLFileFinder

```
typedef const char*(* CPLFileFinder) (const char *, const char *)
```

Callback for CPLPushFileFinder

11.1.4 Function Documentation

11.1.4.1 CPLAtof()

```
double CPL_DLL 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() 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 the user works in a localized environment and the user's input will be done according to 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(), 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

nptr	Pointer to string to convert.

Returns

Converted value, if any.

11.1.4.2 CPLAtofDelim()

```
double CPL_DLL 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() function.

Parameters

nptr	Pointer to string to convert.
point	Decimal delimiter.

Returns

Converted value, if any.

11.1.4.3 CPLAtofM()

```
double CPL_DLL CPLAtofM (
            const char * nptr )
```

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard atof(), but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

Parameters

nptr	The string to convert.

Returns

Converted value, if any. Zero on failure.

11.1.4.4 CPLAtoGIntBig()

```
GIntBig CPL_DLL CPLAtoGIntBig (
            const char * pszString )
```

Convert a string to a 64 bit signed integer.

Parameters

pszString	String containing 64 bit signed integer.

Returns

64 bit signed integer.

Since

GDAL 2.0

11.1.4.5 CPLAtoGIntBigEx()

```
GIntBig CPL_DLL CPLAtoGIntBigEx (
            const char * pszString,
            int bWarn,
            int * pbOverflow )
```

Convert a string to a 64 bit signed integer.

Parameters

pszString	String containing 64 bit signed integer.
bWarn	Issue a warning if an overflow occurs during conversion
pbOverflow	Pointer to an integer to store if an overflow occurred, or NULL

Returns

64 bit signed integer.

Since

GDAL 2.0

11.1.4.6 CPLCalloc()

```
void CPL_DLL* CPLCalloc (
            size_t nCount,
            size_t nSize )
```

Safe version of calloc().

This function is like the C library calloc(), but raises a CE_Fatal error with CPLError() if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSICalloc() to get the memory, so any hooking of VSICalloc() will apply to CPLCalloc() as well. CPLFree() or VSIFree() can be used free memory allocated by CPLCalloc().

Parameters

nCount	number of objects to allocate.
nSize	size (in bytes) of object to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize ∗ nCount is NULL.

11.1.4.7 CPLCheckForFile()

```
int CPL_DLL 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 VSIStatL() is used to test for the files existence, and no case insensitive testing is done.

Parameters

pszFilename	name of file to check for - filename case updated in some cases.
papszSiblingFiles	a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

Returns

TRUE if a match is found, or FALSE if not.

11.1.4.8 CPLCleanTrailingSlash()

```
const char CPL_DLL* CPLCleanTrailingSlash (
            const char * pszPath )
```

Remove trailing forward/backward slash from the path for UNIX/Windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

Parameters

pszPath	the path to be cleaned up

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

11.1.4.9 CPLCloseFileInZip()

```
CPLErr CPL_DLL CPLCloseFileInZip (
            void * hZip )
```

Close current file inside ZIP file

11.1.4.10 CPLCloseShared()

```
void CPL_DLL CPLCloseShared (
            FILE * fp )
```

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A CPLError() is issued if the file is not in the shared file list.

Parameters

fp	file handle from CPLOpenShared() to deaccess.

11.1.4.11 CPLCloseZip()

```
CPLErr CPL_DLL CPLCloseZip (
            void * hZip )
```

Close ZIP file

11.1.4.12 CPLCopyFile()

```
int CPL_DLL CPLCopyFile (
            const char * pszNewPath,
            const char * pszOldPath )
```

Copy a file

11.1.4.13 CPLCopyTree()

```
int CPL_DLL CPLCopyTree (
            const char * pszNewPath,
            const char * pszOldPath )
```

Recursively copy a tree

11.1.4.14 CPLCorrespondingPaths()

```
char CPL_DLL** 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 correspondence assumes there are two possibly kinds of renaming going on. A change of path, and a change of filename stem.

If a consistent renaming cannot be established for all the files this function will return indicating an error.

The returned file list becomes owned by the caller and should be destroyed with CSLDestroy().

Parameters

pszOldFilename	path to old prototype file.
pszNewFilename	path to new prototype file.
papszFileList	list of other files associated with pszOldFilename to rename similarly.

Returns

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

11.1.4.15 CPLCreateFileInZip()

```
CPLErr CPL_DLL CPLCreateFileInZip (
            void * hZip,
            const char * pszFilename,
            char ** papszOptions )
```

Create a file in a ZIP file

11.1.4.16 CPLCreateZip()

```
void CPL_DLL* CPLCreateZip (
            const char * pszZipFilename,
            char ** papszOptions )
```

Create ZIP file

11.1.4.17 CPLDecToDMS()

```
const char CPL_DLL* CPLDecToDMS (
            double dfAngle,
            const char * pszAxis,
            int nPrecision )
```

Translate a decimal degrees value to a DMS string with hemisphere.

11.1.4.18 CPLDecToPackedDMS()

```
double CPL_DLL 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().

Parameters

dfDec	Angle in decimal degrees.

Returns

Angle in packed DMS format.

11.1.4.19 CPLDefaultFindFile()

```
const char CPL_DLL* CPLDefaultFindFile (
            const char * ,
            const char * pszBasename )
```

CPLDefaultFindFile

11.1.4.20 CPLDMSToDec()

```
double CPL_DLL CPLDMSToDec (
            const char * is )
```

CPLDMSToDec

11.1.4.21 CPLDumpSharedList()

```
void CPL_DLL 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() call.

Parameters

fp	File handle to write to.

11.1.4.22 CPLExpandTilde()

```
const char CPL_DLL* CPLExpandTilde (
            const char * pszFilename )
```

Expands ~/ at start of filename.

Assumes that the HOME configuration option is defined.

Parameters

pszFilename	filename potentially starting with ~/

Returns

an expanded filename.

Since

GDAL 2.2

11.1.4.23 CPLExtractRelativePath()

```
const char CPL_DLL* CPLExtractRelativePath (
            const char * pszBaseDir,
            const char * pszTarget,
            int * pbGotRelative )
```

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

Parameters

pszBaseDir	the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relativizing.
pszTarget	the filename to be changed to be relative to pszBaseDir.
pbGotRelative	Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

Returns

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

11.1.4.24 CPLFGets()

```
char CPL_DLL* 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(): single '\r' and '\n' and '\r\n' combination.

Parameters

pszBuffer	pointer to the targeting character buffer.
nBufferSize	maximum size of the string to read (not including terminating '\0').
fp	file pointer to read from.

Returns

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

11.1.4.25 CPLFinderClean()

```
void CPL_DLL CPLFinderClean (
            void  )
```

CPLFinderClean

11.1.4.26 CPLFindFile()

```
const char CPL_DLL* CPLFindFile (
            const char * pszClass,
            const char * pszBasename )
```

CPLFindFile

11.1.4.27 CPLFormCIFilename()

```
const char CPL_DLL* 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() on case insensitive file systems (like Windows).

Parameters

pszPath	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
pszBasename	file basename. May optionally have path and/or extension. May not be NULL.
pszExtension	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

11.1.4.28 CPLFormFilename()

```
const char CPL_DLL* CPLFormFilename (
            const char * pszPath,
            const char * pszBasename,
            const char * pszExtension )
```

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz", "def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL,"def", NULL ) == "def"
CPLFormFilename(NULL, "abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/", "def.dat", NULL ) == "/abc/xyz/def.dat"
```

Parameters

pszPath	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
pszBasename	file basename. May optionally have path and/or extension. Must *NOT* be NULL.
pszExtension	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

11.1.4.29 CPLGenerateTempFilename()

```
const char CPL_DLL* CPLGenerateTempFilename (
            const char * pszStem )
```

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL_TMPDIR configuration option.

Parameters

pszStem	if non-NULL this will be part of the filename.

Returns

a filename which is valid till the next CPL call in this thread.

11.1.4.30 CPLGetBasename()

```
const char CPL_DLL* CPLGetBasename (
            const char * pszFullFilename )
```

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

Parameters

pszFullFilename	the full filename potentially including a path.

Returns

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

11.1.4.31 CPLGetConfigOption()

```
CPL_C_START const char CPL_DLL* CPL_STDCALL 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(), or CPLSetThreadLocalConfigOption() of the same thread. If the given option was no defined with CPLSetConfigOption(), it tries to find it in environment variables.

Note: the string returned by CPLGetConfigOption() might be short-lived, and in particular it will become invalid after a call to CPLSetConfigOption() with the same key.

To override temporary a potentially existing option with a new value, you can use the following snippet :

```
// backup old value
const char* pszOldValTmp = CPLGetConfigOption(pszKey, NULL);
char* pszOldVal = pszOldValTmp ? CPLStrdup(pszOldValTmp) : NULL;
// override with new value
CPLSetConfigOption(pszKey, pszNewVal);
// do something useful
// restore old value
CPLSetConfigOption(pszKey, pszOldVal);
CPLFree(pszOldVal);
```

Parameters

pszKey	the key of the option to retrieve
pszDefault	a default value if the key does not match existing defined options (may be NULL)

Returns

the value associated to the key, or the default value if not found

See also

CPLSetConfigOption(), http://trac.osgeo.org/gdal/wiki/ConfigOptions

11.1.4.32 CPLGetConfigOptions()

```
char CPL_DLL** CPLGetConfigOptions (
            void  )
```

Return the list of configuration options as KEY=VALUE pairs.

The list is the one set through the CPLSetConfigOption() API.

Options that through environment variables or with CPLSetThreadLocalConfigOption() will *not* be listed.

Returns

a copy of the list, to be freed with CSLDestroy().

Since

GDAL 2.2

11.1.4.33 CPLGetCurrentDir()

```
char CPL_DLL* 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.

11.1.4.34 CPLGetDirname()

```
const char CPL_DLL* 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().

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters

pszFilename	the filename potentially including a path.

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

11.1.4.35 CPLGetExtension()

```
const char CPL_DLL* 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

pszFullFilename	the full filename potentially including a path.

Returns

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

11.1.4.36 CPLGetFilename()

```
const char CPL_DLL* CPLGetFilename (
            const char * pszFullFilename )
```

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

Parameters

pszFullFilename	the full filename potentially including a path.

Returns

just the non-directory portion of the path (points back into original string).

11.1.4.37 CPLGetHomeDir()

```
const char CPL_DLL* CPLGetHomeDir (
            void  )
```

Return the path to the home directory

That is the value of the USERPROFILE environment variable on Windows, or HOME on other platforms.

Returns

the home directory, or NULL.

Since

GDAL 2.3

11.1.4.38 CPLGetPath()

```
const char CPL_DLL* CPLGetPath (
            const char * pszFilename )
```

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

Parameters

pszFilename	the filename potentially including a path.

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

11.1.4.39 CPLGetSharedList()

```
CPLSharedFileInfo CPL_DLL* CPLGetSharedList (
            int * pnCount )
```

Fetch list of open shared files.

Parameters

pnCount	place to put the count of entries.

Returns

the pointer to the first in the array of shared file info structures.

11.1.4.40 CPLGetSymbol()

```
void CPL_DLL* 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() or not are not currently well defined, and will have to be resolved in the future.

Currently CPLGetSymbol() 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

pszLibrary	the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.
pszSymbolName	the name of the function to fetch a pointer to.

Returns

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

11.1.4.41 CPLGetThreadLocalConfigOption()

```
const char CPL_DLL* CPL_STDCALL CPLGetThreadLocalConfigOption (
            const char * pszKey,
            const char * pszDefault )
```

Same as CPLGetConfigOption() but only with options set with CPLSetThreadLocalConfigOption()

11.1.4.42 CPLGetThreadLocalConfigOptions()

```
char CPL_DLL** CPLGetThreadLocalConfigOptions (
            void  )
```

Return the list of thread local configuration options as KEY=VALUE pairs.

Options that through environment variables or with CPLSetConfigOption() will *not* be listed.

Returns

a copy of the list, to be freed with CSLDestroy().

Since

GDAL 2.2

11.1.4.43 CPLIsFilenameRelative()

```
int CPL_DLL CPLIsFilenameRelative (
            const char * pszFilename )
```

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

Parameters

pszFilename	the filename with path to test.

Returns

TRUE if the filename is relative or FALSE if it is absolute.

11.1.4.44 CPLIsPowerOfTwo()

```
int CPL_DLL CPLIsPowerOfTwo (
            unsigned int i )
```

CPLIsPowerOfTwo()

Parameters

i	- tested number

Returns

TRUE if i is power of two otherwise return FALSE

11.1.4.45 CPLMalloc()

```
void CPL_DLL* CPLMalloc (
            size_t nSize )
```

Safe version of malloc().

This function is like the C library malloc(), but raises a CE_Fatal error with CPLError() 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() as well. CPLFree() or VSIFree() can be used free memory allocated by CPLMalloc().

Parameters

nSize	size (in bytes) of memory block to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize is zero.

11.1.4.46 CPLMoveFile()

```
int CPL_DLL CPLMoveFile (
            const char * pszNewPath,
            const char * pszOldPath )
```

Move a file

11.1.4.47 CPLOpenShared()

```
FILE CPL_DLL* CPLOpenShared (
            const char * pszFilename,
            const char * pszAccess,
            int bLargeIn )
```

Open a shared file handle.

Some operating systems have limits on the number of file handles that can be open at one time. This function attempts to maintain a registry of already open file handles, and reuse existing ones if the same file is requested by another part of the application.

Note that access is only shared for access types "r", "rb", "r+" and "rb+". All others will just result in direct VSIOpen() calls. Keep in mind that a file is only reused if the file name is exactly the same. Different names referring to the same file will result in different handles.

The VSIFOpen() or VSIFOpenL() function is used to actually open the file, when an existing file handle can't be shared.

Parameters

pszFilename	the name of the file to open.
pszAccess	the normal fopen()/VSIFOpen() style access string.
bLargeIn	If TRUE VSIFOpenL() (for large files) will be used instead of VSIFOpen().

Returns

a file handle or NULL if opening fails.

11.1.4.48 CPLPackedDMSToDec()

```
double CPL_DLL 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: angle = 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 = angle/1000000 (fractional portion truncated)
3. The minutes are separated out: min = (angle - deg ∗ 1000000) / 1000 (fractional portion truncated)
4. The seconds are then computed: sec = angle - 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

dfPacked	Angle in packed DMS format.

Returns

Angle in decimal degrees.

11.1.4.49 CPLPopFileFinder()

```
CPLFileFinder CPL_DLL CPLPopFileFinder (
            void  )
```

CPLPopFileFinder

11.1.4.50 CPLPopFinderLocation()

```
void CPL_DLL CPLPopFinderLocation (
            void  )
```

CPLPopFinderLocation

11.1.4.51 CPLPrintInt32()

```
int CPL_DLL CPLPrintInt32 (
            char * pszBuffer,
            GInt32 iValue,
            int nMaxLen )
```

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

11.1.4.52 CPLPrintPointer()

```
int CPL_DLL CPLPrintPointer (
            char * pszBuffer,
            void * pValue,
            int nMaxLen )
```

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
pValue	Pointer to ASCII encode.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

11.1.4.53 CPLPrintString()

```
int CPL_DLL CPLPrintString (
            char * pszDest,
            const char * pszSrc,
            int nMaxLen )
```

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest.

Parameters

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

11.1.4.54 CPLPrintStringFill()

```
int CPL_DLL CPLPrintStringFill (
            char * pszDest,
            const char * pszSrc,
            int nMaxLen )
```

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

Parameters

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

11.1.4.55 CPLPrintTime()

```
int CPL_DLL CPLPrintTime (
            char * pszBuffer,
            int nMaxLen,
            const char * pszFormat,
            const struct tm * poBrokenTime,
            const char * pszLocale )
```

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Parameters

pszFormat	Controls the output format. Options are the same as for strftime(3) function.
poBrokenTime	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.
pszLocale	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be adjusted differently from the C/POSIX defaults. To solve these problems this option was introduced.

Returns

Number of characters printed.

11.1.4.56 CPLPrintUIntBig()

```
int CPL_DLL CPLPrintUIntBig (
            char * pszBuffer,
            GUIntBig iValue,
            int nMaxLen )
```

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

11.1.4.57 CPLProjectRelativeFilename()

```
const char CPL_DLL* CPLProjectRelativeFilename (
            const char * pszProjectDir,
            const char * pszSecondaryFilename )
```

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def", "tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def", "/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def", "../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\WIN", "abc.gif") == "C:\WIN\abc.gif"
```

Parameters

pszProjectDir	the directory relative to which the secondary files path should be interpreted.
pszSecondaryFilename	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.

11.1.4.58 CPLPushFileFinder()

```
void CPL_DLL CPLPushFileFinder (
            CPLFileFinder pfnFinder )
```

CPLPushFileFinder

11.1.4.59 CPLPushFinderLocation()

```
void CPL_DLL CPLPushFinderLocation (
            const char * pszLocation )
```

CPLPushFinderLocation

11.1.4.60 CPLReadLine()

```
const char CPL_DLL* 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 its value past the next call to CPLReadLine().

Note that CPLReadLine() uses VSIFGets(), so any hooking of VSI file services should apply to CPLReadLine() as well.

CPLReadLine() maintains an internal buffer, which will appear as a single block memory leak in some circumstances. CPLReadLine() may be called with a NULL FILE * at any time to free this working buffer.

Parameters

fp	file pointer opened with VSIFOpen().

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

11.1.4.61 CPLReadLineL()

```
const char CPL_DLL* CPLReadLineL (
            VSILFILE * fp )
```

Simplified line reading from text file.

Similar to CPLReadLine(), but reading from a large file API handle.

Parameters

fp	file pointer opened with VSIFOpenL().

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

11.1.4.62 CPLRealloc()

```
void CPL_DLL* 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() if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIRealloc() to get the memory, so any hooking of VSIRealloc() will apply to CPLRealloc() as well. CPLFree() or VSIFree() can be used free memory allocated by CPLRealloc().

It is also safe to pass NULL in as the existing memory block for CPLRealloc(), in which case it uses VSIMalloc() to allocate a new block.

Parameters

pData	existing memory block which should be copied to the new block.
nNewSize	new size (in bytes) of memory block to allocate.

Returns

pointer to allocated memory, only NULL if nNewSize is zero.

11.1.4.63 CPLResetExtension()

```
const char CPL_DLL* CPLResetExtension (
            const char * pszPath,
            const char * pszExt )
```

Replace the extension with the provided one.

Parameters

pszPath	the input path, this string is not altered.
pszExt	the new extension to apply to the given path.

Returns

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

11.1.4.64 CPLScanDouble()

```
double CPL_DLL 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() to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Double value, converted from its ASCII form.

11.1.4.65 CPLScanLong()

```
long CPL_DLL CPLScanLong (
            const char * pszString,
            int nMaxLength )
```

Scan up to a maximum number of characters from a string and convert the result to a long.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Long value, converted from its ASCII form.

11.1.4.66 CPLScanPointer()

```
void CPL_DLL* CPLScanPointer (
            const char * pszString,
            int nMaxLength )
```

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

pointer value, converted from its ASCII form.

11.1.4.67 CPLScanString()

```
char CPL_DLL* CPLScanString (
            const char * pszString,
            int nMaxLength,
            int bTrimSpaces,
            int bNormalize )
```

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to read. Less characters will be read if a null character is encountered.
bTrimSpaces	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
bNormalize	If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

Returns

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

11.1.4.68 CPLScanUIntBig()

```
GUIntBig CPL_DLL CPLScanUIntBig (
            const char * pszString,
            int nMaxLength )
```

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

GUIntBig value, converted from its ASCII form.

11.1.4.69 CPLScanULong()

```
unsigned long CPL_DLL CPLScanULong (
            const char * pszString,
            int nMaxLength )
```

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

Parameters

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Unsigned long value, converted from its ASCII form.

11.1.4.70 CPLSetConfigOption()

```
void CPL_DLL CPL_STDCALL 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() method.

This mechanism is similar to environment variables, but options set with CPLSetConfigOption() overrides, for CPLGetConfigOption() point of view, values defined in the environment.

If CPLSetConfigOption() 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

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by CPLSetConfigOption()).

Parameters

pszKey	the key of the option
pszValue	the value of the option, or NULL to clear a setting.

See also

http://trac.osgeo.org/gdal/wiki/ConfigOptions

11.1.4.71 CPLSetConfigOptions()

```
void CPL_DLL CPLSetConfigOptions (
            const char *const * papszConfigOptions )
```

Replace the full list of configuration options with the passed list of KEY=VALUE pairs.

This has the same effect of clearing the existing list, and setting individually each pair with the CPLSetConfigOption() API.

This does not affect options set through environment variables or with CPLSetThreadLocalConfigOption().

The passed list is copied by the function.

Parameters

papszConfigOptions	the new list (or NULL).

Since

GDAL 2.2

11.1.4.72 CPLsetlocale()

```
char* CPLsetlocale (
            int category,
            const char * locale )
```

Prevents parallel executions of setlocale().

Calling setlocale() concurrently from two or more threads is a potential data race. A mutex is used to provide a critical region so that only one thread at a time can be executing setlocale().

The return should not be freed, and copied quickly as it may be invalidated by a following next call to CPLsetlocale().

Parameters

category	See your compiler's documentation on setlocale.
locale	See your compiler's documentation on setlocale.

Returns

See your compiler's documentation on setlocale.

11.1.4.73 CPLSetThreadLocalConfigOption()

```
void CPL_DLL CPL_STDCALL 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() method.

This function sets the configuration option that only applies in the current thread, as opposed to CPLSetConfigOption() which sets an option that applies on all threads. CPLSetThreadLocalConfigOption() will override the effect of CPLSetConfigOption) for the current thread.

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable or a value set through CPLSetConfigOption(); it will just unset a value previously set by CPLSetThreadLocalConfigOption()).

Parameters

pszKey	the key of the option
pszValue	the value of the option, or NULL to clear a setting.

11.1.4.74 CPLSetThreadLocalConfigOptions()

```
void CPL_DLL CPLSetThreadLocalConfigOptions (
            const char *const * papszConfigOptions )
```

Replace the full list of thread local configuration options with the passed list of KEY=VALUE pairs.

This has the same effect of clearing the existing list, and setting individually each pair with the CPLSetThreadLocalConfigOption() API.

This does not affect options set through environment variables or with CPLSetConfigOption().

The passed list is copied by the function.

Parameters

papszConfigOptions	the new list (or NULL).

Since

GDAL 2.2

11.1.4.75 CPLStat()

```
int CPL_DLL CPLStat (
            const char * pszPath,
            VSIStatBuf * psStatBuf )
```

Same as VSIStat() except it works on "C:" as if it were "C:\".

11.1.4.76 CPLStrdup()

```
char CPL_DLL* 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() instead of returning NULL. Memory allocated with CPLStrdup() can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into CPLStrdup(). CPLStrdup() will allocate and return a zero length string (as opposed to a NULL string).

Parameters

pszString	input string to be duplicated. May be NULL.

Returns

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

11.1.4.77 CPLStringToComplex()

```
void CPL_DLL CPLStringToComplex (
            const char * pszString,
            double * pdfReal,
            double * pdfImag )
```

Fetch the real and imaginary part of a serialized complex number

11.1.4.78 CPLStrlwr()

```
char CPL_DLL* CPLStrlwr (
            char * pszString )
```

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

Parameters

pszString	input string to be converted.

Returns

pointer to the same string, pszString.

11.1.4.79 CPLStrtod()

```
double CPL_DLL 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() function if you want to specify custom delimiter. Also see notes for CPLAtof() function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns

Converted value, if any.

11.1.4.80 CPLStrtodDelim()

```
double CPL_DLL 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() function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
point	Decimal delimiter.

Returns

Converted value, if any.

11.1.4.81 CPLStrtof()

```
float CPL_DLL 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() function if you want to specify custom delimiter. Also see notes for CPLAtof() function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns

Converted value, if any.

11.1.4.82 CPLStrtofDelim()

```
float CPL_DLL 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() function.

Parameters

nptr	Pointer to string to convert.
endptr	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
point	Decimal delimiter.

Returns

Converted value, if any.

11.1.4.83 CPLSymlink()

```
int CPL_DLL CPLSymlink (
            const char * pszOldPath,
            const char * pszNewPath,
            CSLConstList )
```

Create a symbolic link

11.1.4.84 CPLUnlinkTree()

```
int CPL_DLL CPLUnlinkTree (
            const char * pszPath )
```

Recursively unlink a directory.

Returns

0 on successful completion, -1 if function fails.

11.1.4.85 CPLWriteFileInZip()

```
CPLErr CPL_DLL CPLWriteFileInZip (
            void * hZip,
            const void * pBuffer,
            int nBufferSize )
```

Write in current file inside a ZIP file

11.1.4.86 CPLZLibInflate()

```
void CPL_DLL* CPLZLibInflate (
            const void * ptr,
            size_t nBytes,
            void * outptr,
            size_t nOutAvailableBytes,
            size_t * pnOutBytes )
```

Uncompress a buffer compressed with ZLib DEFLATE compression.

Parameters

ptr	input buffer.
nBytes	size of input buffer in bytes.
outptr	output buffer, or NULL to let the function allocate it.
nOutAvailableBytes	size of output buffer if provided, or ignored.
pnOutBytes	pointer to a size_t, where to store the size of the output buffer.

Returns

the output buffer (to be freed with VSIFree() if not provided) or NULL in case of error.

Since

GDAL 1.10.0

11.2 cpl_error.h File Reference

```
#include "cpl_port.h"
#include <stdarg.h>
#include <stddef.h>
```

Macros

- #define CPLE_None 0
- #define CPLE_AppDefined 1
- #define CPLE_OutOfMemory 2
- #define CPLE_FileIO 3
- #define CPLE_OpenFailed 4
- #define CPLE_IllegalArg 5
- #define CPLE_NotSupported 6
- #define CPLE_AssertionFailed 7
- #define CPLE_NoWriteAccess 8
- #define CPLE_UserInterrupt 9
- #define CPLE_ObjectNull 10
- #define CPLE_HttpResponse 11
- #define CPLE_AWSBucketNotFound 12
- #define CPLE_AWSObjectNotFound 13
- #define CPLE_AWSAccessDenied 14
- #define CPLE_AWSInvalidCredentials 15
- #define CPLE_AWSSignatureDoesNotMatch 16
- #define CPLAssert(expr)
- #define VALIDATE_POINTER0(ptr, func)
- #define VALIDATE_POINTER1(ptr, func, rc)

Typedefs

- typedef int CPLErrorNum
- typedef void(CPL_STDCALL * CPLErrorHandler) (CPLErr, CPLErrorNum, const char *)

Enumerations

- enum CPLErr

Functions

- void CPL_DLL void CPL_DLL CPLErrorV (CPLErr, CPLErrorNum, const char *, va_list)
- void CPL_DLL CPLEmergencyError (const char *) CPL_NO_RETURN
- void CPL_DLL CPL_STDCALL CPLErrorReset (void)
- CPLErrorNum CPL_DLL CPL_STDCALL CPLGetLastErrorNo (void)
- CPLErr CPL_DLL CPL_STDCALL CPLGetLastErrorType (void)
- const char CPL_DLL *CPL_STDCALL CPLGetLastErrorMsg (void)
- GUInt32 CPL_DLL CPL_STDCALL CPLGetErrorCounter (void)
- void CPL_DLL *CPL_STDCALL CPLGetErrorHandlerUserData (void)
- void CPL_DLL CPLErrorSetState (CPLErr eErrClass, CPLErrorNum err_no, const char *pszMsg)

- void CPL_DLL CPL_STDCALL CPLLoggingErrorHandler (CPLErr, CPLErrorNum, const char ∗)
- void CPL_DLL CPL_STDCALL CPLDefaultErrorHandler (CPLErr, CPLErrorNum, const char ∗)
- void CPL_DLL CPL_STDCALL CPLQuietErrorHandler (CPLErr, CPLErrorNum, const char ∗)
- void CPLTurnFailureIntoWarning (int bOn)
- CPLErrorHandler CPL_DLL CPL_STDCALL CPLSetErrorHandler (CPLErrorHandler)
- CPLErrorHandler CPL_DLL CPL_STDCALL CPLSetErrorHandlerEx (CPLErrorHandler, void ∗)
- void CPL_DLL CPL_STDCALL CPLPushErrorHandler (CPLErrorHandler)
- void CPL_DLL CPL_STDCALL CPLPushErrorHandlerEx (CPLErrorHandler, void ∗)
- void CPL_DLL CPL_STDCALL CPLSetCurrentErrorHandlerCatchDebug (int bCatchDebug)
- void CPL_DLL CPL_STDCALL CPLPopErrorHandler (void)
- void CPL_DLL CPL_STDCALL void CPL_DLL CPL_STDCALL _CPLAssert (const char ∗, const char ∗, int) CPL_NO_RETURN

11.2.1 Detailed Description

CPL error handling services.

11.2.2 Macro Definition Documentation

11.2.2.1 CPLAssert

```
#define CPLAssert(
            expr )
```

Assert on an expression. Only enabled in DEBUG mode

11.2.2.2 CPLE_AppDefined

```
#define CPLE_AppDefined 1
```

Application defined error

11.2.2.3 CPLE_AssertionFailed

```
#define CPLE_AssertionFailed 7
```

Assertion failed

11.2.2.4 CPLE_AWSAccessDenied

```
#define CPLE_AWSAccessDenied 14
```

AWSAccessDenied

11.2.2.5 CPLE_AWSBucketNotFound

```
#define CPLE_AWSBucketNotFound 12
```

AWSBucketNotFound

11.2.2.6 CPLE_AWSInvalidCredentials

```
#define CPLE_AWSInvalidCredentials 15
```

AWSInvalidCredentials

11.2.2.7 CPLE_AWSObjectNotFound

```
#define CPLE_AWSObjectNotFound 13
```

AWSObjectNotFound

11.2.2.8 CPLE_AWSSignatureDoesNotMatch

```
#define CPLE_AWSSignatureDoesNotMatch 16
```

AWSSignatureDoesNotMatch

11.2.2.9 CPLE_FileIO

```
#define CPLE_FileIO 3
```

File I/O error

11.2.2.10 CPLE_HttpResponse

```
#define CPLE_HttpResponse 11
```

HTTP response

11.2.2.11 CPLE_IllegalArg

```
#define CPLE_IllegalArg 5
```

Illegal argument

11.2.2.12 CPLE_None

```
#define CPLE_None 0
```

No error

11.2.2.13 CPLE_NotSupported

```
#define CPLE_NotSupported 6
```

Not supported

11.2.2.14 CPLE_NoWriteAccess

```
#define CPLE_NoWriteAccess 8
```

No write access

11.2.2.15 CPLE_ObjectNull

```
#define CPLE_ObjectNull 10
```

NULL object

11.2.2.16 CPLE_OpenFailed

```
#define CPLE_OpenFailed 4
```

Open failed

11.2.2.17 CPLE_OutOfMemory

```
#define CPLE_OutOfMemory 2
```

Out of memory error

11.2.2.18 CPLE_UserInterrupt

```
#define CPLE_UserInterrupt 9
```

User interrupted

11.2.2.19 VALIDATE_POINTER0

```
#define VALIDATE_POINTER0(
               ptr,
               func )
```

Value:

```
do { if( CPL_NULLPTR == ptr ) \
      { \
        CPLErr const ret = VALIDATE_POINTER_ERR; \
        CPLError( ret, CPLE_ObjectNull, \
           "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
         return; }} while(0)
```

Validate that a pointer is not NULL

11.2.2.20 VALIDATE_POINTER1

```
#define VALIDATE_POINTER1(
                ptr,
                func,
                rc )
```

Value:

```
do { if( CPL_NULLPTR == ptr ) \
      { \
          CPLErr const ret = VALIDATE_POINTER_ERR; \
          CPLError( ret, CPLE_ObjectNull, \
           "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
        return (rc); }} while(0)
```

Validate that a pointer is not NULL, and return rc if it is NULL

11.2.3 Typedef Documentation

11.2.3.1 CPLErrorHandler

```
typedef void(CPL_STDCALL * CPLErrorHandler) (CPLErr, CPLErrorNum, const char *)
```

Callback for a custom error handler

11.2.3.2 CPLErrorNum

```
typedef int CPLErrorNum
```

Error number

11.2.4 Enumeration Type Documentation

11.2.4.1 CPLErr

```
enum CPLErr
```

Error category

11.2.5 Function Documentation

11.2.5.1 _CPLAssert()

```
void CPL_DLL CPL_STDCALL void CPL_DLL CPL_STDCALL _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() only if the condition fails. _CPLAssert() will generate a CE_Fatal error call to CPLError(), 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() directly.

11.2.5.2 CPLDefaultErrorHandler()

```
void CPL_DLL CPL_STDCALL CPLDefaultErrorHandler (
            CPLErr eErrClass,
            CPLErrorNum nError,
            const char * pszErrorMsg )
```

Default error handler.

11.2.5.3 CPLEmergencyError()

```
void CPL_DLL CPLEmergencyError (
            const char * pszMessage )
```

Fatal error when things are bad.

This function should be called in an emergency situation where it is unlikely that a regular error report would work. This would include in the case of heap exhaustion for even small allocations, or any failure in the process of reporting an error (such as TLS allocations).

This function should never return. After the error message has been reported as best possible, the application will abort() similarly to how CPLError() aborts on CE_Fatal class errors.

Parameters

pszMessage	the error message to report.

11.2.5.4 CPLErrorReset()

```
void CPL_DLL CPL_STDCALL CPLErrorReset (
            void )
```

Erase any traces of previous errors.

This is normally used to ensure that an error which has been recovered from does not appear to be still in play with high level functions.

11.2.5.5 CPLErrorSetState()

```
void CPL_DLL CPLErrorSetState (
            CPLErr eErrClass,
            CPLErrorNum err_no,
            const char * pszMsg )
```

Restore an error state, without emitting an error.

Can be useful if a routine might call CPLErrorReset() and one wants to preserve the previous error state.

Since

GDAL 2.0

11.2.5.6 CPLErrorV()

```
void CPL_DLL void CPL_DLL CPLErrorV (
            CPLErr eErrClass,
            CPLErrorNum err_no,
            const char * fmt,
            va_list args )
```

Same as CPLError() but with a va_list

11.2.5.7 CPLGetErrorCounter()

```
GUInt32 CPL_DLL CPL_STDCALL CPLGetErrorCounter (
            void  )
```

Get the error counter

Fetches the number of errors emitted in the current error context, since the last call to CPLErrorReset()

Returns

the error counter.

Since

GDAL 2.3

11.2.5.8 CPLGetErrorHandlerUserData()

```
void CPL_DLL* CPL_STDCALL CPLGetErrorHandlerUserData (
            void  )
```

Fetch the user data for the error context

Fetches the user data for the current error context. You can set the user data for the error context when you add your handler by issuing CPLSetErrorHandlerEx() and CPLPushErrorHandlerEx(). Note that user data is primarily intended for providing context within error handlers themselves, but they could potentially be abused in other useful ways with the usual caveat emptor understanding.

Returns

the user data pointer for the error context

11.2.5.9 CPLGetLastErrorMsg()

```
const char CPL_DLL* CPL_STDCALL CPLGetLastErrorMsg (
            void  )
```

Get the last error message.

Fetches the last error message posted with CPLError(), that hasn't been cleared by CPLErrorReset(). 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.

11.2.5.10 CPLGetLastErrorNo()

```
CPLErrorNum CPL_DLL CPL_STDCALL CPLGetLastErrorNo (
            void  )
```

Fetch the last error number.

Fetches the last error number posted with CPLError(), that hasn't been cleared by CPLErrorReset(). This is the error number, not the error class.

Returns

the error number of the last error to occur, or CPLE_None (0) if there are no posted errors.

11.2.5.11 CPLGetLastErrorType()

```
CPLErr CPL_DLL CPL_STDCALL CPLGetLastErrorType (
            void  )
```

Fetch the last error type.

Fetches the last error type posted with CPLError(), that hasn't been cleared by CPLErrorReset(). This is the error class, not the error number.

Returns

the error type of the last error to occur, or CE_None (0) if there are no posted errors.

11.2.5.12 CPLLoggingErrorHandler()

```
void CPL_DLL CPL_STDCALL CPLLoggingErrorHandler (
            CPLErr eErrClass,
            CPLErrorNum nError,
            const char * pszErrorMsg )
```

Error handler that logs into the file defined by the CPL_LOG configuration option, or stderr otherwise.

11.2.5.13 CPLPopErrorHandler()

```
void CPL_DLL CPL_STDCALL 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() call. This method has no effect if there are no error handlers on the current threads error handler stack.

11.2.5.14 CPLPushErrorHandler()

```
void CPL_DLL CPL_STDCALL 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().

The CPLSetErrorHandler() docs have further information on how CPLError handlers work.

Parameters

pfnErrorHandlerNew	new error handler function.

11.2.5.15 CPLPushErrorHandlerEx()

```
void CPL_DLL CPL_STDCALL CPLPushErrorHandlerEx (
            CPLErrorHandler pfnErrorHandlerNew,
            void * pUserData )
```

Push a new CPLError handler with user data on the error context.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with CPLPopErrorHandler(). Obtain the user data back by using CPLGetErrorContext().

The CPLSetErrorHandler() docs have further information on how CPLError handlers work.

Parameters

pfnErrorHandlerNew	new error handler function.
pUserData	User data to put on the error context.

11.2.5.16 CPLQuietErrorHandler()

```
void CPL_DLL CPL_STDCALL CPLQuietErrorHandler (
            CPLErr eErrClass,
            CPLErrorNum nError,
            const char * pszErrorMsg )
```

Error handler that does not do anything, except for debug messages.

11.2.5.17 CPLSetCurrentErrorHandlerCatchDebug()

```
void CPL_DLL CPL_STDCALL CPLSetCurrentErrorHandlerCatchDebug (
            int bCatchDebug )
```

Set if the current error handler should intercept debug messages, or if they should be processed by the previous handler.

By default when installing a custom error handler, this one intercepts debug messages. In some cases, this might not be desirable and the user would prefer that the previous installed handler (or the default one if no previous installed handler exists in the stack) deal with it. In which case, this function should be called with bCatchDebug = FALSE.

Parameters

bCatchDebug	FALSE if the current error handler should not intercept debug messages

Since

GDAL 2.1

11.2.5.18 CPLSetErrorHandler()

```
CPLErrorHandler CPL_DLL CPL_STDCALL CPLSetErrorHandler (
            CPLErrorHandler pfnErrorHandlerNew )
```

Install custom error handler.

Allow the library's user to specify an error handler function. A valid error handler is a C function with the following prototype:

```
void MyErrorHandler(CPLErr eErrClass, int err_no, const char *msg)
```

Pass NULL to come back to the default behavior. The default behaviour (CPLDefaultErrorHandler()) is to write the message to stderr.

The msg will be a partially formatted error message not containing the "ERROR %d:" portion emitted by the default handler. Message formatting is handled by CPLError() before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then CPLError() will call abort(). Applications wanting to interrupt this fatal behaviour will have to use longjmp(), or a C++ exception to indirectly exit the function.

Another standard error handler is CPLQuietErrorHandler() which doesn't make any attempt to report the passed error or warning messages but will process debug messages via CPLDefaultErrorHandler.

Note that error handlers set with CPLSetErrorHandler() apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPush↩ ErrorHandler (and not removed with CPLPopErrorHandler) take precedence over the global error handlers set with CPLSetErrorHandler(). Generally speaking CPLSetErrorHandler() would be used to set a desired global error handler, while CPLPushErrorHandler() would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

Parameters

pfnErrorHandlerNew	new error handler function.

Returns

returns the previously installed error handler.

11.2.5.19 CPLSetErrorHandlerEx()

```
CPLErrorHandler CPL_DLL CPL_STDCALL CPLSetErrorHandlerEx (
            CPLErrorHandler pfnErrorHandlerNew,
            void * pUserData )
```

Install custom error handle with user's data. This method is essentially CPLSetErrorHandler with an added pointer to pUserData. The pUserData is not returned in the CPLErrorHandler, however, and must be fetched via CPLGet↩ ErrorHandlerUserData.

Parameters

pfnErrorHandlerNew	new error handler function.
pUserData	User data to carry along with the error context.

Returns

returns the previously installed error handler.

11.2.5.20 CPLTurnFailureIntoWarning()

```
void CPLTurnFailureIntoWarning (
            int bOn )
```

Whether failures should be turned into warnings.

11.3 cpl_hash_set.h File Reference

```
#include "cpl_port.h"
```

Typedefs

- typedef typedefCPL_C_START struct _CPLHashSet CPLHashSet
- typedef unsigned long(∗ CPLHashSetHashFunc) (const void ∗elt)
- typedef int(∗ CPLHashSetEqualFunc) (const void ∗elt1, const void ∗elt2)
- typedef void(∗ CPLHashSetFreeEltFunc) (void ∗elt)
- typedef int(∗ CPLHashSetIterEltFunc) (void ∗elt, void ∗user_data)

Functions

- CPLHashSet CPL_DLL ∗ CPLHashSetNew (CPLHashSetHashFunc fnHashFunc, CPLHashSetEqualFunc fnEqualFunc, CPLHashSetFreeEltFunc fnFreeEltFunc)
- void CPL_DLL CPLHashSetDestroy (CPLHashSet ∗set)
- void CPL_DLL CPLHashSetClear (CPLHashSet ∗set)
- int CPL_DLL CPLHashSetSize (const CPLHashSet ∗set)
- void CPL_DLL CPLHashSetForeach (CPLHashSet ∗set, CPLHashSetIterEltFunc fnIterFunc, void ∗user_↩ data)
- int CPL_DLL CPLHashSetInsert (CPLHashSet ∗set, void ∗elt)
- void CPL_DLL ∗ CPLHashSetLookup (CPLHashSet ∗set, const void ∗elt)
- int CPL_DLL CPLHashSetRemove (CPLHashSet ∗set, const void ∗elt)
- int CPL_DLL CPLHashSetRemoveDeferRehash (CPLHashSet ∗set, const void ∗elt)
- unsigned long CPL_DLL CPLHashSetHashPointer (const void ∗elt)
- int CPL_DLL CPLHashSetEqualPointer (const void ∗elt1, const void ∗elt2)
- unsigned long CPL_DLL CPLHashSetHashStr (const void ∗pszStr)
- int CPL_DLL CPLHashSetEqualStr (const void ∗pszStr1, const void ∗pszStr2)

11.3.1 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.

11.3.2 Typedef Documentation

11.3.2.1 CPLHashSet

```
typedef typedefCPL_C_START struct _CPLHashSet CPLHashSet
```

Opaque type for a hash set

11.3.2.2 CPLHashSetEqualFunc

```
typedef int(* CPLHashSetEqualFunc) (const void *elt1, const void *elt2)
```

CPLHashSetEqualFunc

11.3.2.3 CPLHashSetFreeEltFunc

```
typedef void(* CPLHashSetFreeEltFunc) (void *elt)
```

CPLHashSetFreeEltFunc

11.3.2.4 CPLHashSetHashFunc

```
typedef unsigned long(* CPLHashSetHashFunc) (const void *elt)
```

CPLHashSetHashFunc

11.3.2.5 CPLHashSetIterEltFunc

```
typedef int(* CPLHashSetIterEltFunc) (void *elt, void *user_data)
```

CPLHashSetIterEltFunc

11.3.3 Function Documentation

11.3.3.1 CPLHashSetClear()

```
void CPL_DLL CPLHashSetClear (
            CPLHashSet * set )
```

Clear all elements from a hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters

set	the hash set

Since

GDAL 2.1

11.3.3.2 CPLHashSetDestroy()

```
void CPL_DLL CPLHashSetDestroy (
            CPLHashSet * set )
```

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters

set	the hash set

11.3.3.3 CPLHashSetEqualPointer()

```
int CPL_DLL CPLHashSetEqualPointer (
            const void * elt1,
            const void * elt2 )
```

Equality function for arbitrary pointers

Parameters

elt1	the first arbitrary pointer to compare
elt2	the second arbitrary pointer to compare

Returns

TRUE if the pointers are equal

11.3.3.4 CPLHashSetEqualStr()

```
int CPL_DLL CPLHashSetEqualStr (
            const void * elt1,
            const void * elt2 )
```

Equality function for strings

Parameters

elt1	the first string to compare. May be NULL.
elt2	the second string to compare. May be NULL.

Returns

TRUE if the strings are equal

11.3.3.5 CPLHashSetForeach()

```
void CPL_DLL CPLHashSetForeach (
            CPLHashSet * set,
            CPLHashSetIterEltFunc fnIterFunc,
            void * user_data )
```

Walk through the hash set and runs the provided function on all the elements

This function is provided the user_data argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

Parameters

set	the hash set.
fnIterFunc	the function called on each element.
user_data	the user data provided to the function.

11.3.3.6 CPLHashSetHashPointer()

```
unsigned long CPL_DLL CPLHashSetHashPointer (
            const void * elt )
```

Hash function for an arbitrary pointer

Parameters

elt	the arbitrary pointer to hash

Returns

the hash value of the pointer

11.3.3.7 CPLHashSetHashStr()

```
unsigned long CPL_DLL CPLHashSetHashStr (
            const void * elt )
```

Hash function for a zero-terminated string

Parameters

elt	the string to hash. May be NULL.

Returns

the hash value of the string

11.3.3.8 CPLHashSetInsert()

```
int CPL_DLL CPLHashSetInsert (
            CPLHashSet * set,
            void * elt )
```

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

Parameters

set	the hash set
elt	the new element to insert in the hash set

Returns

TRUE if the element was not already in the hash set

11.3.3.9 CPLHashSetLookup()

```
void CPL_DLL* CPLHashSetLookup (
            CPLHashSet * set,
            const void * elt )
```

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

Parameters

set	the hash set
elt	the element to look up in the hash set

Returns

the element found in the hash set or NULL

11.3.3.10 CPLHashSetNew()

```
CPLHashSet CPL_DLL* 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, CPLHashSetHash↩ Pointer will be used.

The equal function must return if two elements are equal. If fnEqualFunc is NULL, CPLHashSetEqualPointer will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If fnFreeEltFunc is NULL, elements inserted into the hash set will not be freed.

Parameters

fnHashFunc	hash function. May be NULL.
fnEqualFunc	equal function. May be NULL.
fnFreeEltFunc	element free function. May be NULL.

Returns

a new hash set

11.3.3.11 CPLHashSetRemove()

```
int CPL_DLL CPLHashSetRemove (
            CPLHashSet * set,
            const void * elt )
```

Removes an element from a hash set

Parameters

set	the hash set
elt	the new element to remove from the hash set

Returns

TRUE if the element was in the hash set

11.3.3.12 CPLHashSetRemoveDeferRehash()

```
int CPL_DLL CPLHashSetRemoveDeferRehash (
            CPLHashSet * set,
            const void * elt )
```

Removes an element from a hash set.

This will defer potential rehashing of the set to later calls to CPLHashSetInsert() or CPLHashSetRemove().

Parameters

set	the hash set
elt	the new element to remove from the hash set

Returns

TRUE if the element was in the hash set

Since

GDAL 2.1

11.3.3.13 CPLHashSetSize()

```
int CPL_DLL CPLHashSetSize (
            const CPLHashSet * set )
```

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

Parameters

set	the hash set

Returns

the number of elements in the hash set

11.4 cpl_http.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "cpl_progress.h"
#include "cpl_vsi.h"
```

Classes

- struct CPLMimePart
- struct CPLHTTPResult

Functions

- int CPL_DLL CPLHTTPEnabled (void)

  Return if CPLHTTP services can be useful.
- CPLHTTPResult CPL_DLL * CPLHTTPFetch (const char *pszURL, CSLConstList papszOptions)

  Fetch a document from an url and return in a string.
- CPLHTTPResult CPL_DLL * CPLHTTPFetchEx (const char *pszURL, CSLConstList papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg, CPLHTTPFetchWriteFunc pfnWrite, void *pWriteArg)
- CPLHTTPResult CPL_DLL ** CPLHTTPMultiFetch (const char *const *papszURL, int nURLCount, int nMaxSimultaneous, CSLConstList papszOptions)

  Fetch several documents at once.
- void CPL_DLL CPLHTTPCleanup (void)

  Cleanup function to call at application termination.
- void CPL_DLL CPLHTTPDestroyResult (CPLHTTPResult *psResult)

  Clean the memory associated with the return value of CPLHTTPFetch()
- void CPL_DLL CPLHTTPDestroyMultiResult (CPLHTTPResult **papsResults, int nCount)

  Clean the memory associated with the return value of CPLHTTPMultiFetch()
- int CPL_DLL CPLHTTPParseMultipartMime (CPLHTTPResult *psResult)

  Parses a MIME multipart message.
- char CPL_DLL * GOA2GetAuthorizationURL (const char *pszScope)
- char CPL_DLL * GOA2GetRefreshToken (const char *pszAuthToken, const char *pszScope)
- char CPL_DLL ** GOA2GetAccessTokenFromServiceAccount (const char *pszPrivateKey, const char *pszClientEmail, const char *pszScope, CSLConstList papszAdditionalClaims, CSLConstList papszOptions)
- char CPL_DLL ** GOA2GetAccessTokenFromCloudEngineVM (CSLConstList papszOptions)

11.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

11.4.2 Function Documentation

11.4.2.1 CPLHTTPDestroyMultiResult()

```
void CPL_DLL CPLHTTPDestroyMultiResult (
            CPLHTTPResult ** papsResults,
            int nCount )
```

Clean the memory associated with the return value of CPLHTTPMultiFetch()

Parameters

papsResults	pointer to the return value of CPLHTTPMultiFetch()
nCount	value of the nURLCount parameter passed to CPLHTTPMultiFetch()

Since

GDAL 2.3

11.4.2.2 CPLHTTPDestroyResult()

```
void CPL_DLL CPLHTTPDestroyResult (
            CPLHTTPResult * psResult )
```

Clean the memory associated with the return value of CPLHTTPFetch()

Parameters

psResult	pointer to the return value of CPLHTTPFetch()

11.4.2.3 CPLHTTPEnabled()

```
int CPL_DLL CPLHTTPEnabled (
            void )
```

Return if CPLHTTP services can be useful.

Those services depend on GDAL being build with libcurl support.

Returns

TRUE if libcurl support is enabled

11.4.2.4 CPLHTTPFetch()

```
CPLHTTPResult CPL_DLL* CPLHTTPFetch (
            const char * pszURL,
            CSLConstList papszOptions )
```

Fetch a document from an url and return in a string.

Parameters

pszURL	valid URL recognized by underlying download library (libcurl)

Parameters

papszOptions	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : • CONNECTTIMEOUT=val, where val is in seconds (possibly with decimals). This is the maximum delay for the connection to be established before being aborted (GDAL >= 2.2). • TIMEOUT=val, where val is in seconds. This is the maximum delay for the whole request to complete before being aborted. • LOW_SPEED_TIME=val, where val is in seconds. This is the maximum time where the transfer speed should be below the LOW_SPEED_LIMIT (if not specified 1b/s), before the transfer to be considered too slow and aborted. (GDAL >= 2.1) • LOW_SPEED_LIMIT=val, where val is in bytes/second. See LOW_SPEED_TIME. Has only effect if LOW_SPEED_TIME is specified too. (GDAL >= 2.1) • HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwkt" • HEADER_FILE=filename: filename of a text file with "key: value" headers. (GDAL >= 2.2) • HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use. • USERPWD=userid:password to specify a user and password for authentication • POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request. • PROXY=val, to make requests go through a proxy server, where val is of the form proxy.server.com:port_number • PROXYUSERPWD=val, where val is of the form username:password • PROXYAUTH=[BASIC/NTLM/DIGEST/ANY] to specify an proxy authentication scheme to use. • NETRC=[YES/NO] to enable or disable use of $HOME/.netrc, default YES. • CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL >= 1.9.0) • COOKIE=val, where val is formatted as COOKIE1=VALUE1; COOKIE2=VALUE2; ... • MAX_RETRY=val, where val is the maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs. Default is 0. (GDAL >= 2.0) • RETRY_DELAY=val, where val is the number of seconds between retry attempts. Default is 30. (GDAL >= 2.0) • MAX_FILE_SIZE=val, where val is a number of bytes (GDAL >= 2.2) • CAINFO=/path/to/bundle.crt. This is path to Certificate Authority (CA) bundle file. By default, it will be looked in a system location. If the CAINFO options is not defined, GDAL will also look if the CURL_CA_BUNDLE environment variable is defined to use it as the CAINFO value, and as a fallback to the SSL_CERT_FILE environment variable. (GDAL >= 2.1.3) • HTTP_VERSION=1.0/1.1/2/2TLS (GDAL >= 2.3). Specify HTTP version to use. Will default to 1.1 generally (except on some controlled environments, like Google Compute Engine VMs, where 2TLS will be the default). Support for HTTP/2 requires curl 7.33 or later, built against nghttp2. "2TLS" means that HTTP/2 will be attempted for HTTPS connections only. Whereas "2" means that HTTP/2 will be attempted for HTTP or HTTPS. • SSL_VERIFYSTATUS=YES/NO (GDAL >= 2.3, and curl >= 7.41): determines whether the status of the server cert using the "Certificate Status Request" TLS extension (aka. OCSP stapling) should be checked. If this option is enabled but the server does not

Parameters

Alternatively, if not defined in the papszOptions arguments, the CONNECTTIMEOUT, TIMEOUT, LOW_SPEED_TIME, LOW_SPEED_LIMIT, USERPWD, PROXY, PROXYUSERPWD, PROXYAUTH, NETRC, MAX_RETRY and RETRY_DELAY, HEADER_FILE, HTTP_VERSION, SSL_VERIFYSTATUS, USE_CAPI_STORE values are searched in the configuration options respectively named GDAL_HTTP_CONNECTTIMEOUT, GDAL_HTTP_TIMEOUT, GDAL_HTTP_LOW_SPEED_TIME, GDAL_HTTP_LOW_SPEED_LIMIT, GDAL_HTTP_USERPWD, GDAL_HTTP_PROXY, GDAL_HTTP_PROXYUSERPWD, GDAL_PROXY_AUTH, GDAL_HTTP_NETRC, GDAL_HTTP_MAX_RETRY, GDAL_HTTP_RETRY_DELAY, GDAL_HTTP_HEADER_FILE, GDAL_HTTP_VERSION, GDAL_HTTP_SSL_VERIFYSTATUS, GDAL_HTTP_USE_CAPI_STORE

Returns

a CPLHTTPResult* structure that must be freed by CPLHTTPDestroyResult(), or NULL if libcurl support is disabled

11.4.2.5 CPLHTTPFetchEx()

```
CPLHTTPResult CPL_DLL* CPLHTTPFetchEx (
            const char * pszURL,
            CSLConstList papszOptions,
            GDALProgressFunc pfnProgress,
            void * pProgressArg,
            CPLHTTPFetchWriteFunc pfnWrite,
            void * pWriteArg )
```

Fetch a document from an url and return in a string.

Parameters

pszURL	Url to fetch document from web.
papszOptions	Option list as a NULL-terminated array of strings. Available keys see in CPLHTTPFetch.
pfnProgress	Callback for reporting algorithm progress matching the GDALProgressFunc() semantics. May be NULL.
pProgressArg	Callback argument passed to pfnProgress.
pfnWrite	Write function pointer matching the CPLHTTPWriteFunc() semantics. May be NULL.
pWriteArg	Argument which will pass to a write function.

Returns

A CPLHTTPResult* structure that must be freed by CPLHTTPDestroyResult(), or NULL if libcurl support is disabled.

11.4.2.6 CPLHTTPMultiFetch()

```
CPLHTTPResult CPL_DLL** CPLHTTPMultiFetch (
            const char *const * papszURL,
```

```
int nURLCount,
int nMaxSimultaneous,
CSLConstList papszOptions )
```

Fetch several documents at once.

Parameters

papszURL	array of valid URLs recognized by underlying download library (libcurl)
nURLCount	number of URLs of papszURL
nMaxSimultaneous	maximum number of downloads to issue simultaneously. Any negative or zer value means unlimited.
papszOptions	option list as a NULL-terminated array of strings. May be NULL. Refer to CPLHTTPFetch() for valid options.

Returns

an array of CPLHTTPResult∗ structures that must be freed by CPLHTTPDestroyMultiResult() or NULL if libcurl support is disabled

Since

GDAL 2.3

11.4.2.7 CPLHTTPParseMultipartMime()

```
int CPL_DLL CPLHTTPParseMultipartMime (
            CPLHTTPResult * psResult )
```

Parses a MIME multipart message.

This function will iterate over each part and put it in a separate element of the pasMimePart array of the provided psResult structure.

Parameters

psResult	pointer to the return value of CPLHTTPFetch()

Returns

TRUE if the message contains MIME multipart message.

11.4.2.8 GOA2GetAccessTokenFromCloudEngineVM()

```
char CPL_DLL** GOA2GetAccessTokenFromCloudEngineVM (
            CSLConstList papszOptions )
```

Fetch access token using Cloud Engine internal REST API

The default service accounts bound to the current Google Cloud Engine VM is used for OAuth2 authentication

A CPLError will be reported if the request fails for some reason. Common reasons include the refresh token having been revoked by the user or http connection problems.

Parameters

papszOptions	NULL terminated list of options. None currently

Returns

a list of key=value pairs, including a access_token and expires_in

Since

GDAL 2.3

11.4.2.9 GOA2GetAccessTokenFromServiceAccount()

```
char CPL_DLL** GOA2GetAccessTokenFromServiceAccount (
            const char * pszPrivateKey,
            const char * pszClientEmail,
            const char * pszScope,
            CSLConstList papszAdditionalClaims,
            CSLConstList papszOptions )
```

Fetch access token using Service Account OAuth2

See https://developers.google.com/identity/protocols/OAuth2ServiceAccount

A CPLError will be reported if the request fails for some reason.

Parameters

pszPrivateKey	Private key as a RSA private key
pszClientEmail	Client email
pszScope	the service being requested
papszAdditionalClaims	additional claims, or NULL
papszOptions	NULL terminated list of options. None currently

Returns

a list of key=value pairs, including a access_token and expires_in

Since

GDAL 2.3

See https://developers.google.com/identity/protocols/OAuth2ServiceAccount and https://jwt.io/

11.4.2.10 GOA2GetAuthorizationURL()

```
char CPL_DLL* GOA2GetAuthorizationURL (
            const char * pszScope )
```

Return authorization url for a given scope.

Returns the URL that a user should visit, and use for authentication in order to get an "auth token" indicating their willingness to use a service.

Note that when the user visits this url they will be asked to login (using a google/gmail/etc) account, and to authorize use of the requested scope for the application "GDAL/OGR". Once they have done so, they will be presented with a lengthy string they should "enter into their application". This is the "auth token" to be passed to GOA2GetRefreshToken(). The "auth token" can only be used once.

This function should never fail.

Parameters

pszScope	the service being requested, not yet URL encoded, such as "https://www.googleapis.com/auth/fusiontables".

Returns

the URL to visit - should be freed with CPLFree().

11.4.2.11 GOA2GetRefreshToken()

```
char CPL_DLL* GOA2GetRefreshToken (
            const char * pszAuthToken,
            const char * pszScope )
```

Turn Auth Token into a Refresh Token.

A one time "auth token" provided by the user is turned into a reusable "refresh token" using a google oauth2 web service.

A CPLError will be reported if the translation fails for some reason. Common reasons include the auth token already having been used before, it not being appropriate for the passed scope and configured client api or http connection problems. NULL is returned on error.

Parameters

pszAuthToken	the authorization token from the user.
pszScope	the scope for which it is valid.

Returns

refresh token, to be freed with CPLFree(), null on failure.

11.5 cpl_json.h File Reference

```
#include "cpl_progress.h"
#include <string>
#include <vector>
```

Classes

- class CPLJSONObject

  The CPLJSONArray class holds JSON object from CPLJSONDocument.
- class CPLJSONArray

  The JSONArray class JSON array from JSONDocument.
- class CPLJSONDocument

  The CPLJSONDocument class Wrapper class around json-c library.

11.5.1 Detailed Description

Interface for read and write JSON documents

11.6 cpl_list.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct _CPLList

Typedefs

- typedef typedefCPL_C_START struct _CPLList CPLList

Functions

- CPLList CPL_DLL * CPLListAppend (CPLList *psList, void *pData)
- CPLList CPL_DLL * CPLListInsert (CPLList *psList, void *pData, int nPosition)
- CPLList CPL_DLL * CPLListGetLast (CPLList *psList)
- CPLList CPL_DLL * CPLListGet (CPLList *const psList, int nPosition)
- int CPL_DLL CPLListCount (const CPLList *psList)
- CPLList CPL_DLL * CPLListRemove (CPLList *psList, int nPosition)
- void CPL_DLL CPLListDestroy (CPLList *psList)
- CPLList CPL_DLL * CPLListGetNext (const CPLList *psElement)
- void CPL_DLL * CPLListGetData (const CPLList *psElement)

11.6.1 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.

11.6.2 Typedef Documentation

11.6.2.1 CPLList

```
typedef typedefCPL_C_START struct _CPLList CPLList
```

List element structure.

11.6.3 Function Documentation

11.6.3.1 CPLListAppend()

```
CPLList CPL_DLL* CPLListAppend (
            CPLList * psList,
            void * pData )
```

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

Parameters

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.

Returns

pointer to the head of modified list.

11.6.3.2 CPLListCount()

```
int CPL_DLL CPLListCount (
            const CPLList * psList )
```

Return the number of elements in a list.

Parameters

psList	pointer to list head.

Returns

number of elements in a list.

11.6.3.3 CPLListDestroy()

```
void CPL_DLL CPLListDestroy (
            CPLList * psList )
```

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters

psList	pointer to list head.

11.6.3.4 CPLListGet()

```
CPLList CPL_DLL* CPLListGet (
            CPLList * psList,
            int nPosition )
```

Return the pointer to the specified element in a list.

Parameters

psList	pointer to list head.
nPosition	the index of the element in the list, 0 being the first element.

Returns

pointer to the specified element in a list.

11.6.3.5 CPLListGetData()

```
void CPL_DLL* CPLListGetData (
            const CPLList * psElement )
```

Return pointer to the data object contained in given list element.

Parameters

psElement	pointer to list element.

Returns

pointer to the data object contained in given list element.

11.6.3.6 CPLListGetLast()

```
CPLList CPL_DLL* CPLListGetLast (
            CPLList *const psList )
```

Return the pointer to last element in a list.

Parameters

psList	pointer to list head.

Returns

pointer to last element in a list.

11.6.3.7 CPLListGetNext()

```
CPLList CPL_DLL* CPLListGetNext (
            const CPLList * psElement )
```

Return the pointer to next element in a list.

Parameters

psElement	pointer to list element.

Returns

pointer to the list element preceded by the given element.

11.6.3.8 CPLListInsert()

```
CPLList CPL_DLL* CPLListInsert (
            CPLList * psList,
            void * pData,
            int nPosition )
```

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

Parameters

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.
nPosition	position number to insert an object.

Returns

pointer to the head of modified list.

11.6.3.9 CPLListRemove()

```
CPLList CPL_DLL* CPLListRemove (
            CPLList * psList,
            int nPosition )
```

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

Parameters

psList	pointer to list head.
nPosition	position number to delete an element.

Returns

pointer to the head of modified list.

11.7 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 CPL_DLL * CPLParseXMLString (const char *)

  Parse an XML string into tree form.
- void CPL_DLL CPLDestroyXMLNode (CPLXMLNode *)

  Destroy a tree.
- CPLXMLNode CPL_DLL * CPLGetXMLNode (CPLXMLNode *poRoot, const char *pszPath)

  Find node by path.
- CPLXMLNode CPL_DLL * CPLSearchXMLNode (CPLXMLNode *poRoot, const char *pszTarget)

  Search for a node in document.
- const char CPL_DLL * CPLGetXMLValue (const CPLXMLNode *poRoot, const char *pszPath, const char *pszDefault)

  Fetch element/attribute value.
- CPLXMLNode CPL_DLL * CPLCreateXMLNode (CPLXMLNode *poParent, CPLXMLNodeType eType, const char *pszText)

  Create an document tree item.
- char CPL_DLL * CPLSerializeXMLTree (const CPLXMLNode *psNode)

  Convert tree into string document.
- void CPL_DLL CPLAddXMLChild (CPLXMLNode *psParent, CPLXMLNode *psChild)

  Add child node to parent.
- int CPL_DLL CPLRemoveXMLChild (CPLXMLNode *psParent, CPLXMLNode *psChild)

  Remove child node from parent.
- void CPL_DLL CPLAddXMLSibling (CPLXMLNode *psOlderSibling, CPLXMLNode *psNewSibling)

  Add new sibling.
- CPLXMLNode CPL_DLL * CPLCreateXMLElementAndValue (CPLXMLNode *psParent, const char *psz↩Name, const char *pszValue)

  Create an element and text value.
- void CPL_DLL CPLAddXMLAttributeAndValue (CPLXMLNode *psParent, const char *pszName, const char *pszValue)

Create an attribute and text value.

- CPLXMLNode CPL_DLL * CPLCloneXMLTree (const CPLXMLNode *psTree)

  Copy tree.
- int CPL_DLL CPLSetXMLValue (CPLXMLNode *psRoot, const char *pszPath, const char *pszValue)

  Set element value by path.
- void CPL_DLL CPLStripXMLNamespace (CPLXMLNode *psRoot, const char *pszNameSpace, int b↩ Recurse)

  Strip indicated namespaces.
- void CPL_DLL CPLCleanXMLElementName (char *)

  Make string into safe XML token.
- CPLXMLNode CPL_DLL * CPLParseXMLFile (const char *pszFilename)

  Parse XML file into tree.
- int CPL_DLL CPLSerializeXMLTreeToFile (const CPLXMLNode *psTree, const char *pszFilename)

  Write document tree to a file.

11.7.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

11.7.2 Typedef Documentation

11.7.2.1 CPLXMLNode

```
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(). 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 hierarchical tree structure for a document can be represented as a tree of CPLXMLNode structures.

11.7.3 Enumeration Type Documentation

11.7.3.1 CPLXMLNodeType

```
enum CPLXMLNodeType
```

XML node type

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

11.7.4 Function Documentation

11.7.4.1 CPLAddXMLAttributeAndValue()

```
void CPL_DLL CPLAddXMLAttributeAndValue (
            CPLXMLNode * psParent,
            const char * pszName,
            const char * pszValue )
```

Create an attribute and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psAttributeNode;

psAttributeNode = CPLCreateXMLNode( psParent, CXT_Attribute, pszName );
CPLCreateXMLNode( psAttributeNode, CXT_Text, pszValue );
```

It creates a CXT_Attribute node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

psParent	the parent node to which the resulting node should be attached. Must not be NULL.
pszName	the attribute name to create.
pszValue	the text to attach to the attribute. Must not be NULL.

Since

GDAL 2.0

11.7.4.2 CPLAddXMLChild()

```
void CPL_DLL 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 (its psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

Parameters

psParent	the node to attach the child to. May not be NULL.
psChild	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

11.7.4.3 CPLAddXMLSibling()

```
void CPL_DLL 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().

Parameters

psOlderSibling	the node to attach the sibling after.
psNewSibling	the node to add at the end of psOlderSiblings psNext chain.

11.7.4.4 CPLCleanXMLElementName()

```
void CPL_DLL 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

pszTarget	the string to be adjusted. It is altered in place.

11.7.4.5 CPLCloneXMLTree()

```
CPLXMLNode CPL_DLL* CPLCloneXMLTree (
            const CPLXMLNode * psTree )
```

Copy tree.

Creates a deep copy of a CPLXMLNode tree.

Parameters

psTree	the tree to duplicate.

Returns

a copy of the whole tree.

11.7.4.6 CPLCreateXMLElementAndValue()

```
CPLXMLNode CPL_DLL* CPLCreateXMLElementAndValue (
            CPLXMLNode * psParent,
            const char * pszName,
            const char * pszValue )
```

Create an element and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

psParent	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
pszName	the element name to create.
pszValue	the text to attach to the element. Must not be NULL.

Returns

the pointer to the new element node.

11.7.4.7 CPLCreateXMLNode()

```
CPLXMLNode CPL_DLL* CPLCreateXMLNode (
            CPLXMLNode * poParent,
            CPLXMLNodeType eType,
            const char * pszText )
```

Create an document tree item.

Create a single CPLXMLNode object with the desired value and type, and attach it as a child of the indicated parent.

Parameters

poParent	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
eType	the type of the newly created node
pszText	the value of the newly created node

Returns

the newly created node, now owned by the caller (or parent node).

11.7.4.8 CPLDestroyXMLNode()

```
void CPL_DLL CPLDestroyXMLNode (
            CPLXMLNode * psNode )
```

Destroy a tree.

This function frees resources associated with a CPLXMLNode and all its children nodes.

Parameters

psNode	the tree to free.

11.7.4.9 CPLGetXMLNode()

```
CPLXMLNode CPL_DLL* 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 its siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

Parameters

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	the list of element names in the path (dot separated).

Returns

the requested element node, or NULL if not found.

11.7.4.10 CPLGetXMLValue()

```
const char CPL_DLL* CPLGetXMLValue (
            const 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() (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 no value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

Parameters

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
pszDefault	the value to return if a corresponding value is not found, may be NULL.

Returns

the requested value or pszDefault if not found.

11.7.4.11 CPLParseXMLFile()

```
CPLXMLNode CPL_DLL* CPLParseXMLFile (
            const char * pszFilename )
```

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with CPLParseXMLString(). Errors in reading the file or parsing the XML will be reported by CPLError().

The "large file" API is used, so XML files can come from virtualized files.

Parameters

pszFilename	the file to open.

Returns

NULL on failure, or the document tree on success.

11.7.4.12 CPLParseXMLString()

```
CPLXMLNode CPL_DLL* CPLParseXMLString (
            const char * pszString )
```

Parse an XML string into tree form.

The passed document is parsed into a CPLXMLNode tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via CPLError(). No validation beyond wellformedness is done. The CPLParseXMLFile() convenience function can be used to parse from a file.

The returned document tree is owned by the caller and should be freed with CPLDestroyXMLNode() when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT_Text node.

Parameters

pszString	the document to parse.

Returns

parsed tree or NULL on error.

11.7.4.13 CPLRemoveXMLChild()

```
int CPL_DLL 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 its own children, but is cleanly removed from the child list of the parent.

Parameters

psParent	the node to the child is attached to.
psChild	the child to remove.

Returns

TRUE on success or FALSE if the child was not found.

11.7.4.14 CPLSearchXMLNode()

```
CPLXMLNode CPL_DLL* 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 its first child, that child's first child, and so on.

Use CPLGetXMLNode() to find a specific child, or along a specific node path.

Parameters

psRoot	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
pszElement	the name of the element or attribute to search for.

Returns

The matching node or NULL on failure.

11.7.4.15 CPLSerializeXMLTree()

```
char CPL_DLL* CPLSerializeXMLTree (
            const CPLXMLNode * psNode )
```

Convert tree into string document.

This function converts a CPLXMLNode tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with CPLFree() when no longer needed.

Parameters

psNode	the node to serialize.

Returns

the document on success or NULL on failure.

11.7.4.16 CPLSerializeXMLTreeToFile()

```
int CPL_DLL CPLSerializeXMLTreeToFile (
            const CPLXMLNode * psTree,
            const char * pszFilename )
```

Write document tree to a file.

The passed document tree is converted into one big string (with CPLSerializeXMLTree()) and then written to the named file. Errors writing the file will be reported by CPLError(). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters

psTree	the document tree to write.
pszFilename	the name of the file to write to.

Returns

TRUE on success, FALSE otherwise.

11.7.4.17 CPLSetXMLValue()

```
int CPL_DLL CPLSetXMLValue (
            CPLXMLNode * psRoot,
            const char * pszPath,
            const char * pszValue )
```

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue("Citation.Id.Description", "DOQ dataset"); CPLSetXMLValue("Citation.Id.↩ Description.#name", "doq");

Parameters

psRoot	the subdocument to be updated.
pszPath	the dot separated path to the target element/attribute.
pszValue	the text value to assign.

Returns

TRUE on success.

11.7.4.18 CPLStripXMLNamespace()

```
void CPL_DLL CPLStripXMLNamespace (
            CPLXMLNode * psRoot,
            const char * pszNamespace,
            int bRecurse )
```

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

Parameters

psRoot	the document to operate on.
pszNamespace	the name space prefix (not including colon), or NULL.
bRecurse	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

11.8 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

11.8.1 Detailed Description

ODBC Abstraction Layer (C++).

11.9 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>
#include <strings.h>
```

Macros

- #define GINTBIG_MIN (CPL_STATIC_CAST(GIntBig, 0x80000000) << 32)
- #define GINTBIG_MAX ((CPL_STATIC_CAST(GIntBig, 0x7FFFFFFF) << 32) | 0xFFFFFFFFU)
- #define GUINTBIG_MAX ((CPL_STATIC_CAST(GUIntBig, 0xFFFFFFFFU) << 32) | 0xFFFFFFFFU)
- #define GINT64_MIN GINTBIG_MIN
- #define GINT64_MAX GINTBIG_MAX
- #define GUINT64_MAX GUINTBIG_MAX
- #define CPL_FRMT_GB_WITHOUT_PREFIX "ll"
- #define CPL_FRMT_GIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "d"
- #define CPL_FRMT_GUIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "u"
- #define MIN(a, b) (((a)<(b)) ? (a) : (b))
- #define MAX(a, b) (((a)>(b)) ? (a) : (b))
- #define ABS(x) (((x)<0) ? (-1*(x)) : (x))

- #define M_PI 3.14159265358979323846
- #define STRCASECMP(a, b) (strcasecmp(a,b))
- #define STRNCASECMP(a, b, n) (strncasecmp(a,b,n))
- #define EQUALN(a, b, n) (STRNCASECMP(a,b,n)==0)
- #define EQUAL(a, b) (STRCASECMP(a,b)==0)
- #define STARTS_WITH(a, b) (strncmp(a,b,strlen(b)) == 0)
- #define STARTS_WITH_CI(a, b) EQUALN(a,b,strlen(b))
- #define CPLIsNan(x) isnan(x)
- #define CPL_SWAP16(x) CPL_STATIC_CAST(GUInt16, (CPL_STATIC_CAST(GUInt16, x) << 8) | (CPL↩_STATIC_CAST(GUInt16, x) >> 8))
- #define CPL_SWAP32(x)
- #define CPL_SWAP64(x)
- #define CPL_SWAP16PTR(x)
- #define CPL_SWAP32PTR(x)
- #define CPL_SWAP64PTR(x)
- #define CPL_SWAPDOUBLE(p) CPL_SWAP64PTR(p)
- #define CPL_LSBWORD16(x) (x)
- #define CPL_MSBWORD16(x) CPL_SWAP16(x)
- #define CPL_LSBWORD32(x) (x)
- #define CPL_MSBWORD32(x) CPL_SWAP32(x)
- #define CPL_LSBPTR16(x) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 2)
- #define CPL_MSBPTR16(x) CPL_SWAP16PTR(x)
- #define CPL_LSBPTR32(x) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 4)
- #define CPL_MSBPTR32(x) CPL_SWAP32PTR(x)
- #define CPL_LSBPTR64(x) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 8)
- #define CPL_MSBPTR64(x) CPL_SWAP64PTR(x)
- #define CPL_LSBINT16PTR(x) ((*CPL_REINTERPRET_CAST(const GByte*, x)) | (*((CPL_REINTERPR↩ET_CAST(const GByte*, x))+1) << 8))
- #define CPL_LSBINT32PTR(x)
- #define CPL_LSBSINT16PTR(x) CPL_STATIC_CAST(GInt16,CPL_LSBINT16PTR(x))
- #define CPL_LSBUINT16PTR(x) CPL_STATIC_CAST(GUInt16, CPL_LSBINT16PTR(x))
- #define CPL_LSBSINT32PTR(x) CPL_STATIC_CAST(GInt32, CPL_LSBINT32PTR(x))
- #define CPL_LSBUINT32PTR(x) CPL_STATIC_CAST(GUInt32, CPL_LSBINT32PTR(x))
- #define CPL_NULL_TERMINATED
- #define CPL_PRINT_FUNC_FORMAT(format_idx, arg_idx)
- #define CPL_SCAN_FUNC_FORMAT(format_idx, arg_idx)
- #define CPL_FORMAT_STRING(arg) arg
- #define CPL_SCANF_FORMAT_STRING(arg) arg
- #define CPL_WARN_UNUSED_RESULT
- #define CPL_UNUSED
- #define CPL_NO_RETURN
- #define CPL_RETURNS_NONNULL
- #define CPL_RESTRICT
- #define CPL_FALLTHROUGH

Typedefs

- typedef int GInt32
- typedef unsigned int GUInt32
- typedef short GInt16
- typedef unsigned short GUInt16
- typedef unsigned char GByte
- typedef int GBool
- typedef long long GIntBig

- typedef unsigned long long GUIntBig
- typedef GIntBig GInt64
- typedef GUIntBig GUInt64
- typedef GIntBig GPtrDiff_t
- typedef char ** CSLConstList

11.9.1 Detailed Description

Core portability definitions for CPL.

11.9.2 Macro Definition Documentation

11.9.2.1 ABS

```
#define ABS(
            x ) (((x)<0) ? (-1*(x)) : (x))
```

Macro to compute the absolute value

11.9.2.2 CPL_FALLTHROUGH

```
#define CPL_FALLTHROUGH
```

Macro for fallthrough in a switch case construct

11.9.2.3 CPL_FORMAT_STRING

```
#define CPL_FORMAT_STRING(
            arg ) arg
```

Macro into which to wrap the format argument of a printf-like function

11.9.2.4 CPL_FRMT_GB_WITHOUT_PREFIX

```
#define CPL_FRMT_GB_WITHOUT_PREFIX "ll"
```

Printf formatting suffix for GIntBig

11.9.2.5 CPL_FRMT_GIB

```
#define CPL_FRMT_GIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "d"
```

Printf formatting for GIntBig

11.9.2.6 CPL_FRMT_GUIB

```
#define CPL_FRMT_GUIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "u"
```

Printf formatting for GUIntBig

11.9.2.7 CPL_LSBINT16PTR

```
#define CPL_LSBINT16PTR(
            x ) ((*CPL_REINTERPRET_CAST(const GByte*, x)) | (*((CPL_REINTERPRET_CAST(const
GByte*, x))+1) << 8))
```

Return a Int16 from the 2 bytes ordered in LSB order at address x.

Deprecated Use rather CPL_LSBSINT16PTR or CPL_LSBUINT16PTR for explicit signedness.

11.9.2.8 CPL_LSBINT32PTR

```
#define CPL_LSBINT32PTR(
            x )
```

Value:

```
((*CPL_REINTERPRET_CAST(const GByte*, x)) | (*((CPL_REINTERPRET_CAST(const
      GByte*, x))+1) << 8) | \
                              (*((CPL_REINTERPRET_CAST(const GByte*, x))+2) << 16) | (*((
      CPL_REINTERPRET_CAST(const GByte*, x))+3) << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x.

Deprecated Use rather CPL_LSBSINT32PTR or CPL_LSBUINT32PTR for explicit signedness.

11.9.2.9 CPL_LSBPTR16

```
#define CPL_LSBPTR16(
            x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 2)
```

Byte-swap if necessary a 16bit word at the location pointed from a originally LSB ordered pointer

11.9.2.10 CPL_LSBPTR32

```
#define CPL_LSBPTR32(
            x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 4)
```

Byte-swap if necessary a 32bit word at the location pointed from a originally LSB ordered pointer

11.9.2.11 CPL_LSBPTR64

```
#define CPL_LSBPTR64(
            x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 8)
```

Byte-swap if necessary a 64bit word at the location pointed from a originally LSB ordered pointer

11.9.2.12 CPL_LSBSINT16PTR

```
#define CPL_LSBSINT16PTR(
            x ) CPL_STATIC_CAST(GInt16,CPL_LSBINT16PTR(x))
```

Return a signed Int16 from the 2 bytes ordered in LSB order at address x

11.9.2.13 CPL_LSBSINT32PTR

```
#define CPL_LSBSINT32PTR(
            x ) CPL_STATIC_CAST(GInt32, CPL_LSBINT32PTR(x))
```

Return a signed Int32 from the 4 bytes ordered in LSB order at address x

11.9.2.14 CPL_LSBUINT16PTR

```
#define CPL_LSBUINT16PTR(
            x ) CPL_STATIC_CAST(GUInt16, CPL_LSBINT16PTR(x))
```

Return a unsigned Int16 from the 2 bytes ordered in LSB order at address x

11.9.2.15 CPL_LSBUINT32PTR

```
#define CPL_LSBUINT32PTR(
            x ) CPL_STATIC_CAST(GUInt32, CPL_LSBINT32PTR(x))
```

Return a unsigned Int32 from the 4 bytes ordered in LSB order at address x

11.9.2.16 CPL_LSBWORD16

```
#define CPL_LSBWORD16(
            x ) (x)
```

Return a 16bit word from a originally LSB ordered word

11.9.2.17 CPL_LSBWORD32

```
#define CPL_LSBWORD32(
            x ) (x)
```

Return a 32bit word from a originally LSB ordered word

11.9.2.18 CPL_MSBPTR16

```
#define CPL_MSBPTR16(
            x ) CPL_SWAP16PTR(x)
```

Byte-swap if necessary a 16bit word at the location pointed from a originally MSB ordered pointer

11.9.2.19 CPL_MSBPTR32

```
#define CPL_MSBPTR32(
            x ) CPL_SWAP32PTR(x)
```

Byte-swap if necessary a 32bit word at the location pointed from a originally MSB ordered pointer

11.9.2.20 CPL_MSBPTR64

```
#define CPL_MSBPTR64(
            x ) CPL_SWAP64PTR(x)
```

Byte-swap if necessary a 64bit word at the location pointed from a originally MSB ordered pointer

11.9.2.21 CPL_MSBWORD16

```
#define CPL_MSBWORD16(
            x ) CPL_SWAP16(x)
```

Return a 16bit word from a originally MSB ordered word

11.9.2.22 CPL_MSBWORD32

```
#define CPL_MSBWORD32(
            x ) CPL_SWAP32(x)
```

Return a 32bit word from a originally MSB ordered word

11.9.2.23 CPL_NO_RETURN

```
#define CPL_NO_RETURN
```

Qualifier for a function that does not return at all (terminates the process)

11.9.2.24 CPL_NULL_TERMINATED

```
#define CPL_NULL_TERMINATED
```

Null terminated variadic

11.9.2.25 CPL_PRINT_FUNC_FORMAT

```
#define CPL_PRINT_FUNC_FORMAT(
              format_idx,
              arg_idx )
```

Tag a function to have printf() formatting

11.9.2.26 CPL_RESTRICT

```
#define CPL_RESTRICT
```

restrict keyword to declare that pointers do not alias

11.9.2.27 CPL_RETURNS_NONNULL

```
#define CPL_RETURNS_NONNULL
```

Qualifier for a function that does not return NULL

11.9.2.28 CPL_SCAN_FUNC_FORMAT

```
#define CPL_SCAN_FUNC_FORMAT(
              format_idx,
              arg_idx )
```

Tag a function to have scanf() formatting

11.9.2.29 CPL_SCANF_FORMAT_STRING

```
#define CPL_SCANF_FORMAT_STRING(
              arg ) arg
```

Macro into which to wrap the format argument of a sscanf-like function.

11.9.2.30 CPL_SWAP16

```
#define CPL_SWAP16(
              x ) CPL_STATIC_CAST(GUInt16, (CPL_STATIC_CAST(GUInt16, x) << 8) | (CPL_STATIC_↩
CAST(GUInt16, x) >> 8) )
```

Byte-swap a 16bit unsigned integer

11.9.2.31 CPL_SWAP16PTR

```
#define CPL_SWAP16PTR(
            x )
```

Value:

```
{                                                                   \
    GByte       byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x);              \
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 2); \
                                                                    \
    byTemp = _pabyDataT[0];                                         \
    _pabyDataT[0] = _pabyDataT[1];                                  \
    _pabyDataT[1] = byTemp;                                         \
}
```

Byte-swap a 16 bit pointer

11.9.2.32 CPL_SWAP32

```
#define CPL_SWAP32(
            x )
```

Value:

```
CPL_STATIC_CAST(GUInt32, \
            ((CPL_STATIC_CAST(GUInt32, x) & 0x000000ffU) << 24) | \
            ((CPL_STATIC_CAST(GUInt32, x) & 0x0000ff00U) <<  8) | \
            ((CPL_STATIC_CAST(GUInt32, x) & 0x00ff0000U) >>  8) | \
            ((CPL_STATIC_CAST(GUInt32, x) & 0xff000000U) >> 24) )
```

Byte-swap a 32bit unsigned integer

11.9.2.33 CPL_SWAP32PTR

```
#define CPL_SWAP32PTR(
            x )
```

Value:

```
{                                                                   \
    GByte       byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x);              \
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 4);  \
                                                                    \
    byTemp = _pabyDataT[0];                                         \
    _pabyDataT[0] = _pabyDataT[3];                                  \
    _pabyDataT[3] = byTemp;                                         \
    byTemp = _pabyDataT[1];                                         \
    _pabyDataT[1] = _pabyDataT[2];                                  \
    _pabyDataT[2] = byTemp;                                         \
}
```

Byte-swap a 32 bit pointer

11.9.2.34 CPL_SWAP64

```
#define CPL_SWAP64(
            x )
```

Value:

```
((CPL_STATIC_CAST(GUInt64, CPL_SWAP32(CPL_STATIC_CAST(GUInt32, x))) << 32) | \
            (CPL_STATIC_CAST(GUInt64, CPL_SWAP32(CPL_STATIC_CAST(
      GUInt32, CPL_STATIC_CAST(GUInt64, x) >> 32)))))
```

Byte-swap a 64bit unsigned integer

11.9.2.35 CPL_SWAP64PTR

```
#define CPL_SWAP64PTR(
            x )
```

Value:

```
{                                                                   \
    GByte       byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x);              \
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 8); \
                                                                    \
    byTemp = _pabyDataT[0];                                         \
    _pabyDataT[0] = _pabyDataT[7];                                  \
    _pabyDataT[7] = byTemp;                                         \
    byTemp = _pabyDataT[1];                                         \
    _pabyDataT[1] = _pabyDataT[6];                                  \
    _pabyDataT[6] = byTemp;                                         \
    byTemp = _pabyDataT[2];                                         \
    _pabyDataT[2] = _pabyDataT[5];                                  \
    _pabyDataT[5] = byTemp;                                         \
    byTemp = _pabyDataT[3];                                         \
    _pabyDataT[3] = _pabyDataT[4];                                  \
    _pabyDataT[4] = byTemp;                                         \
}
```

Byte-swap a 64 bit pointer

11.9.2.36 CPL_SWAPDOUBLE

```
#define CPL_SWAPDOUBLE(
            p ) CPL_SWAP64PTR(p)
```

Byte-swap a 64 bit pointer

11.9.2.37 CPL_UNUSED

```
#define CPL_UNUSED
```

Qualifier for an argument that is unused

11.9.2.38 CPL_WARN_UNUSED_RESULT

```
char CPL_DLL CPL_WARN_UNUSED_RESULT
```

Qualifier to warn when the return value of a function is not used

11.9.2.39 CPLIsNan

```
#define CPLIsNan(
            x ) isnan(x)
```

Return whether a floating-pointer number is NaN

11.9.2.40 EQUAL

```
#define EQUAL(
            a,
            b ) (STRCASECMP(a,b)==0)
```

Alias for strcasecmp() == 0

11.9.2.41 EQUALN

```
#define EQUALN(
            a,
            b,
            n ) (STRNCASECMP(a,b,n)==0)
```

Alias for strncasecmp() == 0

11.9.2.42 GINT64_MAX

```
#define GINT64_MAX GINTBIG_MAX
```

Maximum GInt64 value

11.9.2.43 GINT64_MIN

```
#define GINT64_MIN GINTBIG_MIN
```

Minimum GInt64 value

11.9.2.44 GINTBIG_MAX

```
#define GINTBIG_MAX ((CPL_STATIC_CAST(GIntBig, 0x7FFFFFFF) << 32) | 0xFFFFFFFFU)
```

Maximum GIntBig value

11.9.2.45 GINTBIG_MIN

```
#define GINTBIG_MIN (CPL_STATIC_CAST(GIntBig, 0x80000000) << 32)
```

Minimum GIntBig value

11.9.2.46 GUINT64_MAX

```
#define GUINT64_MAX GUINTBIG_MAX
```

Minimum GUInt64 value

11.9.2.47 GUINTBIG_MAX

```
#define GUINTBIG_MAX ((CPL_STATIC_CAST(GUIntBig, 0xFFFFFFFFU) << 32) | 0xFFFFFFFFU)
```

Maximum GUIntBig value

11.9.2.48 M_PI

```
#define M_PI 3.14159265358979323846
```

PI definition

11.9.2.49 MAX

```
#define MAX(
            a,
            b ) (((a)>(b)) ?  (a) :  (b))
```

Macro to compute the maximum of 2 values

11.9.2.50 MIN

```
#define MIN(
            a,
            b ) (((a)<(b)) ?  (a) :  (b))
```

Macro to compute the minimum of 2 values

11.9.2.51 STARTS_WITH

```
#define STARTS_WITH(
            a,
            b ) (strncmp(a,b,strlen(b)) == 0)
```

Returns whether a starts with b

11.9.2.52 STARTS_WITH_CI

```
#define STARTS_WITH_CI(
            a,
            b ) EQUALN(a,b,strlen(b))
```

Returns whether a starts with b (case insensitive comparison)

11.9.2.53 STRCASECMP

```
#define STRCASECMP(
            a,
            b ) (strcasecmp(a,b))
```

Alias for strcasecmp()

11.9.2.54 STRNCASECMP

```
#define STRNCASECMP(
            a,
            b,
            n ) (strncasecmp(a,b,n))
```

Alias for strncasecmp()

11.9.3 Typedef Documentation

11.9.3.1 CSLConstList

```
typedef char** CSLConstList
```

Type of a constant null-terminated list of nul terminated strings. Seen as char∗∗ from C and const char∗ const∗ from C++

11.9.3.2 GBool

```
typedef int GBool
```

Type for boolean values (alias to int)

11.9.3.3 GByte

```
typedef unsigned char GByte
```

Unsigned byte type

11.9.3.4 GInt16

```
typedef short GInt16
```

Int16 type

11.9.3.5 GInt32

```
typedef int GInt32
```

Int32 type

11.9.3.6 GInt64

```
typedef GIntBig GInt64
```

Signed 64 bit integer type

11.9.3.7 GIntBig

```
typedef long long GIntBig
```

Large signed integer type (generally 64-bit integer type). Use GInt64 when exactly 64 bit is needed

11.9.3.8 GPtrDiff_t

```
typedef GIntBig GPtrDiff_t
```

Integer type large enough to hold the difference between 2 addresses

11.9.3.9 GUInt16

```
typedef unsigned short GUInt16
```

Unsigned int16 type

11.9.3.10 GUInt32

```
typedef unsigned int GUInt32
```

Unsigned int32 type

11.9.3.11 GUInt64

```
typedef GUIntBig GUInt64
```

Unsigned 64 bit integer type

11.9.3.12 GUIntBig

```
typedef unsigned long long GUIntBig
```

Large unsigned integer type (generally 64-bit unsigned integer type). Use GUInt64 when exactly 64 bit is needed

11.10 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct CPLRectObj

Typedefs

- typedef struct _CPLQuadTree CPLQuadTree
- typedef void(∗ CPLQuadTreeGetBoundsFunc) (const void ∗hFeature, CPLRectObj ∗pBounds)
- typedef int(∗ CPLQuadTreeForeachFunc) (void ∗pElt, void ∗pUserData)
- typedef void(∗ CPLQuadTreeDumpFeatureFunc) (const void ∗hFeature, int nIndentLevel, void ∗pUserData)

Functions

- CPLQuadTree CPL_DLL ∗ CPLQuadTreeCreate (const CPLRectObj ∗pGlobalBounds, CPLQuadTreeGetBoundsFunc pfnGetBounds)
- void CPL_DLL CPLQuadTreeDestroy (CPLQuadTree ∗hQuadtree)
- void CPL_DLL CPLQuadTreeSetBucketCapacity (CPLQuadTree ∗hQuadtree, int nBucketCapacity)
- int CPL_DLL CPLQuadTreeGetAdvisedMaxDepth (int nExpectedFeatures)
- void CPL_DLL CPLQuadTreeSetMaxDepth (CPLQuadTree ∗hQuadtree, int nMaxDepth)
- void CPL_DLL CPLQuadTreeInsert (CPLQuadTree ∗hQuadtree, void ∗hFeature)
- void CPL_DLL CPLQuadTreeInsertWithBounds (CPLQuadTree ∗hQuadtree, void ∗hFeature, const CPLRectObj ∗psBounds)
- void CPL_DLL ∗∗ CPLQuadTreeSearch (const CPLQuadTree ∗hQuadtree, const CPLRectObj ∗pAoi, int ∗pnFeatureCount)
- void CPL_DLL CPLQuadTreeForeach (const CPLQuadTree ∗hQuadtree, CPLQuadTreeForeachFunc pfn↩Foreach, void ∗pUserData)
- void CPL_DLL CPLQuadTreeDump (const CPLQuadTree ∗hQuadtree, CPLQuadTreeDumpFeatureFunc pfnDumpFeatureFunc, void ∗pUserData)
- void CPL_DLL CPLQuadTreeGetStats (const CPLQuadTree ∗hQuadtree, int ∗pnFeatureCount, int ∗pn↩NodeCount, int ∗pnMaxDepth, int ∗pnMaxBucketCapacity)

11.10.1 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

11.10.2 Typedef Documentation

11.10.2.1 CPLQuadTree

```
typedef struct _CPLQuadTree CPLQuadTree
```

Opaque type for a quad tree

11.10.2.2 CPLQuadTreeDumpFeatureFunc

```
typedef void(* CPLQuadTreeDumpFeatureFunc) (const void *hFeature, int nIndentLevel, void *p↩
UserData)
```

CPLQuadTreeDumpFeatureFunc

11.10.2.3 CPLQuadTreeForeachFunc

```
typedef int(* CPLQuadTreeForeachFunc) (void *pElt, void *pUserData)
```

CPLQuadTreeForeachFunc

11.10.2.4 CPLQuadTreeGetBoundsFunc

```
typedef void(* CPLQuadTreeGetBoundsFunc) (const void *hFeature, CPLRectObj *pBounds)
```

CPLQuadTreeGetBoundsFunc

11.10.3 Function Documentation

11.10.3.1 CPLQuadTreeCreate()

```
CPLQuadTree CPL_DLL* CPLQuadTreeCreate (
            const CPLRectObj * pGlobalBounds,
            CPLQuadTreeGetBoundsFunc pfnGetBounds )
```

Create a new quadtree

Parameters

pGlobalBounds	a pointer to the global extent of all the elements that will be inserted
pfnGetBounds	a user provided function to get the bounding box of the inserted elements. If it is set to NULL, then CPLQuadTreeInsertWithBounds() must be used, and extra memory will be used to keep features bounds in the quad tree.

Returns

a newly allocated quadtree

11.10.3.2 CPLQuadTreeDestroy()

```
void CPL_DLL CPLQuadTreeDestroy (
            CPLQuadTree * hQuadTree )
```

Destroy a quadtree

Parameters

hQuadTree	the quad tree to destroy

11.10.3.3 CPLQuadTreeDump()

```
void CPL_DLL CPLQuadTreeDump (
            const CPLQuadTree * hQuadTree,
            CPLQuadTreeDumpFeatureFunc pfnDumpFeatureFunc,
            void * pUserData )
```

Dump quad tree

11.10.3.4 CPLQuadTreeForeach()

```
void CPL_DLL CPLQuadTreeForeach (
            const CPLQuadTree * hQuadTree,
            CPLQuadTreeForeachFunc pfnForeach,
            void * pUserData )
```

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters

hQuadTree	the quad tree
pfnForeach	the function called on each element.
pUserData	the user data provided to the function.

11.10.3.5 CPLQuadTreeGetAdvisedMaxDepth()

```
int CPL_DLL CPLQuadTreeGetAdvisedMaxDepth (
            int nExpectedFeatures )
```

Returns the optimal depth of a quadtree to hold nExpectedFeatures

Parameters

nExpectedFeatures	the expected maximum number of elements to be inserted.

Returns

the optimal depth of a quadtree to hold nExpectedFeatures

11.10.3.6 CPLQuadTreeGetStats()

```
void CPL_DLL CPLQuadTreeGetStats (
            const CPLQuadTree * hQuadTree,
            int * pnFeatureCount,
            int * pnNodeCount,
            int * pnMaxDepth,
            int * pnMaxBucketCapacity )
```

Get stats

11.10.3.7 CPLQuadTreeInsert()

```
void CPL_DLL CPLQuadTreeInsert (
            CPLQuadTree * hQuadTree,
            void * hFeature )
```

Insert a feature into a quadtree

Parameters

hQuadTree	the quad tree
hFeature	the feature to insert

11.10.3.8 CPLQuadTreeInsertWithBounds()

```
void CPL_DLL CPLQuadTreeInsertWithBounds (
            CPLQuadTree * hQuadTree,
```

```
void * hFeature,
const CPLRectObj * psBounds )
```

Insert a feature into a quadtree

Parameters

hQuadTree	the quad tree
hFeature	the feature to insert
psBounds	bounds of the feature

11.10.3.9 CPLQuadTreeSearch()

```
void CPL_DLL** CPLQuadTreeSearch (
            const CPLQuadTree * hQuadTree,
            const CPLRectObj * pAoi,
            int * pnFeatureCount )
```

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters

hQuadTree	the quad tree
pAoi	the pointer to the area of interest
pnFeatureCount	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

11.10.3.10 CPLQuadTreeSetBucketCapacity()

```
void CPL_DLL CPLQuadTreeSetBucketCapacity (
            CPLQuadTree * hQuadTree,
            int nBucketCapacity )
```

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters

hQuadTree	the quad tree
nBucketCapacity	the maximum capacity of a node of a quadtree

11.10.3.11 CPLQuadTreeSetMaxDepth()

```
void CPL_DLL CPLQuadTreeSetMaxDepth (
```

```
CPLQuadTree * hQuadTree,
int nMaxDepth )
```

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

Parameters

hQuadTree	the quad tree
nMaxDepth	the maximum depth allowed

11.11 cpl_string.h File Reference

```
#include "cpl_error.h"
#include "cpl_conv.h"
#include "cpl_vsi.h"
```

Macros

- #define CSLT_HONOURSTRINGS 0x0001
- #define CSLT_ALLOWEMPTYTOKENS 0x0002
- #define CSLT_PRESERVEQUOTES 0x0004
- #define CSLT_PRESERVEESCAPES 0x0008
- #define CSLT_STRIPLEADSPACES 0x0010
- #define CSLT_STRIPENDSPACES 0x0020
- #define CPLES_BackslashQuotable 0
- #define CPLES_XML 1
- #define CPLES_URL 2
- #define CPLES_SQL 3
- #define CPLES_CSV 4
- #define CPLES_XML_BUT_QUOTES 5
- #define CPLES_CSV_FORCE_QUOTING 6
- #define CPL_ENC_LOCALE ""
- #define CPL_ENC_UTF8 "UTF-8"
- #define CPL_ENC_UTF16 "UTF-16"
- #define CPL_ENC_UCS2 "UCS-2"
- #define CPL_ENC_UCS4 "UCS-4"
- #define CPL_ENC_ASCII "ASCII"
- #define CPL_ENC_ISO8859_1 "ISO-8859-1"

Enumerations

- enum CPLValueType { CPL_VALUE_STRING, CPL_VALUE_REAL, CPL_VALUE_INTEGER }

Functions

- CPL_C_START char CPL_DLL ** CSLAddString (char **papszStrList, const char *pszNewString) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLAddStringMayFail (char **papszStrList, const char *pszNewString) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CSLCount (CSLConstList papszStrList)
- const char CPL_DLL * CSLGetField (CSLConstList, int)
- void CPL_DLL CPL_STDCALL CSLDestroy (char **papszStrList)
- char CPL_DLL ** CSLDuplicate (CSLConstList papszStrList) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLMerge (char **papszOrig, CSLConstList papszOverride) CPL_WARN_UNUSED_RESULT

  Merge two lists.
- char CPL_DLL ** CSLTokenizeString (const char *pszString) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLTokenizeStringComplex (const char *pszString, const char *pszDelimiter, int bHonourStrings, int bAllowEmptyTokens) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLTokenizeString2 (const char *pszString, const char *pszDelimiter, int nCSLTFlags) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CSLPrint (CSLConstList papszStrList, FILE *fpOut)
- char CPL_DLL ** CSLLoad (const char *pszFname) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLLoad2 (const char *pszFname, int nMaxLines, int nMaxCols, CSLConstList papszOptions) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CSLSave (CSLConstList papszStrList, const char *pszFname)
- char CPL_DLL ** CSLInsertStrings (char **papszStrList, int nInsertAtLineNo, CSLConstList papszNewLines) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLInsertString (char **papszStrList, int nInsertAtLineNo, const char *pszNewLine) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLRemoveStrings (char **papszStrList, int nFirstLineToDelete, int nNumToRemove, char ***ppapszRetStrings) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CSLFindString (CSLConstList papszList, const char *pszTarget)
- int CPL_DLL CSLFindStringCaseSensitive (CSLConstList papszList, const char *pszTarget)
- int CPL_DLL CSLPartialFindString (CSLConstList papszHaystack, const char *pszNeedle)
- int CPL_DLL CSLFindName (CSLConstList papszStrList, const char *pszName)
- int CPL_DLL CSLFetchBoolean (CSLConstList papszStrList, const char *pszKey, int bDefault)
- int CPL_DLL CSLTestBoolean (const char *pszValue)
- int CPL_DLL CPLTestBoolean (const char *pszValue)
- const char CPL_DLL * CPLParseNameValue (const char *pszNameValue, char **ppszKey)
- const char CPL_DLL * CSLFetchNameValue (CSLConstList papszStrList, const char *pszName)
- const char CPL_DLL * CSLFetchNameValueDef (CSLConstList papszStrList, const char *pszName, const char *pszDefault)
- char CPL_DLL ** CSLFetchNameValueMultiple (CSLConstList papszStrList, const char *pszName)
- char CPL_DLL ** CSLAddNameValue (char **papszStrList, const char *pszName, const char *pszValue) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** CSLSetNameValue (char **papszStrList, const char *pszName, const char *pszValue) CPL_WARN_UNUSED_RESULT
- void CPL_DLL CSLSetNameValueSeparator (char **papszStrList, const char *pszSeparator)
- char CPL_DLL ** CSLParseCommandLine (const char *pszCommandLine)
- char CPL_DLL * CPLEscapeString (const char *pszString, int nLength, int nScheme) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * CPLUnescapeString (const char *pszString, int *pnLength, int nScheme) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * CPLBinaryToHex (int nBytes, const GByte *pabyData) CPL_WARN_UNUSED_RESULT
- GByte CPL_DLL * CPLHexToBinary (const char *pszHex, int *pnBytes) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * CPLBase64Encode (int nBytes, const GByte *pabyData) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CPLBase64DecodeInPlace (GByte *pszBase64) CPL_WARN_UNUSED_RESULT
- CPLValueType CPL_DLL CPLGetValueType (const char *pszValue)
- size_t CPL_DLL CPLStrlcpy (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t CPL_DLL CPLStrlcat (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t CPL_DLL CPLStrnlen (const char *pszStr, size_t nMaxLen)

- int CPL_DLL CPLEncodingCharSize (const char *pszEncoding)
- char CPL_DLL * CPLRecode (const char *pszSource, const char *pszSrcEncoding, const char *pszDst↩ Encoding) CPL_WARN_UNUSED_RESULT CPL_RETURNS_NONNULL
- char CPL_DLL * CPLRecodeFromWChar (const wchar_t *pwszSource, const char *pszSrcEncoding, const char *pszDstEncoding) CPL_WARN_UNUSED_RESULT
- wchar_t CPL_DLL * CPLRecodeToWChar (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CPLIsUTF8 (const char *pabyData, int nLen)
- char CPL_DLL * CPLForceToASCII (const char *pabyData, int nLen, char chReplacementChar) CPL_WARN_UNUSED_RESULT
- int CPL_DLL CPLStrlenUTF8 (const char *pszUTF8Str)

11.11.1 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(), and CSLFetchNameValue()).

To some extent the CPLStringList C++ class can be used to abstract managing string lists a bit but still be able to return them from C functions.

11.11.2 Macro Definition Documentation

11.11.2.1 CPL_ENC_ASCII

```
#define CPL_ENC_ASCII "ASCII"
```

ASCII encoding

11.11.2.2 CPL_ENC_ISO8859_1

```
#define CPL_ENC_ISO8859_1 "ISO-8859-1"
```

ISO-8859-1 (LATIN1) encoding

11.11.2.3 CPL_ENC_LOCALE

```
#define CPL_ENC_LOCALE ""
```

Encoding of the current locale

11.11.2.4 CPL_ENC_UCS2

```
#define CPL_ENC_UCS2 "UCS-2"
```

UCS-2 encoding

11.11.2.5 CPL_ENC_UCS4

```
#define CPL_ENC_UCS4 "UCS-4"
```

UCS-4 encoding

11.11.2.6 CPL_ENC_UTF16

```
#define CPL_ENC_UTF16 "UTF-16"
```

UTF-16 encoding

11.11.2.7 CPL_ENC_UTF8

```
#define CPL_ENC_UTF8 "UTF-8"
```

UTF-8 encoding

11.11.2.8 CPLES_BackslashQuotable

```
#define CPLES_BackslashQuotable 0
```

Scheme for CPLEscapeString()/CPLUnescapeString() for backlash quoting

11.11.2.9 CPLES_CSV

```
#define CPLES_CSV 4
```

Scheme for CPLEscapeString()/CPLUnescapeString() for CSV

11.11.2.10 CPLES_CSV_FORCE_QUOTING

```
#define CPLES_CSV_FORCE_QUOTING 6
```

Scheme for CPLEscapeString()/CPLUnescapeString() for CSV (forced quoting)

11.11.2.11 CPLES_SQL

```
#define CPLES_SQL 3
```

Scheme for CPLEscapeString()/CPLUnescapeString() for SQL

11.11.2.12 CPLES_URL

```
#define CPLES_URL 2
```

Scheme for CPLEscapeString()/CPLUnescapeString() for URL

11.11.2.13 CPLES_XML

```
#define CPLES_XML 1
```

Scheme for CPLEscapeString()/CPLUnescapeString() for XML

11.11.2.14 CPLES_XML_BUT_QUOTES

```
#define CPLES_XML_BUT_QUOTES 5
```

Scheme for CPLEscapeString()/CPLUnescapeString() for XML (preserves quotes)

11.11.2.15 CSLT_ALLOWEMPTYTOKENS

```
#define CSLT_ALLOWEMPTYTOKENS 0x0002
```

Flag for CSLTokenizeString2() to allow empty tokens

11.11.2.16 CSLT_HONOURSTRINGS

```
#define CSLT_HONOURSTRINGS 0x0001
```

Flag for CSLTokenizeString2() to honour strings

11.11.2.17 CSLT_PRESERVEESCAPES

```
#define CSLT_PRESERVEESCAPES 0x0008
```

Flag for CSLTokenizeString2() to preserve escape characters

11.11.2.18 CSLT_PRESERVEQUOTES

```
#define CSLT_PRESERVEQUOTES 0x0004
```

Flag for CSLTokenizeString2() to preserve quotes

11.11.2.19 CSLT_STRIPENDSPACES

```
#define CSLT_STRIPENDSPACES 0x0020
```

Flag for CSLTokenizeString2() to strip trailaing spaces

11.11.2.20 CSLT_STRIPLEADSPACES

```
#define CSLT_STRIPLEADSPACES 0x0010
```

Flag for CSLTokenizeString2() to strip leading spaces

11.11.3 Enumeration Type Documentation

11.11.3.1 CPLValueType

```
enum CPLValueType
```

Type of value

Enumerator

CPL_VALUE_STRING	String
CPL_VALUE_REAL	Real number
CPL_VALUE_INTEGER	Integer

11.11.4 Function Documentation

11.11.4.1 CPLBase64DecodeInPlace()

```
int CPL_DLL CPLBase64DecodeInPlace (
            GByte * pszBase64 )
```

Decode base64 string "pszBase64" (null terminated) in place.

Returns length of decoded array or 0 on failure.

11.11.4.2 CPLBase64Encode()

```
char CPL_DLL* CPLBase64Encode (
            int nDataLen,
            const GByte * pabyBytesToEncode )
```

Base64 encode a buffer.

11.11.4.3 CPLBinaryToHex()

```
char CPL_DLL* CPLBinaryToHex (
            int nBytes,
            const GByte * pabyData )
```

Binary to hexadecimal translation.

Parameters

nBytes	number of bytes of binary data in pabyData.
pabyData	array of data bytes to translate.

Returns

hexadecimal translation, zero terminated. Free with CPLFree().

11.11.4.4 CPLEncodingCharSize()

```
int CPL_DLL CPLEncodingCharSize (
            const char * pszEncoding )
```

Return bytes per character for encoding.

This function returns the size in bytes of the smallest character in this encoding. For fixed width encodings (ASCII, UCS-2, UCS-4) this is straight forward. For encodings like UTF8 and UTF16 which represent some characters as a sequence of atomic character sizes the function still returns the atomic character size (1 for UTF8, 2 for UTF16).

This function will return the correct value for well known encodings with corresponding CPL_ENC_ values. It may not return the correct value for other encodings even if they are supported by the underlying iconv or windows transliteration services. Hopefully it will improve over time.

Parameters

pszEncoding	the name of the encoding.

Returns

the size of a minimal character in bytes or -1 if the size is unknown.

11.11.4.5 CPLEscapeString()

```
char CPL_DLL* 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 its 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 alphanumerics and the characters '$', '-', '_', '.', '+', '!', '*', '"', '(', ')' and ',' (see RFC1738) 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() currently does not support this format, only CPLEscapeString(). See cpl_csv.cpp for CSV parsing support.

Parameters

pszInput	the string to escape.
nLength	The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.
nScheme	the encoding scheme to use.

Returns

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

11.11.4.6 CPLForceToASCII()

```
char CPL_DLL* CPLForceToASCII (
            const char * pabyData,
            int nLen,
            char chReplacementChar )
```

Return a new string that is made only of ASCII characters. If non-ASCII characters are found in the input string, they will be replaced by the provided replacement character.

Parameters

pabyData	input string to test
nLen	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
chReplacementChar	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

11.11.4.7 CPLGetValueType()

```
CPLValueType CPL_DLL CPLGetValueType (
            const char * pszValue )
```

Detect the type of the value contained in a string, whether it is a real, an integer or a string Leading and trailing spaces are skipped in the analysis.

Note: in the context of this function, integer must be understood in a broad sense. It does not mean that the value can fit into a 32 bit integer for example. It might be larger.

Parameters

pszValue	the string to analyze

Returns

returns the type of the value contained in the string.

11.11.4.8 CPLHexToBinary()

```
GByte CPL_DLL* CPLHexToBinary (
            const char * pszHex,
            int * pnBytes )
```

Hexadecimal to binary translation

Parameters

pszHex	the input hex encoded string.
pnBytes	the returned count of decoded bytes placed here.

Returns

returns binary buffer of data - free with CPLFree().

11.11.4.9 CPLIsUTF8()

```
int CPL_DLL CPLIsUTF8 (
            const char * pabyData,
            int nLen )
```

Test if a string is encoded as UTF-8.

Parameters

pabyData	input string to test
nLen	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.

Returns

TRUE if the string is encoded as UTF-8. FALSE otherwise

Since

GDAL 1.7.0

11.11.4.10 CPLParseNameValue()

```
const char CPL_DLL* CPLParseNameValue (
            const char * pszNameValue,
            char ** ppszKey )
```

Parse NAME=VALUE string into name and value components.

Note that if ppszKey is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimiter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use CPLParseNameValue().

Parameters

pszNameValue	string in "NAME=VALUE" format.
ppszKey	optional pointer though which to return the name portion.

Returns

the value portion (pointing into original string).

11.11.4.11 CPLRecode()

```
char CPL_DLL* 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().

Parameters

pszSource	a NULL terminated string.
pszSrcEncoding	the source encoding.
pszDstEncoding	the destination encoding.

Returns

a NULL terminated string which should be freed with CPLFree().

Since

GDAL 1.6.0

11.11.4.12 CPLRecodeFromWChar()

```
char CPL_DLL* 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_E↩NC_UCS2, and the only guaranteed supported destination encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8859_1. In some cases (i.e. 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().

Parameters

pwszSource	the source wchar_t string, terminated with a 0 wchar_t.
pszSrcEncoding	the source encoding, typically CPL_ENC_UCS2.
pszDstEncoding	the destination encoding, typically CPL_ENC_UTF8.

Returns

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

Since

GDAL 1.6.0

11.11.4.13 CPLRecodeToWChar()

```
wchar_t CPL_DLL* 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().

Parameters

pszSource	input multi-byte character string.
pszSrcEncoding	source encoding, typically CPL_ENC_UTF8.
pszDstEncoding	destination encoding, typically CPL_ENC_UCS2.

Returns

the zero terminated wchar_t string (to be freed with CPLFree()) or NULL on error.

Since

GDAL 1.6.0

11.11.4.14 CPLStrlcat()

```
size_t CPL_DLL CPLStrlcat (
            char * pszDest,
            const char * pszSrc,
            size_t nDestSize )
```

Appends a source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1 and that there is at least one byte free in pszDest, that is to say strlen(pszDest_before) < nDestSize)

This function is designed to be a safer, more consistent, and less error prone replacement for strncat. Its contract is identical to libbsd's strlcat.

Truncation can be detected by testing if the return value of CPLStrlcat is greater or equal to nDestSize.

```
char szDest[5] = {};
CPLStrlcpy(szDest, "ab", sizeof(szDest));
if( CPLStrlcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest) )
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

pszDest	destination buffer. Must be NUL terminated before running CPLStrlcat
pszSrc	source string. Must be NUL terminated
nDestSize	size of destination buffer (including space for the NUL terminator character)

Returns

the 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

11.11.4.15 CPLStrlcpy()

```
size_t CPL_DLL CPLStrlcpy (
            char * pszDest,
            const char * pszSrc,
            size_t nDestSize )
```

Copy source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1).

This function is designed to be a safer, more consistent, and less error prone replacement for strncpy. Its contract is identical to libbsd's strlcpy.

Truncation can be detected by testing if the return value of CPLStrlcpy is greater or equal to nDestSize.

```
char szDest[5] = {};
if( CPLStrlcpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest) )
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

pszDest	destination buffer
pszSrc	source string. Must be NUL terminated
nDestSize	size of destination buffer (including space for the NUL terminator character)

Returns

the length of the source string (=strlen(pszSrc))

Since

GDAL 1.7.0

11.11.4.16 CPLStrlenUTF8()

```
int CPL_DLL CPLStrlenUTF8 (
            const char * pszUTF8Str )
```

Return the number of UTF-8 characters of a nul-terminated string.

This is different from strlen() which returns the number of bytes.

Parameters

pszUTF8Str	a nul-terminated UTF-8 string

Returns

the number of UTF-8 characters.

11.11.4.17 CPLStrnlen()

```
size_t CPL_DLL 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() function returns min(strlen(pszStr), nMaxLen). Only the first nMaxLen bytes of the string will be read. Useful to test if a string contains at least nMaxLen characters without reading the full string up to the NUL terminating character.

Parameters

pszStr	a NUL terminated string
nMaxLen	maximum number of bytes to read in pszStr

Returns

strlen(pszStr) if the length is lesser than nMaxLen, otherwise nMaxLen if the NUL character has not been found in the first nMaxLen bytes.

Since

GDAL 1.7.0

11.11.4.18 CPLTestBoolean()

```
int CPL_DLL CPLTestBoolean (
            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.

Use this only in C code. In C++, prefer CPLTestBool().

Parameters

pszValue	the string should be tested.

Returns

TRUE or FALSE.

11.11.4.19 CPLUnescapeString()

```
char CPL_DLL* CPLUnescapeString (
            const char * pszInput,
            int * pnLength,
            int nScheme )
```

Unescape a string.

This function does the opposite of CPLEscapeString(). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to its original form.

Parameters

pszInput	the input string. This is a zero terminated string.
pnLength	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
nScheme	the escaped scheme to undo (see CPLEscapeString() for a list). Does not yet support CSV.

Returns

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

11.11.4.20 CSLAddNameValue()

```
char CPL_DLL** CSLAddNameValue (
            char ** papszStrList,
            const char * pszName,
            const char * pszValue )
```

Add a new entry to a StringList of "Name=Value" pairs, ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

This function does not check if a "Name=Value" pair already exists for that name and can generate multiple entries for the same name. Use CSLSetNameValue() if you want each name to have only one value.

Returns the modified StringList.

11.11.4.21 CSLAddString()

```
CPL_C_START char CPL_DLL** CSLAddString (
            char ** papszStrList,
            const char * pszNewString )
```

Append a string to a StringList and return a pointer to the modified StringList.

If the input StringList is NULL, then a new StringList is created. Note that CSLAddString performance when building a list is in O(n^2) which can cause noticeable slow down when n > 10000.

11.11.4.22 CSLAddStringMayFail()

```
char CPL_DLL** CSLAddStringMayFail (
            char ** papszStrList,
            const char * pszNewString )
```

Same as CSLAddString() but may return NULL in case of (memory) failure

11.11.4.23 CSLCount()

```
int CPL_DLL CSLCount (
            CSLConstList 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() as an end condition for loops as it will result in O(n^2) behavior.

Parameters

papszStrList	the string list to count.

Returns

the number of entries.

11.11.4.24 CSLDestroy()

```
void CPL_DLL CPL_STDCALL CSLDestroy (
            char ** papszStrList )
```

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters

papszStrList	the list to free.

11.11.4.25 CSLDuplicate()

```
char CPL_DLL** CSLDuplicate (
            CSLConstList 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().

Parameters

papszStrList	the input string list.

Returns

newly allocated copy.

11.11.4.26 CSLFetchBoolean()

```
int CPL_DLL CSLFetchBoolean (
            CSLConstList papszStrList,
```

```
            const char * pszKey,
            int bDefault )
```

DEPRECATED. Check for boolean key value.

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

Parameters

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

Returns

TRUE or FALSE

11.11.4.27 CSLFetchNameValue()

```
const char CPL_DLL* CSLFetchNameValue (
            CSLConstList papszStrList,
            const char * pszName )
```

In a StringList of "Name=Value" pairs, look for the first value associated with the specified name. The search is not case sensitive. ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

Returns a reference to the value in the StringList that the caller should not attempt to free.

Returns NULL if the name is not found.

11.11.4.28 CSLFetchNameValueDef()

```
const char CPL_DLL* CSLFetchNameValueDef (
            CSLConstList papszStrList,
            const char * pszName,
            const char * pszDefault )
```

Same as CSLFetchNameValue() but return pszDefault in case of no match

11.11.4.29 CSLFetchNameValueMultiple()

```
char CPL_DLL** CSLFetchNameValueMultiple (
            CSLConstList papszStrList,
            const char * pszName )
```

In a StringList of "Name=Value" pairs, look for all the values with the specified name. The search is not case sensitive. ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

Returns StringList with one entry for each occurrence of the specified name. The StringList should eventually be destroyed by calling CSLDestroy().

Returns NULL if the name is not found.

11.11.4.30 CSLFindName()

```
int CPL_DLL CSLFindName (
            CSLConstList papszStrList,
            const char * pszName )
```

Find StringList entry with given key name.

Parameters

papszStrList	the string list to search.
pszName	the key value to look for (case insensitive).

Returns

-1 on failure or the list index of the first occurrence matching the given key.

11.11.4.31 CSLFindString()

```
int CPL_DLL CSLFindString (
            CSLConstList papszList,
            const char * pszTarget )
```

Find a string within a string list (case insensitive).

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

Parameters

papszList	the string list to be searched.
pszTarget	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

11.11.4.32 CSLFindStringCaseSensitive()

```
int CPL_DLL CSLFindStringCaseSensitive (
            CSLConstList papszList,
            const char * pszTarget )
```

Find a string within a string list(case sensitive)

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.

Parameters

papszList	the string list to be searched.
pszTarget	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

Since

GDAL 2.0

11.11.4.33 CSLGetField()

```
const char CPL_DLL* CSLGetField (
            CSLConstList papszStrList,
            int iField )
```

Fetches the indicated field, being careful not to crash if the field doesn't exist within this string list.

The returned pointer should not be freed, and doesn't necessarily last long.

11.11.4.34 CSLInsertString()

```
char CPL_DLL** CSLInsertString (
            char ** papszStrList,
            int nInsertAtLineNo,
            const char * pszNewLine )
```

Insert a string at a given line number inside a StringList

nInsertAtLineNo is a 0-based line index before which the new string should be inserted. If this value is -1 or is larger than the actual number of strings in the list then the string is added at the end of the source StringList.

Returns the modified StringList.

11.11.4.35 CSLInsertStrings()

```
char CPL_DLL** CSLInsertStrings (
            char ** papszStrList,
            int nInsertAtLineNo,
            CSLConstList papszNewLines )
```

Copies the contents of a StringList inside another StringList before the specified line.

nInsertAtLineNo is a 0-based line index before which the new strings should be inserted. If this value is -1 or is larger than the actual number of strings in the list then the strings are added at the end of the source StringList.

Returns the modified StringList.

11.11.4.36 CSLLoad()

```
char CPL_DLL** CSLLoad (
            const char * pszFname )
```

Load a text file into a string list.

The VSI∗L API is used, so VSIFOpenL() 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()).

If reading the file fails a CPLError() will be issued and NULL returned.

Parameters

pszFname	the name of the file to read.

Returns

a string list with the files lines, now owned by caller. To be freed with CSLDestroy()

11.11.4.37 CSLLoad2()

```
char CPL_DLL** CSLLoad2 (
            const char * pszFname,
            int nMaxLines,
            int nMaxCols,
            CSLConstList papszOptions )
```

Load a text file into a string list.

The VSI∗L API is used, so VSIFOpenL() 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()).

If reading the file fails a CPLError() will be issued and NULL returned.

Parameters

pszFname	the name of the file to read.
nMaxLines	maximum number of lines to read before stopping, or -1 for no limit.
nMaxCols	maximum number of characters in a line before stopping, or -1 for no limit.
papszOptions	NULL-terminated array of options. Unused for now.

Returns

a string list with the files lines, now owned by caller. To be freed with CSLDestroy()

Since

GDAL 1.7.0

11.11.4.38 CSLMerge()

```
char CPL_DLL** CSLMerge (
            char ** papszOrig,
            CSLConstList 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

papszOrig	the original list, being modified.
papszOverride	the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns

updated list.

11.11.4.39 CSLParseCommandLine()

```
char CPL_DLL** CSLParseCommandLine (
            const char * pszCommandLine )
```

Tokenize command line arguments in a list of strings.

Parameters

pszCommandLine	command line

Returns

NULL terminated list of strings to free with CSLDestroy()

Since

GDAL 2.1

11.11.4.40 CSLPartialFindString()

```
int CPL_DLL CSLPartialFindString (
            CSLConstList 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()).

Parameters

papszHaystack	the string list to be searched.
pszNeedle	the substring to be searched for.

Returns

the index of the string within the list or -1 on failure.

11.11.4.41 CSLPrint()

```
int CPL_DLL CSLPrint (
            CSLConstList papszStrList,
            FILE * fpOut )
```

Print a StringList to fpOut. If fpOut==NULL, then output is sent to stdout.

Returns the number of lines printed.

11.11.4.42 CSLRemoveStrings()

```
char CPL_DLL** CSLRemoveStrings (
            char ** papszStrList,
            int nFirstLineToDelete,
            int nNumToRemove,
            char *** ppapszRetStrings )
```

Remove strings inside a StringList

nFirstLineToDelete is the 0-based line index of the first line to remove. If this value is -1 or is larger than the actual number of strings in list then the nNumToRemove last strings are removed.

If ppapszRetStrings != NULL then the deleted strings won't be free'd, they will be stored in a new StringList and the pointer to this new list will be returned in *ppapszRetStrings.

Returns the modified StringList.

11.11.4.43 CSLSave()

```
int CPL_DLL CSLSave (
            CSLConstList papszStrList,
            const char * pszFname )
```

Write a StringList to a text file.

Returns the number of lines written, or 0 if the file could not be written.

11.11.4.44 CSLSetNameValue()

```
char CPL_DLL** CSLSetNameValue (
            char ** papszList,
            const char * pszName,
            const char * pszValue )
```

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

Parameters

papszList	the original list, the modified version is returned.
pszName	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
pszValue	the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

Returns

modified StringList.

11.11.4.45 CSLSetNameValueSeparator()

```
void CPL_DLL 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 manipulable by the CSL name/value functions any more.

The CPLParseNameValue() 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. e.g. ": " or " = ".

Parameters

papszList	the list to update. Component strings may be freed but the list array will remain at the same location.
pszSeparator	the new separator string to insert.

11.11.4.46 CSLTestBoolean()

```
int CPL_DLL 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.

Deprecated. Removed in GDAL 3.x.

Use CPLTestBoolean() for C and CPLTestBool() for C++.

Parameters

pszValue	the string should be tested.

Returns

TRUE or FALSE.

11.11.4.47 CSLTokenizeString()

```
char CPL_DLL** CSLTokenizeString (
            const char * pszString )
```

Tokenizes a string and returns a StringList with one string for each token.

11.11.4.48 CSLTokenizeString2()

```
char CPL_DLL** 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() 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 =
    CSLTokenizeString2( pszCommand, " \t\n",
                        CSLT_HONOURSTRINGS |
      CSLT_ALLOWEMPTYTOKENS );

for( int i = 0; papszTokens != NULL && papszTokens[i] != NULL; ++i )
    printf( "arg %d: '%s'", papszTokens[i] );  // ok

CSLDestroy( papszTokens );
```

Parameters

pszString	the string to be split into tokens.
pszDelimiters	one or more characters to be used as token delimiters.
nCSLTFlags	an ORing of one or more of the CSLT_ flag values.

Returns

a string list of tokens owned by the caller.

11.11.4.49 CSLTokenizeStringComplex()

```
char CPL_DLL** CSLTokenizeStringComplex (
            const char * pszString,
            const char * pszDelimiters,
            int bHonourStrings,
            int bAllowEmptyTokens )
```

Obsolete tokenizing api. Use CSLTokenizeString2()

11.12 cpl_virtualmem.h File Reference

```
#include <stddef.h>
#include "cpl_port.h"
#include "cpl_vsi.h"
```

Typedefs

- typedef struct CPLVirtualMem CPLVirtualMem
- typedef void(∗ CPLVirtualMemCachePageCbk) (CPLVirtualMem ∗ctxt, size_t nOffset, void ∗pPageToFill, size_t nToFill, void ∗pUserData)
- typedef void(∗ CPLVirtualMemUnCachePageCbk) (CPLVirtualMem ∗ctxt, size_t nOffset, const void ∗p↩ PageToBeEvicted, size_t nToBeEvicted, void ∗pUserData)
- typedef void(∗ CPLVirtualMemFreeUserData) (void ∗pUserData)

Enumerations

- enum CPLVirtualMemAccessMode { VIRTUALMEM_READONLY, VIRTUALMEM_READONLY_ENFORCED, VIRTUALMEM_READWRITE }

Functions

- size_t CPL_DLL CPLGetPageSize (void)
- CPLVirtualMem CPL_DLL * CPLVirtualMemNew (size_t nSize, size_t nCacheSize, size_t nPageSizeHint, int bSingleThreadUsage, CPLVirtualMemAccessMode eAccessMode, CPLVirtualMemCachePageCbk pfn↩CachePage, CPLVirtualMemUnCachePageCbk pfnUnCachePage, CPLVirtualMemFreeUserData pfnFree↩UserData, void *pCbkUserData)
- int CPL_DLL CPLIsVirtualMemFileMapAvailable (void)
- CPLVirtualMem CPL_DLL * CPLVirtualMemFileMapNew (VSILFILE *fp, vsi_l_offset nOffset, vsi_l_offset nLength, CPLVirtualMemAccessMode eAccessMode, CPLVirtualMemFreeUserData pfnFreeUserData, void *pCbkUserData)
- CPLVirtualMem CPL_DLL * CPLVirtualMemDerivedNew (CPLVirtualMem *pVMemBase, vsi_l_offset n↩Offset, vsi_l_offset nSize, CPLVirtualMemFreeUserData pfnFreeUserData, void *pCbkUserData)
- void CPL_DLL CPLVirtualMemFree (CPLVirtualMem *ctxt)
- void CPL_DLL * CPLVirtualMemGetAddr (CPLVirtualMem *ctxt)
- size_t CPL_DLL CPLVirtualMemGetSize (CPLVirtualMem *ctxt)
- int CPL_DLL CPLVirtualMemIsFileMapping (CPLVirtualMem *ctxt)
- CPLVirtualMemAccessMode CPL_DLL CPLVirtualMemGetAccessMode (CPLVirtualMem *ctxt)
- size_t CPL_DLL CPLVirtualMemGetPageSize (CPLVirtualMem *ctxt)
- int CPL_DLL CPLVirtualMemIsAccessThreadSafe (CPLVirtualMem *ctxt)
- void CPL_DLL CPLVirtualMemDeclareThread (CPLVirtualMem *ctxt)
- void CPL_DLL CPLVirtualMemUnDeclareThread (CPLVirtualMem *ctxt)
- void CPL_DLL CPLVirtualMemPin (CPLVirtualMem *ctxt, void *pAddr, size_t nSize, int bWriteOp)
- void CPL_DLL CPLVirtualMemManagerTerminate (void)

11.12.1 Detailed Description

Virtual memory management.

This file provides mechanism to define virtual memory mappings, whose content is allocated transparently and filled on-the-fly. Those virtual memory mappings can be much larger than the available RAM, but only parts of the virtual memory mapping, in the limit of the allowed the cache size, will actually be physically allocated.

This exploits low-level mechanisms of the operating system (virtual memory allocation, page protection and handler of virtual memory exceptions).

It is also possible to create a virtual memory mapping from a file or part of a file.

The current implementation is Linux only.

11.12.2 Typedef Documentation

11.12.2.1 CPLVirtualMem

```
typedef struct CPLVirtualMem CPLVirtualMem
```

Opaque type that represents a virtual memory mapping.

11.12.2.2 CPLVirtualMemCachePageCbk

```
typedef void(* CPLVirtualMemCachePageCbk) (CPLVirtualMem *ctxt, size_t nOffset, void *pPageTo↩
Fill, size_t nToFill, void *pUserData)
```

Callback triggered when a still unmapped page of virtual memory is accessed. The callback has the responsibility of filling the page with relevant values

Parameters

ctxt	virtual memory handle.
nOffset	offset of the page in the memory mapping.
pPageToFill	address of the page to fill. Note that the address might be a temporary location, and not at CPLVirtualMemGetAddr() + nOffset.
nToFill	number of bytes of the page.
pUserData	user data that was passed to CPLVirtualMemNew().

11.12.2.3 CPLVirtualMemFreeUserData

```
typedef void(* CPLVirtualMemFreeUserData) (void *pUserData)
```

Callback triggered when a virtual memory mapping is destroyed.

Parameters

pUserData	user data that was passed to CPLVirtualMemNew().

11.12.2.4 CPLVirtualMemUnCachePageCbk

```
typedef void(* CPLVirtualMemUnCachePageCbk) (CPLVirtualMem *ctxt, size_t nOffset, const void
*pPageToBeEvicted, size_t nToBeEvicted, void *pUserData)
```

Callback triggered when a dirty mapped page is going to be freed. (saturation of cache, or termination of the virtual memory mapping).

Parameters

ctxt	virtual memory handle.
nOffset	offset of the page in the memory mapping.
pPageToBeEvicted	address of the page that will be flushed. Note that the address might be a temporary location, and not at CPLVirtualMemGetAddr() + nOffset.
nToBeEvicted	number of bytes of the page.
pUserData	user data that was passed to CPLVirtualMemNew().

11.12.3 Enumeration Type Documentation

11.12.3.1 CPLVirtualMemAccessMode

```
enum CPLVirtualMemAccessMode
```

Access mode of a virtual memory mapping.

Enumerator

VIRTUALMEM_READONLY	The mapping is meant at being read-only, but writes will not be prevented. Note that any content written will be lost.
VIRTUALMEM_READONLY_ENFORCED	The mapping is meant at being read-only, and this will be enforced through the operating system page protection mechanism.
VIRTUALMEM_READWRITE	The mapping is meant at being read-write, and modified pages can be saved thanks to the pfnUnCachePage callback

11.12.4 Function Documentation

11.12.4.1 CPLGetPageSize()

```
size_t CPL_DLL CPLGetPageSize (
            void  )
```

Return the size of a page of virtual memory.

Returns

the page size.

Since

GDAL 1.11

11.12.4.2 CPLIsVirtualMemFileMapAvailable()

```
int CPL_DLL CPLIsVirtualMemFileMapAvailable (
            void  )
```

Return if virtual memory mapping of a file is available.

Returns

TRUE if virtual memory mapping of a file is available.

Since

GDAL 1.11

11.12.4.3 CPLVirtualMemDeclareThread()

```
void CPL_DLL CPLVirtualMemDeclareThread (
            CPLVirtualMem * ctxt )
```

Declare that a thread will access a virtual memory mapping.

This function must be called by a thread that wants to access the content of a virtual memory mapping, except if the virtual memory mapping has been created with bSingleThreadUsage = TRUE.

This function must be paired with CPLVirtualMemUnDeclareThread().

Parameters

ctxt	context returned by CPLVirtualMemNew().

Since

GDAL 1.11

11.12.4.4 CPLVirtualMemDerivedNew()

```
CPLVirtualMem CPL_DLL* CPLVirtualMemDerivedNew (
            CPLVirtualMem * pVMemBase,
            vsi_l_offset nOffset,
            vsi_l_offset nSize,
            CPLVirtualMemFreeUserData pfnFreeUserData,
            void * pCbkUserData )
```

Create a new virtual memory mapping derived from an other virtual memory mapping.

This may be useful in case of creating mapping for pixel interleaved data.

The new mapping takes a reference on the base mapping.

Parameters

pVMemBase	Base virtual memory mapping
nOffset	Offset in the base virtual memory mapping from which to start the new mapping.
nSize	Size of the base virtual memory mapping to expose in the the new mapping.
pfnFreeUserData	callback that is called when the object is destroyed.
pCbkUserData	user data passed to pfnFreeUserData.

Returns

a virtual memory object that must be freed by CPLVirtualMemFree(), or NULL in case of failure.

Since

GDAL 1.11

11.12.4.5 CPLVirtualMemFileMapNew()

```
CPLVirtualMem CPL_DLL* CPLVirtualMemFileMapNew (
            VSILFILE * fp,
            vsi_l_offset nOffset,
            vsi_l_offset nLength,
            CPLVirtualMemAccessMode eAccessMode,
            CPLVirtualMemFreeUserData pfnFreeUserData,
            void * pCbkUserData )
```

Create a new virtual memory mapping from a file.

The file must be a "real" file recognized by the operating system, and not a VSI extended virtual file.

In VIRTUALMEM_READWRITE mode, updates to the memory mapping will be written in the file.

On Linux AMD64 platforms, the maximum value for nLength is 128 TB. On Linux x86 platforms, the maximum value for nLength is 2 GB.

Supported on Linux only in GDAL <= 2.0, and all POSIX systems supporting mmap() in GDAL >= 2.1

Parameters

fp	Virtual file handle.
nOffset	Offset in the file to start the mapping from.
nLength	Length of the portion of the file to map into memory.
eAccessMode	Permission to use for the virtual memory mapping. This must be consistent with how the file has been opened.
pfnFreeUserData	callback that is called when the object is destroyed.
pCbkUserData	user data passed to pfnFreeUserData.

Returns

a virtual memory object that must be freed by CPLVirtualMemFree(), or NULL in case of failure.

Since

GDAL 1.11

11.12.4.6 CPLVirtualMemFree()

```
void CPL_DLL CPLVirtualMemFree (
            CPLVirtualMem * ctxt )
```

Free a virtual memory mapping.

The pointer returned by CPLVirtualMemGetAddr() will no longer be valid. If the virtual memory mapping was created with read/write permissions and that they are dirty (i.e. modified) pages, they will be flushed through the pfnUnCachePage callback before being freed.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Since

GDAL 1.11

11.12.4.7 CPLVirtualMemGetAccessMode()

```
CPLVirtualMemAccessMode CPL_DLL CPLVirtualMemGetAccessMode (
            CPLVirtualMem * ctxt )
```

Return the access mode of the virtual memory mapping.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

the access mode of the virtual memory mapping.

Since

GDAL 1.11

11.12.4.8 CPLVirtualMemGetAddr()

```
void CPL_DLL* CPLVirtualMemGetAddr (
            CPLVirtualMem * ctxt )
```

Return the pointer to the start of a virtual memory mapping.

The bytes in the range [p:p+CPLVirtualMemGetSize()-1] where p is the pointer returned by this function will be valid, until CPLVirtualMemFree() is called.

Note that if a range of bytes used as an argument of a system call (such as read() or write()) contains pages that have not been "realized", the system call will fail with EFAULT. CPLVirtualMemPin() can be used to work around this issue.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

the pointer to the start of a virtual memory mapping.

Since

GDAL 1.11

11.12.4.9 CPLVirtualMemGetPageSize()

```
size_t CPL_DLL CPLVirtualMemGetPageSize (
            CPLVirtualMem * ctxt )
```

Return the page size associated to a virtual memory mapping.

The value returned will be at least CPLGetPageSize(), but potentially larger.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

the page size

Since

GDAL 1.11

11.12.4.10 CPLVirtualMemGetSize()

```
size_t CPL_DLL CPLVirtualMemGetSize (
            CPLVirtualMem * ctxt )
```

Return the size of the virtual memory mapping.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

the size of the virtual memory mapping.

Since

GDAL 1.11

11.12.4.11 CPLVirtualMemIsAccessThreadSafe()

```
int CPL_DLL CPLVirtualMemIsAccessThreadSafe (
            CPLVirtualMem * ctxt )
```

Return TRUE if this memory mapping can be accessed safely from concurrent threads.

The situation that can cause problems is when several threads try to access a page of the mapping that is not yet mapped.

The return value of this function depends on whether bSingleThreadUsage has been set of not in CPLVirtualMemNew() and/or the implementation.

On Linux, this will always return TRUE if bSingleThreadUsage = FALSE.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

TRUE if this memory mapping can be accessed safely from concurrent threads.

Since

GDAL 1.11

11.12.4.12 CPLVirtualMemIsFileMapping()

```
int CPL_DLL CPLVirtualMemIsFileMapping (
            CPLVirtualMem * ctxt )
```

Return if the virtual memory mapping is a direct file mapping.

Parameters

ctxt	context returned by CPLVirtualMemNew().

Returns

TRUE if the virtual memory mapping is a direct file mapping.

Since

GDAL 1.11

11.12.4.13 CPLVirtualMemManagerTerminate()

```
void CPL_DLL CPLVirtualMemManagerTerminate (
            void  )
```

Cleanup any resource and handlers related to virtual memory.

This function must be called after the last CPLVirtualMem object has been freed.

Since

GDAL 2.0

11.12.4.14 CPLVirtualMemNew()

```
CPLVirtualMem CPL_DLL* CPLVirtualMemNew (
            size_t nSize,
            size_t nCacheSize,
            size_t nPageSizeHint,
            int bSingleThreadUsage,
            CPLVirtualMemAccessMode eAccessMode,
            CPLVirtualMemCachePageCbk pfnCachePage,
            CPLVirtualMemUnCachePageCbk pfnUnCachePage,
            CPLVirtualMemFreeUserData pfnFreeUserData,
            void * pCbkUserData )
```

Create a new virtual memory mapping.

This will reserve an area of virtual memory of size nSize, whose size might be potentially much larger than the physical memory available. Initially, no physical memory will be allocated. As soon as memory pages will be accessed, they will be allocated transparently and filled with the pfnCachePage callback. When the allowed cache size is reached, the least recently used pages will be unallocated.

On Linux AMD64 platforms, the maximum value for nSize is 128 TB. On Linux x86 platforms, the maximum value for nSize is 2 GB.

Only supported on Linux for now.

Note that on Linux, this function will install a SIGSEGV handler. The original handler will be restored by CPLVirtualMemManagerTerminate().

Parameters

nSize	size in bytes of the virtual memory mapping.
nCacheSize	size in bytes of the maximum memory that will be really allocated (must ideally fit into RAM).
nPageSizeHint	hint for the page size. Must be a multiple of the system page size, returned by CPLGetPageSize(). Minimum value is generally 4096. Might be set to 0 to let the function determine a default page size.
bSingleThreadUsage	set to TRUE if there will be no concurrent threads that will access the virtual memory mapping. This can optimize performance a bit.
eAccessMode	permission to use for the virtual memory mapping.
pfnCachePage	callback triggered when a still unmapped page of virtual memory is accessed. The callback has the responsibility of filling the page with relevant values.
pfnUnCachePage	callback triggered when a dirty mapped page is going to be freed (saturation of cache, or termination of the virtual memory mapping). Might be NULL.
pfnFreeUserData	callback that can be used to free pCbkUserData. Might be NULL
pCbkUserData	user data passed to pfnCachePage and pfnUnCachePage.

Returns

a virtual memory object that must be freed by CPLVirtualMemFree(), or NULL in case of failure.

Since

GDAL 1.11

11.12.4.15 CPLVirtualMemPin()

```
void CPL_DLL CPLVirtualMemPin (
            CPLVirtualMem * ctxt,
            void * pAddr,
            size_t nSize,
            int bWriteOp )
```

Make sure that a region of virtual memory will be realized.

Calling this function is not required, but might be useful when debugging a process with tools like gdb or valgrind that do not naturally like segmentation fault signals.

It is also needed when wanting to provide part of virtual memory mapping to a system call such as read() or write(). If read() or write() is called on a memory region not yet realized, the call will fail with EFAULT.

Parameters

ctxt	context returned by CPLVirtualMemNew().
pAddr	the memory region to pin.
nSize	the size of the memory region.
bWriteOp	set to TRUE if the memory are will be accessed in write mode.

Since

GDAL 1.11

11.12.4.16 CPLVirtualMemUnDeclareThread()

```
void CPL_DLL CPLVirtualMemUnDeclareThread (
            CPLVirtualMem * ctxt )
```

Declare that a thread will stop accessing a virtual memory mapping.

This function must be called by a thread that will no longer access the content of a virtual memory mapping, except if the virtual memory mapping has been created with bSingleThreadUsage = TRUE.

This function must be paired with CPLVirtualMemDeclareThread().

Parameters

ctxt	context returned by CPLVirtualMemNew().

Since

GDAL 1.11

11.13 cpl_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

Macros

- #define VSI_ISLNK(x) S_ISLNK(x)
- #define VSI_ISREG(x) S_ISREG(x)
- #define VSI_ISDIR(x) S_ISDIR(x)
- #define VSI_L_OFFSET_MAX GUINTBIG_MAX
- #define VSI_STAT_EXISTS_FLAG 0x1
- #define VSI_STAT_NATURE_FLAG 0x2
- #define VSI_STAT_SIZE_FLAG 0x4
- #define VSI_STAT_SET_ERROR_FLAG 0x8
- #define VSI_MALLOC_ALIGNED_AUTO_VERBOSE(size) VSIMallocAlignedAutoVerbose(size,__FILE__,↩ __LINE__)
- #define VSI_MALLOC_VERBOSE(size) VSIMallocVerbose(size,__FILE__,__LINE__)
- #define VSI_MALLOC2_VERBOSE(nSize1, nSize2) VSIMalloc2Verbose(nSize1,nSize2,__FILE__,__LIN↩ E__)
- #define VSI_MALLOC3_VERBOSE(nSize1, nSize2, nSize3) VSIMalloc3Verbose(nSize1,nSize2,nSize3,_↩ _FILE__,__LINE__)

- #define VSI_CALLOC_VERBOSE(nCount, nSize) VSICallocVerbose(nCount,nSize,__FILE__,__LINE__)
- #define VSI_REALLOC_VERBOSE(pOldPtr, nNewSize) VSIReallocVerbose(pOldPtr,nNewSize,__FILE__↩ ,__LINE__)
- #define VSI_STRDUP_VERBOSE(pszStr) VSIStrdupVerbose(pszStr,__FILE__,__LINE__)
- #define CPLReadDir VSIReadDir

Typedefs

- typedef GUIntBig vsi_l_offset
- typedef FILE VSILFILE
- typedef struct VSI_STAT64_T VSIStatBufL
- typedef size_t(∗ VSIWriteFunction) (const void ∗ptr, size_t size, size_t nmemb, FILE ∗stream)

Enumerations

- enum VSIRangeStatus { VSI_RANGE_STATUS_UNKNOWN, VSI_RANGE_STATUS_DATA, VSI_RANGE_STATUS_HOLE }

Functions

- VSILFILE CPL_DLL ∗ VSIFOpenL (const char ∗, const char ∗) CPL_WARN_UNUSED_RESULT

  Open file.
- VSILFILE CPL_DLL ∗ VSIFOpenExL (const char ∗, const char ∗, int) CPL_WARN_UNUSED_RESULT

  Open file.
- int CPL_DLL VSIFCloseL (VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Close file.
- int CPL_DLL VSIFSeekL (VSILFILE ∗, vsi_l_offset, int) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Seek to requested offset.
- vsi_l_offset CPL_DLL VSIFTellL (VSILFILE ∗) CPL_WARN_UNUSED_RESULT

  Tell current file offset.
- void CPL_DLL VSIRewindL (VSILFILE ∗)

  Rewind the file pointer to the beginning of the file.
- size_t CPL_DLL VSIFReadL (void ∗, size_t, size_t, VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSE↩ D_RESULT

  Read bytes from file.
- int CPL_DLL VSIFReadMultiRangeL (int nRanges, void ∗∗ppData, const vsi_l_offset ∗panOffsets, const size_t ∗panSizes, VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Read several ranges of bytes from file.
- size_t CPL_DLL VSIFWriteL (const void ∗, size_t, size_t, VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UN↩ USED_RESULT

  Write bytes to file.
- int CPL_DLL VSIFEofL (VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Test for end of file.
- int CPL_DLL VSIFTruncateL (VSILFILE ∗, vsi_l_offset) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Truncate/expand the file to the specified size.
- int CPL_DLL VSIFFlushL (VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Flush pending writes to disk.
- int CPL_DLL int CPL_DLL VSIFPutcL (int, VSILFILE ∗) EXPERIMENTAL_CPL_WARN_UNUSED_RESULT

  Write a single byte to the file.
- VSIRangeStatus CPL_DLL VSIFGetRangeStatusL (VSILFILE ∗fp, vsi_l_offset nStart, vsi_l_offset nLength)

Return if a given file range contains data or holes filled with zeroes.

- int CPL_DLL VSIIngestFile (VSILFILE *fp, const char *pszFilename, GByte **ppabyRet, vsi_l_offset *pn↩ Size, GIntBig nMaxSize) CPL_WARN_UNUSED_RESULT

  Ingest a file into memory.
- int CPL_DLL VSIStatL (const char *, VSIStatBufL *) CPL_WARN_UNUSED_RESULT

  Get filesystem object info.
- int CPL_DLL VSIStatExL (const char *pszFilename, VSIStatBufL *psStatBuf, int nFlags) CPL_WARN_UNUSED_RESULT

  Get filesystem object info.
- int CPL_DLL VSIIsCaseSensitiveFS (const char *pszFilename)

  Returns if the filenames of the filesystem are case sensitive.
- int CPL_DLL VSISupportsSparseFiles (const char *pszPath)

  Returns if the filesystem supports sparse files.
- int CPL_DLL VSIHasOptimizedReadMultiRange (const char *pszPath)

  Returns if the filesystem supports efficient multi-range reading.
- const char CPL_DLL * VSIGetActualURL (const char *pszFilename)

  Returns the actual URL of a supplied filename.
- char CPL_DLL * VSIGetSignedURL (const char *pszFilename, CSLConstList papszOptions)

  Returns a signed URL of a supplied filename.
- const char CPL_DLL * VSIGetFileSystemOptions (const char *pszFilename)

  Return the list of options associated with a virtual file system handler as a serialized XML string.
- char CPL_DLL ** VSIGetFileSystemsPrefixes (void)

  Return the list of prefixes for virtual file system handlers currently registered.
- void CPL_DLL * VSIFGetNativeFileDescriptorL (VSILFILE *)

  Returns the "native" file descriptor for the virtual handle.
- void CPL_DLL * VSICalloc (size_t, size_t) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMalloc (size_t) CPL_WARN_UNUSED_RESULT
- void CPL_DLL VSIFree (void *)
- void CPL_DLL * VSIRealloc (void *, size_t) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * VSIStrdup (const char *) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMallocAligned (size_t nAlignment, size_t nSize) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMallocAlignedAuto (size_t nSize) CPL_WARN_UNUSED_RESULT
- void CPL_DLL VSIFreeAligned (void *ptr)
- void CPL_DLL * VSIMallocAlignedAutoVerbose (size_t nSize, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMalloc2 (size_t nSize1, size_t nSize2) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMalloc3 (size_t nSize1, size_t nSize2, size_t nSize3) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMallocVerbose (size_t nSize, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMalloc2Verbose (size_t nSize1, size_t nSize2, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIMalloc3Verbose (size_t nSize1, size_t nSize2, size_t nSize3, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSICallocVerbose (size_t nCount, size_t nSize, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * VSIReallocVerbose (void *pOldPtr, size_t nNewSize, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * VSIStrdupVerbose (const char *pszStr, const char *pszFile, int nLine) CPL_WARN_UNUSED_RESULT
- GIntBig CPL_DLL CPLGetPhysicalRAM (void)
- GIntBig CPL_DLL CPLGetUsablePhysicalRAM (void)
- char CPL_DLL ** VSIReadDir (const char *)

  Read names in a directory.
- char CPL_DLL ** VSIReadDirRecursive (const char *pszPath)

  Read names in a directory recursively.
- char CPL_DLL ** VSIReadDirEx (const char *pszPath, int nMaxFiles)

  Read names in a directory.

- int CPL_DLL VSIMkdir (const char *pszPathname, long mode)

  Create a directory.
- int CPL_DLL VSIMkdirRecursive (const char *pszPathname, long mode)

  Create a directory and all its ancestors.
- int CPL_DLL VSIRmdir (const char *pszDirname)

  Delete a directory.
- int CPL_DLL VSIRmdirRecursive (const char *pszDirname)

  Delete a directory recursively.
- int CPL_DLL VSIUnlink (const char *pszFilename)

  Delete a file.
- int CPL_DLL VSIRename (const char *oldpath, const char *newpath)

  Rename a file.
- char CPL_DLL * VSIStrerror (int)
- GIntBig CPL_DLL VSIGetDiskFreeSpace (const char *pszDirname)

  Return free disk space available on the filesystem.
- void CPL_DLL VSIInstallMemFileHandler (void)

  Install "memory" file system handler.
- void CPL_DLL VSIInstallSubFileHandler (void)
- void VSIInstallGZipFileHandler (void)

  Install GZip file system handler.
- void VSIInstallZipFileHandler (void)

  Install ZIP file system handler.
- void VSIInstallStdinHandler (void)

  Install /vsistdin/ file system handler.
- void VSIInstallStdoutHandler (void)

  Install /vsistdout/ file system handler.
- void CPL_DLL VSIInstallSparseFileHandler (void)
- void VSIInstallTarFileHandler (void)

  Install /vsitar/ file system handler.
- VSILFILE CPL_DLL * VSIFileFromMemBuffer (const char *pszFilename, GByte *pabyData, vsi_l_offset n↩DataLength, int bTakeOwnership) CPL_WARN_UNUSED_RESULT

  Create memory "file" from a buffer.
- GByte CPL_DLL * VSIGetMemFileBuffer (const char *pszFilename, vsi_l_offset *pnDataLength, int b↩UnlinkAndSeize)

  Fetch buffer underlying memory file.
- void CPL_DLL VSIStdoutSetRedirection (VSIWriteFunction pFct, FILE *stream)

11.13.1 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.

It 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.

11.13.2 Macro Definition Documentation

11.13.2.1 CPLReadDir

```
#define CPLReadDir VSIReadDir
```

Alias of VSIReadDir()

11.13.2.2 VSI_CALLOC_VERBOSE

```
#define VSI_CALLOC_VERBOSE(
            nCount,
            nSize ) VSICallocVerbose(nCount,nSize,__FILE__,__LINE__)
```

VSI_CALLOC_VERBOSE

11.13.2.3 VSI_ISDIR

```
#define VSI_ISDIR(
            x ) S_ISDIR(x)
```

Test if the file is a directory

11.13.2.4 VSI_ISLNK

```
#define VSI_ISLNK(
            x ) S_ISLNK(x)
```

Test if the file is a symbolic link

11.13.2.5 VSI_ISREG

```
#define VSI_ISREG(
            x ) S_ISREG(x)
```

Test if the file is a regular file

11.13.2.6 VSI_L_OFFSET_MAX

```
#define VSI_L_OFFSET_MAX GUINTBIG_MAX
```

Maximum value for a file offset

11.13.2.7 VSI_MALLOC2_VERBOSE

```
#define VSI_MALLOC2_VERBOSE(
            nSize1,
            nSize2 ) VSIMalloc2Verbose(nSize1,nSize2,__FILE__,__LINE__)
```

VSI_MALLOC2_VERBOSE

11.13.2.8 VSI_MALLOC3_VERBOSE

```
#define VSI_MALLOC3_VERBOSE(
            nSize1,
            nSize2,
            nSize3 ) VSIMalloc3Verbose(nSize1,nSize2,nSize3,__FILE__,__LINE__)
```

VSI_MALLOC3_VERBOSE

11.13.2.9 VSI_MALLOC_ALIGNED_AUTO_VERBOSE

```
#define VSI_MALLOC_ALIGNED_AUTO_VERBOSE(
            size ) VSIMallocAlignedAutoVerbose(size,__FILE__,__LINE__)
```

VSIMallocAlignedAutoVerbose() with FILE and LINE reporting

11.13.2.10 VSI_MALLOC_VERBOSE

```
#define VSI_MALLOC_VERBOSE(
            size ) VSIMallocVerbose(size,__FILE__,__LINE__)
```

VSI_MALLOC_VERBOSE

11.13.2.11 VSI_REALLOC_VERBOSE

```
#define VSI_REALLOC_VERBOSE(
            pOldPtr,
            nNewSize ) VSIReallocVerbose(pOldPtr,nNewSize,__FILE__,__LINE__)
```

VSI_REALLOC_VERBOSE

11.13.2.12 VSI_STAT_EXISTS_FLAG

```
#define VSI_STAT_EXISTS_FLAG 0x1
```

Flag provided to VSIStatExL() to test if the file exists

11.13.2.13 VSI_STAT_NATURE_FLAG

```
#define VSI_STAT_NATURE_FLAG 0x2
```

Flag provided to VSIStatExL() to query the nature (file/dir) of the file

11.13.2.14 VSI_STAT_SET_ERROR_FLAG

```
#define VSI_STAT_SET_ERROR_FLAG 0x8
```

Flag provided to VSIStatExL() to issue a VSIError in case of failure

11.13.2.15 VSI_STAT_SIZE_FLAG

```
#define VSI_STAT_SIZE_FLAG 0x4
```

Flag provided to VSIStatExL() to query the file size

11.13.2.16 VSI_STRDUP_VERBOSE

```
#define VSI_STRDUP_VERBOSE(
            pszStr ) VSIStrdupVerbose(pszStr,__FILE__,__LINE__)
```

VSI_STRDUP_VERBOSE

11.13.3 Typedef Documentation

11.13.3.1 vsi_l_offset

```
typedef GUIntBig vsi_l_offset
```

Type for a file offset

11.13.3.2 VSILFILE

```
typedef FILE VSILFILE
```

Opaque type for a FILE that implements the VSIVirtualHandle API

11.13.3.3 VSIStatBufL

```
typedef struct VSI_STAT64_T VSIStatBufL
```

Type for VSIStatL()

11.13.3.4 VSIWriteFunction

```
typedef size_t(* VSIWriteFunction) (const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

Callback used by VSIStdoutSetRedirection()

11.13.4 Enumeration Type Documentation

11.13.4.1 VSIRangeStatus

```
enum VSIRangeStatus
```

Range status

Enumerator

VSI_RANGE_STATUS_UNKNOWN	Unknown
VSI_RANGE_STATUS_DATA	Data present
VSI_RANGE_STATUS_HOLE	Hole

11.13.5 Function Documentation

11.13.5.1 CPLGetPhysicalRAM()

```
GIntBig CPL_DLL CPLGetPhysicalRAM (
            void  )
```

Return the total physical RAM in bytes.

Returns

the total physical RAM in bytes (or 0 in case of failure).

Since

GDAL 2.0

11.13.5.2 CPLGetUsablePhysicalRAM()

```
GIntBig CPL_DLL CPLGetUsablePhysicalRAM (
            void  )
```

Return the total physical RAM, usable by a process, in bytes.

This is the same as CPLGetPhysicalRAM() except it will limit to 2 GB for 32 bit processes.

Note: This memory may already be partly used by other processes.

Returns

the total physical RAM, usable by a process, in bytes (or 0 in case of failure).

Since

GDAL 2.0

11.13.5.3 VSICalloc()

```
void CPL_DLL* VSICalloc (
            size_t nCount,
            size_t nSize )
```

Analog of calloc(). Use VSIFree() to free

11.13.5.4 VSICallocVerbose()

```
void CPL_DLL* VSICallocVerbose (
            size_t nCount,
            size_t nSize,
            const char * pszFile,
            int nLine )
```

VSICallocVerbose

11.13.5.5 VSIFCloseL()

```
int CPL_DLL VSIFCloseL (
            VSILFILE * fp )
```

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

Parameters

fp	file handle opened with VSIFOpenL(). Passing a nullptr produces undefined behavior.

Returns

0 on success or -1 on failure.

11.13.5.6 VSIFEofL()

```
int CPL_DLL VSIFEofL (
            VSILFILE * fp )
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful VSIFSeekL() call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters

fp	file handle opened with VSIFOpenL().

Returns

TRUE if at EOF else FALSE.

11.13.5.7 VSIFFlushL()

```
int CPL_DLL VSIFFlushL (
            VSILFILE * fp )
```

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters

fp	file handle opened with VSIFOpenL().

Returns

0 on success or -1 on error.

11.13.5.8 VSIFGetNativeFileDescriptorL()

```
void CPL_DLL* VSIFGetNativeFileDescriptorL (
            VSILFILE * fp )
```

Returns the "native" file descriptor for the virtual handle.

This will only return a non-NULL value for "real" files handled by the operating system (to be opposed to GDAL virtual file systems).

On POSIX systems, this will be a integer value ("fd") cast as a void∗. On Windows systems, this will be the HANDLE.

Parameters

fp	file handle opened with VSIFOpenL().

Returns

the native file descriptor, or NULL.

11.13.5.9 VSIFGetRangeStatusL()

```
VSIRangeStatus CPL_DLL VSIFGetRangeStatusL (
            VSILFILE * fp,
            vsi_l_offset nOffset,
            vsi_l_offset nLength )
```

Return if a given file range contains data or holes filled with zeroes.

This uses the filesystem capabilities of querying which regions of a sparse file are allocated or not. This is currently only implemented for Linux (and no other Unix derivatives) and Windows.

Note: A return of VSI_RANGE_STATUS_DATA doesn't exclude that the extent is filled with zeroes! It must be interpreted as "may contain non-zero data".

Parameters

fp	file handle opened with VSIFOpenL().
nOffset	offset of the start of the extent.
nLength	extent length.

Returns

extent status: VSI_RANGE_STATUS_UNKNOWN, VSI_RANGE_STATUS_DATA or VSI_RANGE_STATU↩S_HOLE

Since

GDAL 2.2

11.13.5.10 VSIFileFromMemBuffer()

```
VSILFILE CPL_DLL* VSIFileFromMemBuffer (
            const char * pszFilename,
            GByte * pabyData,
            vsi_l_offset nDataLength,
            int bTakeOwnership )
```

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If bTakeOwnership is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the pabyData contents.

Parameters

pszFilename	the filename to be created.
pabyData	the data buffer for the file.
nDataLength	the length of buffer in bytes.
bTakeOwnership	TRUE to transfer "ownership" of buffer or FALSE.

Returns

open file handle on created file (see VSIFOpenL()).

11.13.5.11 VSIFOpenExL()

```
VSILFILE CPL_DLL* VSIFOpenExL (
            const char * pszFilename,
            const char * pszAccess,
            int bSetError )
```

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the pszAccess string.

Note that the "VSILFILE ∗" returned by this function is *NOT* a standard C library FILE ∗, and cannot be used with any functions other than the "VSI∗L" family of functions. They aren't "real" FILE objects.

On windows it is possible to define the configuration option GDAL_FILE_IS_UTF8 to have pszFilename treated as being in the local encoding instead of UTF-8, restoring the pre-1.8.0 behavior of VSIFOpenL().

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

pszFilename	the file to open. UTF-8 encoded.
pszAccess	access requested (i.e. "r", "r+", "w")
bSetError	flag determining whether or not this open call should set VSIErrors on failure.

Returns

NULL on failure, or the file handle.

Since

GDAL 2.1

11.13.5.12 VSIFOpenL()

```
VSILFILE CPL_DLL* VSIFOpenL (
            const char * pszFilename,
            const char * pszAccess )
```

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the pszAccess string.

Note that the "VSILFILE ∗" returned since GDAL 1.8.0 by this function is *NOT* a standard C library FILE ∗, and cannot be used with any functions other than the "VSI∗L" family of functions. They aren't "real" FILE objects.

On windows it is possible to define the configuration option GDAL_FILE_IS_UTF8 to have pszFilename treated as being in the local encoding instead of UTF-8, restoring the pre-1.8.0 behavior of VSIFOpenL().

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

pszFilename	the file to open. UTF-8 encoded.
pszAccess	access requested (i.e. "r", "r+", "w")

Returns

NULL on failure, or the file handle.

11.13.5.13 VSIFPutcL()

```
int CPL_DLL int CPL_DLL VSIFPutcL (
            int nChar,
            VSILFILE * fp )
```

Write a single byte to the file.

Writes the character nChar, cast to an unsigned char, to file.

Almost an analog of the POSIX fputc() call, except that it returns the number of character written (1 or 0), and not the (cast) character itself or EOF.

Parameters

nChar	character to write.
fp	file handle opened with VSIFOpenL().

Returns

1 in case of success, 0 on error.

11.13.5.14 VSIFReadL()

```
size_t CPL_DLL VSIFReadL (
            void * pBuffer,
            size_t nSize,
            size_t nCount,
            VSILFILE * fp )
```

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with VSIFOpenL().

Returns

number of objects successfully read.

11.13.5.15 VSIFReadMultiRangeL()

```
int CPL_DLL VSIFReadMultiRangeL (
            int nRanges,
            void ** ppData,
            const vsi_l_offset * panOffsets,
            const size_t * panSizes,
            VSILFILE * fp )
```

Read several ranges of bytes from file.

Reads nRanges objects of panSizes[i] bytes from the indicated file at the offset panOffsets[i] into the buffer ppData[i].

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

Parameters

nRanges	number of ranges to read.
ppData	array of nRanges buffer into which the data should be read (ppData[i] must be at list panSizes[i] bytes).
panOffsets	array of nRanges offsets at which the data should be read.
panSizes	array of nRanges sizes of objects to read (in bytes).
fp	file handle opened with VSIFOpenL().

Returns

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

11.13.5.16 VSIFree()

```
void CPL_DLL VSIFree (
            void * pData )
```

Analog of free() for data allocated with VSIMalloc(), VSICalloc(), VSIRealloc()

11.13.5.17 VSIFreeAligned()

```
void CPL_DLL VSIFreeAligned (
            void * ptr )
```

Free a buffer allocated with VSIMallocAligned().

Parameters

ptr	Buffer to free.

Since

GDAL 2.2

11.13.5.18 VSIFSeekL()

```
int CPL_DLL VSIFSeekL (
            VSILFILE * fp,
            vsi_l_offset nOffset,
            int nWhence )
```

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Caution: vsi_l_offset is a unsigned type, so SEEK_CUR can only be used for positive seek. If negative seek is needed, use VSIFSeekL(fp, VSIFTellL(fp) + negative_offset, SEEK_SET).

Parameters

fp	file handle opened with VSIFOpenL().
nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

11.13.5.19 VSIFTellL()

```
vsi_l_offset CPL_DLL VSIFTellL (
            VSILFILE * fp )
```

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

Parameters

fp	file handle opened with VSIFOpenL().

Returns

file offset in bytes.

11.13.5.20 VSIFTruncateL()

```
int CPL_DLL VSIFTruncateL (
            VSILFILE * fp,
            vsi_l_offset nNewSize )
```

Truncate/expand the file to the specified size.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

Parameters

fp	file handle opened with VSIFOpenL().
nNewSize	new size in bytes.

Returns

0 on success

Since

GDAL 1.9.0

11.13.5.21 VSIFWriteL()

```
size_t CPL_DLL VSIFWriteL (
            const void * pBuffer,
            size_t nSize,
            size_t nCount,
            VSILFILE * fp )
```

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters

pBuffer	the buffer from which the data should be written (at least nCount ∗ nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with VSIFOpenL().

Returns

number of objects successfully written.

11.13.5.22 VSIGetActualURL()

```
const char CPL_DLL* VSIGetActualURL (
            const char * pszFilename )
```

Returns the actual URL of a supplied filename.

Currently only returns a non-NULL value for network-based virtual file systems. For example "/vsis3/bucket/filename" will be expanded as "https://bucket.s3.amazon.com/filename"

Note that the lifetime of the returned string, is short, and may be invalidated by any following GDAL functions.

Parameters

pszFilename	the path of the filesystem object. UTF-8 encoded.

Returns

the actual URL corresponding to the supplied filename, or NULL. Should not be freed.

Since

GDAL 2.3

11.13.5.23 VSIGetDiskFreeSpace()

```
GIntBig CPL_DLL VSIGetDiskFreeSpace (
            const char * pszDirname )
```

Return free disk space available on the filesystem.

This function returns the free disk space available on the filesystem.

Parameters

pszDirname	a directory of the filesystem to query.

Returns

The free space in bytes. Or -1 in case of error.

Since

GDAL 2.1

11.13.5.24 VSIGetFileSystemOptions()

```
const char CPL_DLL* VSIGetFileSystemOptions (
            const char * pszFilename )
```

Return the list of options associated with a virtual file system handler as a serialized XML string.

Those options may be set as configuration options with CPLSetConfigOption().

Parameters

pszFilename	a filename, or prefix of a virtual file system handler.

Returns

a string, which must not be freed, or NULL if no options is declared.

Since

GDAL 2.3

11.13.5.25 VSIGetFileSystemsPrefixes()

```
char CPL_DLL** VSIGetFileSystemsPrefixes (
            void  )
```

Return the list of prefixes for virtual file system handlers currently registered.

Typically: "", "/vsimem/", "/vsicurl/", etc

Returns

a NULL terminated list of prefixes. Must be freed with CSLDestroy()

Since

GDAL 2.3

11.13.5.26 VSIGetMemFileBuffer()

```
GByte CPL_DLL* 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

pszFilename	the name of the file to grab the buffer of.
pnDataLength	(file) length returned in this variable.
bUnlinkAndSeize	TRUE to remove the file, or FALSE to leave unaltered.

Returns

pointer to memory buffer or NULL on failure.

11.13.5.27 VSIGetSignedURL()

```
char CPL_DLL* VSIGetSignedURL (
            const char * pszFilename,
            CSLConstList papszOptions )
```

Returns a signed URL of a supplied filename.

Currently only returns a non-NULL value for /vsis3/, /vsigs/, /vsiaz/ and /vsioss/ For example "/vsis3/bucket/filename" will be expanded as "https://bucket.s3.amazon.com/filename?X-Amz-Algorithm=AWS4-HMAC-SHA256..." Configuration options that apply for file opening (typically to provide credentials), and are returned by VSIGetFileSystemOptions(), are also valid in that context.

Parameters

pszFilename	the path of the filesystem object. UTF-8 encoded.
papszOptions	list of options, or NULL. Depend on file system handler. For /vsis3/, /vsigs/, /vsiaz/ and /vsioss/, the following options are supported: • START_DATE=YYMMDDTHHMMSSZ: date and time in UTC following ISO 8601 standard, corresponding to the start of validity of the URL. If not specified, current date time. • EXPIRATION_DELAY=number_of_seconds: number between 1 and 604800 (seven days) for the validity of the signed URL. Defaults to 3600 (one hour) • VERB=GET/HEAD/DELETE/PUT/POST: HTTP VERB for which the request will be used. Default to GET.

/vsiaz/ supports additional options:

- SIGNEDIDENTIFIER=value: to relate the given shared access signature to a corresponding stored access policy.
- SIGNEDPERMISSIONS=r|w: permissions associated with the shared access signature. Normally deduced from VERB.

Returns

a signed URL, or NULL. Should be freed with CPLFree().

Since

GDAL 2.3

11.13.5.28 VSIHasOptimizedReadMultiRange()

```
int CPL_DLL VSIHasOptimizedReadMultiRange (
            const char * pszPath )
```

Returns if the filesystem supports efficient multi-range reading.

Currently only returns TRUE for /vsicurl/ and derived file systems.

Parameters

pszPath	the path of the filesystem object to be tested. UTF-8 encoded.

Returns

TRUE if the file system is known to have an efficient multi-range reading.

Since

GDAL 2.3

11.13.5.29 VSIIngestFile()

```
int CPL_DLL VSIIngestFile (
            VSILFILE * fp,
            const char * pszFilename,
            GByte ** ppabyRet,
            vsi_l_offset * pnSize,
            GIntBig nMaxSize )
```

Ingest a file into memory.

Read the whole content of a file into a memory buffer.

Either fp or pszFilename can be NULL, but not both at the same time.

If fp is passed non-NULL, it is the responsibility of the caller to close it.

If non-NULL, the returned buffer is guaranteed to be NUL-terminated.

Parameters

fp	file handle opened with VSIFOpenL().
pszFilename	filename.
ppabyRet	pointer to the target buffer. *ppabyRet must be freed with VSIFree()
pnSize	pointer to variable to store the file size. May be NULL.
nMaxSize	maximum size of file allowed. If no limit, set to a negative value.

Returns

TRUE in case of success.

Since

GDAL 1.11

11.13.5.30 VSIInstallGZipFileHandler()

```
void VSIInstallGZipFileHandler (
            void  )
```

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly and writing in GZip (.gz) files.

All portions of the file system underneath the base path "/vsigzip/" will be handled by this driver.

Additional documentation is to be found at: http://trac.osgeo.org/gdal/wiki/UserDocs/Read↩InZip

Since

GDAL 1.6.0

11.13.5.31 VSIInstallMemFileHandler()

```
void CPL_DLL VSIInstallMemFileHandler (
            void  )
```

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI∗L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

Directory related functions are supported.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
                            vsi_l_offset *pnOutDataLength )
{
    // create memory file system object from buffer.
    VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                      nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                             NULL, NULL );

    // close source file, and "unlink" it.
    GDALClose( hDS );
    VSIUnlink( "/vsimem/work.dat" );

    // seize the buffer associated with the output file.

    return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}
```

11.13.5.32 VSIInstallSparseFileHandler()

```
void CPL_DLL VSIInstallSparseFileHandler (
            void  )
```

Install /vsisparse/ virtual file handler.

See also

/vsisparse/ documentation

11.13.5.33 VSIInstallStdinHandler()

```
void VSIInstallStdinHandler (
            void  )
```

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input stream.

The file operations available are of course limited to Read() and forward Seek() (full seek in the first MB of a file).

Since

GDAL 1.8.0

11.13.5.34 VSIInstallStdoutHandler()

```
void VSIInstallStdoutHandler (
            void  )
```

Install /vsistdout/ file system handler.

A special file handler is installed that allows writing to the standard output stream.

The file operations available are of course limited to Write().

A variation of this file system exists as the /vsistdout_redirect/ file system handler, where the output function can be defined with VSIStdoutSetRedirection().

Since

GDAL 1.8.0

11.13.5.35 VSIInstallSubFileHandler()

```
void CPL_DLL VSIInstallSubFileHandler (
            void  )
```

Install /vsisubfile/ virtual file handler.

See also

/vsisubfile/ documentation

11.13.5.36 VSIInstallTarFileHandler()

```
void VSIInstallTarFileHandler (
            void  )
```

Install /vsitar/ file system handler.

A special file handler is installed that allows reading on-the-fly in TAR (regular .tar, or compressed .tar.gz/.tgz) archives.

All portions of the file system underneath the base path "/vsitar/" will be handled by this driver.

The syntax to open a file inside a tar file is /vsitar/path/to/the/file.tar/path/inside/the/tar/file were path/to/the/file.tar is relative or absolute and path/inside/the/tar/file is the relative path to the file inside the archive.

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .tar extension : /vsitar/{/path/to/the/archive}/path/inside/the/tar/file. Note that /path/to/the/archive may also itself this alternate syntax.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsitar//home/gdal/... For example gdalinfo /vsitar/myarchive.tar/subdir1/file1.tif

Syntactic sugar : if the tar archive contains only one file located at its root, just mentionning "/vsitar/path/to/the/file.↩tar" will work

VSIStatL() will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through VSIReadDir().

Since

GDAL 1.8.0

11.13.5.37 VSIInstallZipFileHandler()

```
void VSIInstallZipFileHandler (
            void  )
```

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives.

All portions of the file system underneath the base path "/vsizip/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsizip/path/to/the/file.zip/path/inside/the/zip/file where path/to/the/file.zip is relative or absolute and path/inside/the/zip/file is the relative path to the file inside the archive.

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .zip extension : /vsizip/{/path/to/the/archive}/path/inside/the/zip/file. Note that /path/to/the/archive may also itself use this alternate syntax.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsizip//home/gdal/... For example gdalinfo /vsizip/myarchive.zip/subdir1/file1.tif

Syntactic sugar : if the .zip file contains only one file located at its root, just mentioning "/vsizip/path/to/the/file.zip" will work

VSIStatL() will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through VSIReadDir().

Since GDAL 1.8.0, write capabilities are available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file. Read and write operations cannot be interleaved : the new zip must be closed before being re-opened for read.

Additional documentation is to be found at http://trac.osgeo.org/gdal/wiki/UserDocs/Read↩InZip

Since

GDAL 1.6.0

11.13.5.38 VSIIsCaseSensitiveFS()

```
int CPL_DLL VSIIsCaseSensitiveFS (
            const char * pszFilename )
```

Returns if the filenames of the filesystem are case sensitive.

This method retrieves to which filesystem belongs the passed filename and return TRUE if the filenames of that filesystem are case sensitive.

Currently, this will return FALSE only for Windows real filenames. Other VSI virtual filesystems are case sensitive.

This methods avoid ugly #ifndef WIN32 / #endif code, that is wrong when dealing with virtual filenames.

Parameters

pszFilename	the path of the filesystem object to be tested. UTF-8 encoded.

Returns

TRUE if the filenames of the filesystem are case sensitive.

Since

GDAL 1.8.0

11.13.5.39 VSIMalloc()

```
void CPL_DLL* VSIMalloc (
            size_t nSize )
```

Analog of malloc(). Use VSIFree() to free

11.13.5.40 VSIMalloc2()

```
void CPL_DLL* 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(). 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.

11.13.5.41 VSIMalloc2Verbose()

```
void CPL_DLL* VSIMalloc2Verbose (
            size_t nSize1,
            size_t nSize2,
            const char * pszFile,
            int nLine )
```

VSIMalloc2Verbose

11.13.5.42 VSIMalloc3()

```
void CPL_DLL* VSIMalloc3 (
            size_t nSize1,
            size_t nSize2,
            size_t nSize3 )
```

VSIMalloc3 allocates (nSize1 ∗ nSize2 ∗ nSize3) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with CPLError(). 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.

11.13.5.43 VSIMalloc3Verbose()

```
void CPL_DLL* VSIMalloc3Verbose (
            size_t nSize1,
            size_t nSize2,
            size_t nSize3,
            const char * pszFile,
            int nLine )
```

VSIMalloc3Verbose

11.13.5.44 VSIMallocAligned()

```
void CPL_DLL* VSIMallocAligned (
            size_t nAlignment,
            size_t nSize )
```

Allocates a buffer with an alignment constraint.

The return value must be freed with VSIFreeAligned().

Parameters

nAlignment	Must be a power of 2, multiple of sizeof(void∗), and lesser than 256.
nSize	Size of the buffer to allocate.

Returns

a buffer aligned on nAlignment and of size nSize, or NULL

Since

GDAL 2.2

11.13.5.45 VSIMallocAlignedAuto()

```
void CPL_DLL* VSIMallocAlignedAuto (
            size_t nSize )
```

Allocates a buffer with an alignment constraint such that it can be used by the most demanding vector instruction set on that platform.

The return value must be freed with VSIFreeAligned().

Parameters

nSize	Size of the buffer to allocate.

Returns

an aligned buffer of size nSize, or NULL

Since

GDAL 2.2

11.13.5.46 VSIMallocAlignedAutoVerbose()

```
void CPL_DLL* VSIMallocAlignedAutoVerbose (
            size_t nSize,
            const char * pszFile,
            int nLine )
```

See VSIMallocAlignedAuto()

11.13.5.47 VSIMallocVerbose()

```
void CPL_DLL* VSIMallocVerbose (
            size_t nSize,
            const char * pszFile,
            int nLine )
```

VSIMallocVerbose

11.13.5.48 VSIMkdir()

```
int CPL_DLL VSIMkdir (
            const char * pszPathname,
            long mode )
```

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters

pszPathname	the path to the directory to create. UTF-8 encoded.
mode	the permissions mode.

Returns

0 on success or -1 on an error.

11.13.5.49 VSIMkdirRecursive()

```
int CPL_DLL VSIMkdirRecursive (
            const char * pszPathname,
            long mode )
```

Create a directory and all its ancestors.

Parameters

pszPathname	the path to the directory to create. UTF-8 encoded.
mode	the permissions mode.

Returns

0 on success or -1 on an error.

Since

GDAL 2.3

11.13.5.50 VSIReadDir()

```
char CPL_DLL** VSIReadDir (
            const char * pszPath )
```

Read names in a directory.

This function abstracts access to directory contains. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with CSLDestroy() when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

Parameters

pszPath	the relative, or absolute path of a directory to read. UTF-8 encoded.

Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

11.13.5.51 VSIReadDirEx()

```
char CPL_DLL** VSIReadDirEx (
            const char * pszPath,
            int nMaxFiles )
```

Read names in a directory.

This function abstracts access to directory contains. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with CSLDestroy() when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

If nMaxFiles is set to a positive number, directory listing will stop after that limit has been reached. Note that to indicate truncate, at least one element more than the nMaxFiles limit will be returned. If CSLCount() on the result is lesser or equal to nMaxFiles, then no truncation occurred.

Parameters

pszPath	the relative, or absolute path of a directory to read. UTF-8 encoded.
nMaxFiles	maximum number of files after which to stop, or 0 for no limit.

Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

Since

GDAL 2.1

11.13.5.52 VSIReadDirRecursive()

```
char CPL_DLL** VSIReadDirRecursive (
            const char * pszPathIn )
```

Read names in a directory recursively.

This function abstracts access to directory contents and subdirectories. It returns a list of strings containing the names of files and directories in this directory and all subdirectories. The resulting string list becomes the responsibility of the application and should be freed with CSLDestroy() when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

Parameters

psz↩ PathIn	the relative, or absolute path of a directory to read. UTF-8 encoded.

Returns

The list of entries in the directory and subdirectories or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

Since

GDAL 1.10.0

11.13.5.53 VSIRealloc()

```
void CPL_DLL* VSIRealloc (
            void * pData,
            size_t nNewSize )
```

Analog of realloc(). Use VSIFree() to free

11.13.5.54 VSIReallocVerbose()

```
void CPL_DLL* VSIReallocVerbose (
            void * pOldPtr,
            size_t nNewSize,
            const char * pszFile,
            int nLine )
```

VSIReallocVerbose

11.13.5.55 VSIRename()

```
int CPL_DLL VSIRename (
            const char * oldpath,
            const char * newpath )
```

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

Parameters

oldpath	the name of the file to be renamed. UTF-8 encoded.
newpath	the name the file should be given. UTF-8 encoded.

Returns

0 on success or -1 on an error.

11.13.5.56 VSIRewindL()

```
void CPL_DLL VSIRewindL (
            VSILFILE * fp )
```

Rewind the file pointer to the beginning of the file.

This is equivalent to VSIFSeekL(fp, 0, SEEK_SET)

Analog of the POSIX rewind() call.

Parameters

fp	file handle opened with VSIFOpenL().

11.13.5.57 VSIRmdir()

```
int CPL_DLL VSIRmdir (
            const char * pszDirname )
```

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

Parameters

pszDirname	the path of the directory to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

11.13.5.58 VSIRmdirRecursive()

```
int CPL_DLL VSIRmdirRecursive (
            const char * pszDirname )
```

Delete a directory recursively.

Deletes a directory object and its content from the file system.

Returns

0 on success or -1 on an error.

Since

GDAL 2.3

11.13.5.59 VSIStatExL()

```
int CPL_DLL VSIStatExL (
            const char * pszFilename,
            VSIStatBufL * psStatBuf,
            int nFlags )
```

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability, only use the st_size (size in bytes) and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function, with an extra parameter to specify which information is needed, which offers a potential for speed optimizations on specialized and potentially slow virtual filesystem objects (/vsigzip/, /vsicurl/)

Parameters

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.
nFlags	0 to get all information, or VSI_STAT_EXISTS_FLAG, VSI_STAT_NATURE_FLAG or VSI_STAT_SIZE_FLAG, or a combination of those to get partial info.

Returns

0 on success or -1 on an error.

Since

GDAL 1.8.0

11.13.5.60 VSIStatL()

```
int CPL_DLL VSIStatL (
            const char * pszFilename,
            VSIStatBufL * psStatBuf )
```

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability, only use the st_size (size in bytes) and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

Parameters

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.

Returns

0 on success or -1 on an error.

11.13.5.61 VSIStdoutSetRedirection()

```
void CPL_DLL VSIStdoutSetRedirection (
            VSIWriteFunction pFct,
            FILE * stream )
```

Set an alternative write function and output file handle instead of fwrite() / stdout.

Parameters

pFct	Function with same signature as fwrite()
stream	File handle on which to output. Passed to pFct.

Since

GDAL 2.0

11.13.5.62 VSIStrdup()

```
char CPL_DLL* VSIStrdup (
            const char * pszString )
```

Analog of strdup(). Use VSIFree() to free

11.13.5.63 VSIStrdupVerbose()

```
char CPL_DLL* VSIStrdupVerbose (
            const char * pszStr,
            const char * pszFile,
            int nLine )
```

VSIStrdupVerbose

11.13.5.64 VSIStrerror()

```
char CPL_DLL* VSIStrerror (
            int nErrno )
```

Return the error string corresponding to the error number. Do not free it

11.13.5.65 VSISupportsSparseFiles()

```
int CPL_DLL VSISupportsSparseFiles (
            const char * pszPath )
```

Returns if the filesystem supports sparse files.

Only supported on Linux (and no other Unix derivatives) and Windows. On Linux, the answer depends on a few hardcoded signatures for common filesystems. Other filesystems will be considered as not supporting sparse files.

Parameters

pszPath	the path of the filesystem object to be tested. UTF-8 encoded.

Returns

TRUE if the file system is known to support sparse files. FALSE may be returned both in cases where it is known to not support them, or when it is unknown.

Since

GDAL 2.2

11.13.5.66 VSIUnlink()

```
int CPL_DLL VSIUnlink (
            const char * pszFilename )
```

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters

pszFilename	the path of the file to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

11.14 cpl_worker_thread_pool.h File Reference

```
#include "cpl_multiproc.h"
#include "cpl_list.h"
#include <vector>
```

Classes

- struct CPLWorkerThreadJob
- struct CPLWorkerThread
- class CPLWorkerThreadPool

11.14.1 Detailed Description

Class to manage a pool of worker threads.

Since

GDAL 2.1

11.15 sdtsdataset.cpp File Reference

```
#include "gdal_frmts.h"
#include "gdal_pam.h"
#include "ogr_spatialref.h"
#include "sdts_al.h"
```

Classes

- class SDTSDataset
- class SDTSRasterBand

11.15.1 Detailed Description

exclude

Index

_CPLAssert
    cpl_error.h, 217
_CPLHashSet, 43
_CPLList, 43
    pData, 43
    psNext, 44
_CPLLock, 44
_CPLQuadTree, 44
_CPLSpawnedProcess, 44
_CPLSpinLock, 44
_CPL_SHA256Context, 43
_MutexLinkedElt, 45
_QuadTreeNode, 45
~CPLODBCSession
    CPLODBCSession, 77
~CPLODBCStatement
    CPLODBCStatement, 80
~CPLWorkerThreadPool
    CPLWorkerThreadPool, 95
~DDFModule
    DDFModule, 106

ABS
    cpl_port.h, 258
Add
    CPLJSONArray, 48–50
    CPLJSONObject, 59–61
AddField
    DDFModule, 106
    DDFRecord, 110
AddNull
    CPLJSONObject, 62
Append
    CPLODBCStatement, 80, 81
AppendEscaped
    CPLODBCStatement, 81
Appendf
    CPLODBCStatement, 81
AssembleRings
    SDTSPolygonReader, 138
    SDTSRawPolygon, 147
AttachToPolygons
    SDTSLineReader, 135

bLarge
    CPLSharedFileInfo, 93
BeginTransaction
    CPLODBCSession, 77

CPL_ENC_ASCII
    cpl_string.h, 277
CPL_ENC_ISO8859_1
    cpl_string.h, 277
CPL_ENC_LOCALE
    cpl_string.h, 277
CPL_ENC_UCS2
    cpl_string.h, 277
CPL_ENC_UCS4
    cpl_string.h, 278
CPL_ENC_UTF16
    cpl_string.h, 278
CPL_ENC_UTF8
    cpl_string.h, 278
CPL_FALLTHROUGH
    cpl_port.h, 258
CPL_FORMAT_STRING
    cpl_port.h, 258
CPL_FRMT_GB_WITHOUT_PREFIX
    cpl_port.h, 258
CPL_FRMT_GIB
    cpl_port.h, 258
CPL_FRMT_GUIB
    cpl_port.h, 258
CPL_LSBINT16PTR
    cpl_port.h, 259
CPL_LSBINT32PTR
    cpl_port.h, 259
CPL_LSBPTR16
    cpl_port.h, 259
CPL_LSBPTR32
    cpl_port.h, 259
CPL_LSBPTR64
    cpl_port.h, 259
CPL_LSBSINT16PTR
    cpl_port.h, 260
CPL_LSBSINT32PTR
    cpl_port.h, 260
CPL_LSBUINT16PTR
    cpl_port.h, 260
CPL_LSBUINT32PTR
    cpl_port.h, 260
CPL_LSBWORD16
    cpl_port.h, 260
CPL_LSBWORD32
    cpl_port.h, 260
CPL_MSBPTR16
    cpl_port.h, 260
CPL_MSBPTR32
    cpl_port.h, 261

CPL_MSBPTR64
cpl_port.h, 261
CPL_MSBWORD16
cpl_port.h, 261
CPL_MSBWORD32
cpl_port.h, 261
CPL_NO_RETURN
cpl_port.h, 261
CPL_NULL_TERMINATED
cpl_port.h, 261
CPL_PRINT_FUNC_FORMAT
cpl_port.h, 261
CPL_RESTRICT
cpl_port.h, 262
CPL_RETURNS_NONNULL
cpl_port.h, 262
CPL_SCAN_FUNC_FORMAT
cpl_port.h, 262
CPL_SCANF_FORMAT_STRING
cpl_port.h, 262
CPL_SHA1Context, 45
CPL_SWAP16
cpl_port.h, 262
CPL_SWAP16PTR
cpl_port.h, 262
CPL_SWAP32
cpl_port.h, 263
CPL_SWAP32PTR
cpl_port.h, 263
CPL_SWAP64
cpl_port.h, 263
CPL_SWAP64PTR
cpl_port.h, 264
CPL_SWAPDOUBLE
cpl_port.h, 264
CPL_UNUSED
cpl_port.h, 264
CPL_WARN_UNUSED_RESULT
cpl_port.h, 264
CPLAddXMLAttributeAndValue
cpl_minixml.h, 247
CPLAddXMLChild
cpl_minixml.h, 247
CPLAddXMLSibling
cpl_minixml.h, 248
CPLAssert
cpl_error.h, 214
CPLAtoGIntBig
cpl_conv.h, 179
CPLAtoGIntBigEx
cpl_conv.h, 180
CPLAtof
cpl_conv.h, 178
CPLAtofDelim
cpl_conv.h, 178
CPLAtofM
cpl_conv.h, 179
CPLBase64DecodeInPlace
cpl_string.h, 280
CPLBase64Encode
cpl_string.h, 280
CPLBinaryToHex
cpl_string.h, 280
CPLCalloc
cpl_conv.h, 180
CPLCheckForFile
cpl_conv.h, 181
CPLCleanTrailingSlash
cpl_conv.h, 181
CPLCleanXMLElementName
cpl_minixml.h, 248
CPLCloneXMLTree
cpl_minixml.h, 249
CPLCloseFileInZip
cpl_conv.h, 182
CPLCloseShared
cpl_conv.h, 182
CPLCloseZip
cpl_conv.h, 182
CPLCopyFile
cpl_conv.h, 182
CPLCopyTree
cpl_conv.h, 183
CPLCorrespondingPaths
cpl_conv.h, 183
CPLCreateFileInZip
cpl_conv.h, 183
CPLCreateXMLElementAndValue
cpl_minixml.h, 249
CPLCreateXMLNode
cpl_minixml.h, 250
CPLCreateZip
cpl_conv.h, 183
CPLDMSToDec
cpl_conv.h, 184
CPLDecToDMS
cpl_conv.h, 184
CPLDecToPackedDMS
cpl_conv.h, 184
CPLDefaultErrorHandler
cpl_error.h, 218
CPLDefaultFindFile
cpl_conv.h, 184
CPLDestroyXMLNode
cpl_minixml.h, 250
CPLDumpSharedList
cpl_conv.h, 185
CPLE_AWSAccessDenied
cpl_error.h, 214
CPLE_AWSBucketNotFound
cpl_error.h, 214
CPLE_AWSInvalidCredentials
cpl_error.h, 215
CPLE_AWSObjectNotFound
cpl_error.h, 215
CPLE_AWSSignatureDoesNotMatch

cpl_error.h, 215
CPLE_AppDefined
cpl_error.h, 214
CPLE_AssertionFailed
cpl_error.h, 214
CPLE_FileIO
cpl_error.h, 215
CPLE_HttpResponse
cpl_error.h, 215
CPLE_IllegalArg
cpl_error.h, 215
CPLE_NoWriteAccess
cpl_error.h, 216
CPLE_None
cpl_error.h, 215
CPLE_NotSupported
cpl_error.h, 215
CPLE_ObjectNull
cpl_error.h, 216
CPLE_OpenFailed
cpl_error.h, 216
CPLE_OutOfMemory
cpl_error.h, 216
CPLE_UserInterrupt
cpl_error.h, 216
CPLES_BackslashQuotable
cpl_string.h, 278
CPLES_CSV_FORCE_QUOTING
cpl_string.h, 278
CPLES_CSV
cpl_string.h, 278
CPLES_SQL
cpl_string.h, 278
CPLES_URL
cpl_string.h, 278
CPLES_XML_BUT_QUOTES
cpl_string.h, 279
CPLES_XML
cpl_string.h, 279
CPLEmergencyError
cpl_error.h, 218
CPLEncodingCharSize
cpl_string.h, 281
CPLErr
cpl_error.h, 217
CPLErrorContext, 45
CPLErrorHandler
cpl_error.h, 217
CPLErrorNum
cpl_error.h, 217
CPLErrorReset
cpl_error.h, 218
CPLErrorSetState
cpl_error.h, 218
CPLErrorV
cpl_error.h, 219
CPLEscapeString
cpl_string.h, 281
CPLExpandTilde
cpl_conv.h, 185
CPLExtractRelativePath
cpl_conv.h, 185
CPLFGets
cpl_conv.h, 186
CPLFileFinder
cpl_conv.h, 177
CPLFindFile
cpl_conv.h, 187
CPLFinderClean
cpl_conv.h, 186
CPLForceToASCII
cpl_string.h, 282
CPLFormCIFilename
cpl_conv.h, 187
CPLFormFilename
cpl_conv.h, 187
CPLFree
cpl_conv.h, 177
CPLGenerateTempFilename
cpl_conv.h, 188
CPLGetBasename
cpl_conv.h, 188
CPLGetConfigOption
cpl_conv.h, 189
CPLGetConfigOptions
cpl_conv.h, 189
CPLGetCurrentDir
cpl_conv.h, 190
CPLGetDirname
cpl_conv.h, 190
CPLGetErrorCounter
cpl_error.h, 219
CPLGetErrorHandlerUserData
cpl_error.h, 219
CPLGetExtension
cpl_conv.h, 191
CPLGetFilename
cpl_conv.h, 191
CPLGetHomeDir
cpl_conv.h, 192
CPLGetLastErrorMsg
cpl_error.h, 220
CPLGetLastErrorNo
cpl_error.h, 220
CPLGetLastErrorType
cpl_error.h, 220
CPLGetPageSize
cpl_virtualmem.h, 304
CPLGetPath
cpl_conv.h, 192
CPLGetPhysicalRAM
cpl_vsi.h, 319
CPLGetSharedList
cpl_conv.h, 193
CPLGetSymbol
cpl_conv.h, 193

CPLGetThreadLocalConfigOption
cpl_conv.h, 194
CPLGetThreadLocalConfigOptions
cpl_conv.h, 194
CPLGetUsablePhysicalRAM
cpl_vsi.h, 319
CPLGetValueType
cpl_string.h, 283
CPLGetXMLNode
cpl_minixml.h, 250
CPLGetXMLValue
cpl_minixml.h, 251
CPLHTTPDestroyMultiResult
cpl_http.h, 232
CPLHTTPDestroyResult
cpl_http.h, 233
CPLHTTPEnabled
cpl_http.h, 233
CPLHTTPFetch
cpl_http.h, 233
CPLHTTPFetchEx
cpl_http.h, 236
CPLHTTPMultiFetch
cpl_http.h, 236
CPLHTTPParseMultipartMime
cpl_http.h, 237
CPLHTTPResult, 45
nDataAlloc, 46
nDataLen, 46
nMimePartCount, 46
nStatus, 46
pabyData, 46
papszHeaders, 46
pasMimePart, 46
pszContentType, 47
pszErrBuf, 47
CPLHashSet
cpl_hash_set.h, 225
CPLHashSetClear
cpl_hash_set.h, 225
CPLHashSetDestroy
cpl_hash_set.h, 226
CPLHashSetEqualFunc
cpl_hash_set.h, 225
CPLHashSetEqualPointer
cpl_hash_set.h, 226
CPLHashSetEqualStr
cpl_hash_set.h, 226
CPLHashSetForeach
cpl_hash_set.h, 228
CPLHashSetFreeEltFunc
cpl_hash_set.h, 225
CPLHashSetHashFunc
cpl_hash_set.h, 225
CPLHashSetHashPointer
cpl_hash_set.h, 228
CPLHashSetHashStr
cpl_hash_set.h, 228
CPLHashSetInsert
cpl_hash_set.h, 229
CPLHashSetIterEltFunc
cpl_hash_set.h, 225
CPLHashSetLookup
cpl_hash_set.h, 229
CPLHashSetNew
cpl_hash_set.h, 230
CPLHashSetRemove
cpl_hash_set.h, 230
CPLHashSetRemoveDeferRehash
cpl_hash_set.h, 231
CPLHashSetSize
cpl_hash_set.h, 231
CPLHexToBinary
cpl_string.h, 283
CPLIsFilenameRelative
cpl_conv.h, 194
CPLIsNan
cpl_port.h, 265
CPLIsPowerOfTwo
cpl_conv.h, 195
CPLIsUTF8
cpl_string.h, 283
CPLIsVirtualMemFileMapAvailable
cpl_virtualmem.h, 304
CPLJSONArray, 47
Add, 48–50
operator[], 50, 52
Size, 52
CPLJSONDocument, 53
GetRoot, 53
Load, 53
LoadChunks, 54
LoadMemory, 54, 55
LoadUrl, 55
Save, 56
SaveAsString, 56
CPLJSONObject, 57
Add, 59–61
AddNull, 62
Deinit, 62
Delete, 62
Format, 63
GetArray, 63
GetBool, 64
GetChildren, 64
GetDouble, 64
GetInteger, 65
GetLong, 65
GetObj, 66
GetString, 66
GetType, 67
IsValid, 67
operator[], 67
PrettyFormat, 58
Set, 68–70
SetNull, 70

ToArray, 70
ToBool, 71
ToDouble, 71
ToInteger, 72
ToLong, 72
ToString, 72
Type, 59
CPLList
cpl_list.h, 241
CPLListAppend
cpl_list.h, 241
CPLListCount
cpl_list.h, 242
CPLListDestroy
cpl_list.h, 242
CPLListGet
cpl_list.h, 242
CPLListGetData
cpl_list.h, 243
CPLListGetLast
cpl_list.h, 243
CPLListGetNext
cpl_list.h, 243
CPLListInsert
cpl_list.h, 244
CPLListRemove
cpl_list.h, 244
CPLLoggingErrorHandler
cpl_error.h, 221
CPLMD5Context, 73
CPLMalloc
cpl_conv.h, 195
CPLMimePart, 73
nDataLen, 74
pabyData, 74
papszHeaders, 74
CPLMoveFile
cpl_conv.h, 195
CPLODBCDriverInstaller, 74
GetLastError, 75
GetLastErrorCode, 75
GetPathOut, 75
GetUsageCount, 75
InstallDriver, 75
RemoveDriver, 76
CPLODBCSession, 76
~CPLODBCSession, 77
BeginTransaction, 77
CPLODBCSession, 77
ClearTransaction, 77
CloseSession, 77
CommitTransaction, 77
EstablishSession, 77
Failed, 78
GetConnection, 78
GetEnvironment, 78
GetLastError, 78
IsInTransaction, 78
RollbackTransaction, 79
CPLODBCStatement, 79
~CPLODBCStatement, 80
Append, 80, 81
AppendEscaped, 81
Appendf, 81
CPLODBCStatement, 80
Clear, 82
ClearColumnData, 82
CollectResultsInfo, 82
DumpResult, 82
ExecuteSQL, 83
Fetch, 83
GetColColumnDef, 84
GetColCount, 84
GetColData, 84, 85
GetColDataLength, 85
GetColId, 85
GetColName, 86
GetColNullable, 86
GetColPrecision, 86
GetColSize, 88
GetColType, 88
GetColTypeName, 88
GetColumns, 89
GetCommand, 89
GetPrimaryKeys, 89
GetRowCountAffected, 90
GetStatement, 90
GetTables, 90
GetTypeMapping, 91
GetTypeName, 91
CPLOpenShared
cpl_conv.h, 196
CPLPackedDMSToDec
cpl_conv.h, 196
CPLParseNameValue
cpl_string.h, 284
CPLParseXMLFile
cpl_minixml.h, 252
CPLParseXMLString
cpl_minixml.h, 252
CPLPopErrorHandler
cpl_error.h, 221
CPLPopFileFinder
cpl_conv.h, 197
CPLPopFinderLocation
cpl_conv.h, 197
CPLPrintInt32
cpl_conv.h, 197
CPLPrintPointer
cpl_conv.h, 198
CPLPrintString
cpl_conv.h, 198
CPLPrintStringFill
cpl_conv.h, 199
CPLPrintTime
cpl_conv.h, 199

CPLPrintUIntBig
cpl_conv.h, 200
CPLProjectRelativeFilename
cpl_conv.h, 200
CPLPushErrorHandler
cpl_error.h, 221
CPLPushErrorHandlerEx
cpl_error.h, 222
CPLPushFileFinder
cpl_conv.h, 201
CPLPushFinderLocation
cpl_conv.h, 201
CPLQuadTree
cpl_quad_tree.h, 270
CPLQuadTreeCreate
cpl_quad_tree.h, 270
CPLQuadTreeDestroy
cpl_quad_tree.h, 271
CPLQuadTreeDump
cpl_quad_tree.h, 271
CPLQuadTreeDumpFeatureFunc
cpl_quad_tree.h, 270
CPLQuadTreeForeach
cpl_quad_tree.h, 271
CPLQuadTreeForeachFunc
cpl_quad_tree.h, 270
CPLQuadTreeGetAdvisedMaxDepth
cpl_quad_tree.h, 271
CPLQuadTreeGetBoundsFunc
cpl_quad_tree.h, 270
CPLQuadTreeGetStats
cpl_quad_tree.h, 272
CPLQuadTreeInsert
cpl_quad_tree.h, 272
CPLQuadTreeInsertWithBounds
cpl_quad_tree.h, 272
CPLQuadTreeSearch
cpl_quad_tree.h, 274
CPLQuadTreeSetBucketCapacity
cpl_quad_tree.h, 274
CPLQuadTreeSetMaxDepth
cpl_quad_tree.h, 274
CPLQuietErrorHandler
cpl_error.h, 222
CPLReadDir
cpl_vsi.h, 316
CPLReadLine
cpl_conv.h, 201
CPLReadLineL
cpl_conv.h, 202
CPLRealloc
cpl_conv.h, 202
CPLRecode
cpl_string.h, 284
CPLRecodeFromWChar
cpl_string.h, 285
CPLRecodeToWChar
cpl_string.h, 286
CPLRectObj, 92
maxx, 92
maxy, 92
minx, 92
miny, 92
CPLRemoveXMLChild
cpl_minixml.h, 253
CPLResetExtension
cpl_conv.h, 203
CPLScanDouble
cpl_conv.h, 203
CPLScanLong
cpl_conv.h, 203
CPLScanPointer
cpl_conv.h, 204
CPLScanString
cpl_conv.h, 204
CPLScanUIntBig
cpl_conv.h, 205
CPLScanULong
cpl_conv.h, 205
CPLSearchXMLNode
cpl_minixml.h, 253
CPLSerializeXMLTree
cpl_minixml.h, 254
CPLSerializeXMLTreeToFile
cpl_minixml.h, 254
CPLSetConfigOption
cpl_conv.h, 206
CPLSetConfigOptions
cpl_conv.h, 206
CPLSetCurrentErrorHandlerCatchDebug
cpl_error.h, 222
CPLSetErrorHandler
cpl_error.h, 223
CPLSetErrorHandlerEx
cpl_error.h, 223
CPLSetThreadLocalConfigOption
cpl_conv.h, 207
CPLSetThreadLocalConfigOptions
cpl_conv.h, 208
CPLSetXMLValue
cpl_minixml.h, 254
CPLSharedFileInfo, 93
bLarge, 93
fp, 93
nRefCount, 93
pszAccess, 94
pszFilename, 94
CPLSharedFileInfoExtra, 94
CPLStat
cpl_conv.h, 208
CPLStdCallThreadInfo, 94
CPLStrdup
cpl_conv.h, 208
CPLStringToComplex
cpl_conv.h, 209
CPLStripXMLNamespace

cpl_minixml.h, 255
CPLStrlcat
cpl_string.h, 286
CPLStrlcpy
cpl_string.h, 287
CPLStrlenUTF8
cpl_string.h, 288
CPLStrlwr
cpl_conv.h, 209
CPLStrnlen
cpl_string.h, 288
CPLStrtod
cpl_conv.h, 209
CPLStrtodDelim
cpl_conv.h, 210
CPLStrtof
cpl_conv.h, 210
CPLStrtofDelim
cpl_conv.h, 211
CPLSymlink
cpl_conv.h, 211
CPLTestBoolean
cpl_string.h, 289
CPLTurnFailureIntoWarning
cpl_error.h, 224
CPLUnescapeString
cpl_string.h, 289
CPLUnlinkTree
cpl_conv.h, 211
CPLValueType
cpl_string.h, 280
CPLVirtualMem, 94
cpl_virtualmem.h, 302
CPLVirtualMemAccessMode
cpl_virtualmem.h, 303
CPLVirtualMemCachePageCbk
cpl_virtualmem.h, 302
CPLVirtualMemDeclareThread
cpl_virtualmem.h, 304
CPLVirtualMemDerivedNew
cpl_virtualmem.h, 305
CPLVirtualMemFileMapNew
cpl_virtualmem.h, 306
CPLVirtualMemFree
cpl_virtualmem.h, 306
CPLVirtualMemFreeUserData
cpl_virtualmem.h, 303
CPLVirtualMemGetAccessMode
cpl_virtualmem.h, 307
CPLVirtualMemGetAddr
cpl_virtualmem.h, 307
CPLVirtualMemGetPageSize
cpl_virtualmem.h, 308
CPLVirtualMemGetSize
cpl_virtualmem.h, 308
CPLVirtualMemIsAccessThreadSafe
cpl_virtualmem.h, 309
CPLVirtualMemIsFileMapping
cpl_virtualmem.h, 309
CPLVirtualMemManagerTerminate
cpl_virtualmem.h, 310
CPLVirtualMemNew
cpl_virtualmem.h, 310
CPLVirtualMemPin
cpl_virtualmem.h, 311
CPLVirtualMemUnCachePageCbk
cpl_virtualmem.h, 303
CPLVirtualMemUnDeclareThread
cpl_virtualmem.h, 312
CPLWorkerThread, 95
CPLWorkerThreadJob, 95
CPLWorkerThreadPool, 95
~CPLWorkerThreadPool, 95
CPLWorkerThreadPool, 95
GetThreadCount, 96
Setup, 96
SubmitJob, 96
SubmitJobs, 97
WaitCompletion, 97
CPLWriteFileInZip
cpl_conv.h, 212
CPLXMLNode, 98
cpl_minixml.h, 246
eType, 98
psChild, 98
psNext, 98
pszValue, 99
CPLXMLNodeType
cpl_minixml.h, 246
CPLZLibInflate
cpl_conv.h, 212
CPLZip, 99
CPLsetlocale
cpl_conv.h, 207
CSLAddNameValue
cpl_string.h, 290
CSLAddString
cpl_string.h, 290
CSLAddStringMayFail
cpl_string.h, 290
CSLConstList
cpl_port.h, 267
CSLCount
cpl_string.h, 290
CSLDestroy
cpl_string.h, 291
CSLDuplicate
cpl_string.h, 291
CSLFetchBoolean
cpl_string.h, 291
CSLFetchNameValue
cpl_string.h, 292
CSLFetchNameValueDef
cpl_string.h, 292
CSLFetchNameValueMultiple
cpl_string.h, 292

CSLFindName
cpl_string.h, 292
CSLFindString
cpl_string.h, 293
CSLFindStringCaseSensitive
cpl_string.h, 293
CSLGetField
cpl_string.h, 294
CSLInsertString
cpl_string.h, 294
CSLInsertStrings
cpl_string.h, 294
CSLLoad
cpl_string.h, 294
CSLLoad2
cpl_string.h, 296
CSLMerge
cpl_string.h, 296
CSLParseCommandLine
cpl_string.h, 297
CSLPartialFindString
cpl_string.h, 297
CSLPrint
cpl_string.h, 298
CSLRemoveStrings
cpl_string.h, 298
CSLSave
cpl_string.h, 298
CSLSetNameValue
cpl_string.h, 298
CSLSetNameValueSeparator
cpl_string.h, 299
CSLT_ALLOWEMPTYTOKENS
cpl_string.h, 279
CSLT_HONOURSTRINGS
cpl_string.h, 279
CSLT_PRESERVEESCAPES
cpl_string.h, 279
CSLT_PRESERVEQUOTES
cpl_string.h, 279
CSLT_STRIPENDSPACES
cpl_string.h, 279
CSLT_STRIPLEADSPACES
cpl_string.h, 279
CSLTestBoolean
cpl_string.h, 299
CSLTokenizeString
cpl_string.h, 300
CSLTokenizeString2
cpl_string.h, 300
CSLTokenizeStringComplex
cpl_string.h, 301
Clear
CPLODBCStatement, 82
ClearColumnData
CPLODBCStatement, 82
ClearIndex
SDTSIndexedReader, 133
ClearTransaction
CPLODBCSession, 77
Clone
DDFRecord, 110
CloneOn
DDFRecord, 111
Close
DDFModule, 107
VSISparseFileHandle, 159
VSISubFileHandle, 163
VSIVirtualHandle, 167
CloseSession
CPLODBCSession, 77
CollectResultsInfo
CPLODBCStatement, 82
CommitTransaction
CPLODBCSession, 77
cpl_conv.h, 175
CPLAtoGIntBig, 179
CPLAtoGIntBigEx, 180
CPLAtof, 178
CPLAtofDelim, 178
CPLAtofM, 179
CPLCalloc, 180
CPLCheckForFile, 181
CPLCleanTrailingSlash, 181
CPLCloseFileInZip, 182
CPLCloseShared, 182
CPLCloseZip, 182
CPLCopyFile, 182
CPLCopyTree, 183
CPLCorrespondingPaths, 183
CPLCreateFileInZip, 183
CPLCreateZip, 183
CPLDMSToDec, 184
CPLDecToDMS, 184
CPLDecToPackedDMS, 184
CPLDefaultFindFile, 184
CPLDumpSharedList, 185
CPLExpandTilde, 185
CPLExtractRelativePath, 185
CPLFGets, 186
CPLFileFinder, 177
CPLFindFile, 187
CPLFinderClean, 186
CPLFormCIFilename, 187
CPLFormFilename, 187
CPLFree, 177
CPLGenerateTempFilename, 188
CPLGetBasename, 188
CPLGetConfigOption, 189
CPLGetConfigOptions, 189
CPLGetCurrentDir, 190
CPLGetDirname, 190
CPLGetExtension, 191
CPLGetFilename, 191
CPLGetHomeDir, 192
CPLGetPath, 192

CPLGetSharedList, 193
CPLGetSymbol, 193
CPLGetThreadLocalConfigOption, 194
CPLGetThreadLocalConfigOptions, 194
CPLIsFilenameRelative, 194
CPLIsPowerOfTwo, 195
CPLMalloc, 195
CPLMoveFile, 195
CPLOpenShared, 196
CPLPackedDMSToDec, 196
CPLPopFileFinder, 197
CPLPopFinderLocation, 197
CPLPrintInt32, 197
CPLPrintPointer, 198
CPLPrintString, 198
CPLPrintStringFill, 199
CPLPrintTime, 199
CPLPrintUIntBig, 200
CPLProjectRelativeFilename, 200
CPLPushFileFinder, 201
CPLPushFinderLocation, 201
CPLReadLine, 201
CPLReadLineL, 202
CPLRealloc, 202
CPLResetExtension, 203
CPLScanDouble, 203
CPLScanLong, 203
CPLScanPointer, 204
CPLScanString, 204
CPLScanUIntBig, 205
CPLScanULong, 205
CPLSetConfigOption, 206
CPLSetConfigOptions, 206
CPLSetThreadLocalConfigOption, 207
CPLSetThreadLocalConfigOptions, 208
CPLStat, 208
CPLStrdup, 208
CPLStringToComplex, 209
CPLStrlwr, 209
CPLStrtod, 209
CPLStrtodDelim, 210
CPLStrtof, 210
CPLStrtofDelim, 211
CPLSymlink, 211
CPLUnlinkTree, 211
CPLWriteFileInZip, 212
CPLZLibInflate, 212
CPLsetlocale, 207
cpl_error.h, 213
_CPLAssert, 217
CPLAssert, 214
CPLDefaultErrorHandler, 218
CPLE_AWSAccessDenied, 214
CPLE_AWSBucketNotFound, 214
CPLE_AWSInvalidCredentials, 215
CPLE_AWSObjectNotFound, 215
CPLE_AWSSignatureDoesNotMatch, 215
CPLE_AppDefined, 214
CPLE_AssertionFailed, 214
CPLE_FileIO, 215
CPLE_HttpResponse, 215
CPLE_IllegalArg, 215
CPLE_NoWriteAccess, 216
CPLE_None, 215
CPLE_NotSupported, 215
CPLE_ObjectNull, 216
CPLE_OpenFailed, 216
CPLE_OutOfMemory, 216
CPLE_UserInterrupt, 216
CPLEmergencyError, 218
CPLErr, 217
CPLErrorHandler, 217
CPLErrorNum, 217
CPLErrorReset, 218
CPLErrorSetState, 218
CPLErrorV, 219
CPLGetErrorCounter, 219
CPLGetErrorHandlerUserData, 219
CPLGetLastErrorMsg, 220
CPLGetLastErrorNo, 220
CPLGetLastErrorType, 220
CPLLoggingErrorHandler, 221
CPLPopErrorHandler, 221
CPLPushErrorHandler, 221
CPLPushErrorHandlerEx, 222
CPLQuietErrorHandler, 222
CPLSetCurrentErrorHandlerCatchDebug, 222
CPLSetErrorHandler, 223
CPLSetErrorHandlerEx, 223
CPLTurnFailureIntoWarning, 224
VALIDATE_POINTER0, 216
VALIDATE_POINTER1, 216
cpl_hash_set.h, 224
CPLHashSet, 225
CPLHashSetClear, 225
CPLHashSetDestroy, 226
CPLHashSetEqualFunc, 225
CPLHashSetEqualPointer, 226
CPLHashSetEqualStr, 226
CPLHashSetForeach, 228
CPLHashSetFreeEltFunc, 225
CPLHashSetHashFunc, 225
CPLHashSetHashPointer, 228
CPLHashSetHashStr, 228
CPLHashSetInsert, 229
CPLHashSetIterEltFunc, 225
CPLHashSetLookup, 229
CPLHashSetNew, 230
CPLHashSetRemove, 230
CPLHashSetRemoveDeferRehash, 231
CPLHashSetSize, 231
cpl_http.h, 232
CPLHTTPDestroyMultiResult, 232
CPLHTTPDestroyResult, 233
CPLHTTPEnabled, 233
CPLHTTPFetch, 233

CPLHTTPFetchEx, 236
CPLHTTPMultiFetch, 236
CPLHTTPParseMultipartMime, 237
GOA2GetAccessTokenFromCloudEngineVM, 237
GOA2GetAccessTokenFromServiceAccount, 238
GOA2GetAuthorizationURL, 239
GOA2GetRefreshToken, 239
cpl_json.h, 240
cpl_list.h, 240
CPLList, 241
CPLListAppend, 241
CPLListCount, 242
CPLListDestroy, 242
CPLListGet, 242
CPLListGetData, 243
CPLListGetLast, 243
CPLListGetNext, 243
CPLListInsert, 244
CPLListRemove, 244
cpl_minixml.h, 245
CPLAddXMLAttributeAndValue, 247
CPLAddXMLChild, 247
CPLAddXMLSibling, 248
CPLCleanXMLElementName, 248
CPLCloneXMLTree, 249
CPLCreateXMLElementAndValue, 249
CPLCreateXMLNode, 250
CPLDestroyXMLNode, 250
CPLGetXMLNode, 250
CPLGetXMLValue, 251
CPLParseXMLFile, 252
CPLParseXMLString, 252
CPLRemoveXMLChild, 253
CPLSearchXMLNode, 253
CPLSerializeXMLTree, 254
CPLSerializeXMLTreeToFile, 254
CPLSetXMLValue, 254
CPLStripXMLNamespace, 255
CPLXMLNode, 246
CPLXMLNodeType, 246
cpl_odbc.h, 256
cpl_port.h, 256
ABS, 258
CPL_FALLTHROUGH, 258
CPL_FORMAT_STRING, 258
CPL_FRMT_GB_WITHOUT_PREFIX, 258
CPL_FRMT_GIB, 258
CPL_FRMT_GUIB, 258
CPL_LSBINT16PTR, 259
CPL_LSBINT32PTR, 259
CPL_LSBPTR16, 259
CPL_LSBPTR32, 259
CPL_LSBPTR64, 259
CPL_LSBSINT16PTR, 260
CPL_LSBSINT32PTR, 260
CPL_LSBUINT16PTR, 260
CPL_LSBUINT32PTR, 260
CPL_LSBWORD16, 260
CPL_LSBWORD32, 260
CPL_MSBPTR16, 260
CPL_MSBPTR32, 261
CPL_MSBPTR64, 261
CPL_MSBWORD16, 261
CPL_MSBWORD32, 261
CPL_NO_RETURN, 261
CPL_NULL_TERMINATED, 261
CPL_PRINT_FUNC_FORMAT, 261
CPL_RESTRICT, 262
CPL_RETURNS_NONNULL, 262
CPL_SCAN_FUNC_FORMAT, 262
CPL_SCANF_FORMAT_STRING, 262
CPL_SWAP16, 262
CPL_SWAP16PTR, 262
CPL_SWAP32, 263
CPL_SWAP32PTR, 263
CPL_SWAP64, 263
CPL_SWAP64PTR, 264
CPL_SWAPDOUBLE, 264
CPL_UNUSED, 264
CPL_WARN_UNUSED_RESULT, 264
CPLIsNan, 265
CSLConstList, 267
EQUALN, 265
EQUAL, 265
GBool, 267
GByte, 267
GINT64_MAX, 265
GINT64_MIN, 265
GINTBIG_MAX, 265
GINTBIG_MIN, 265
GInt16, 267
GInt32, 268
GInt64, 268
GIntBig, 268
GPtrDiff_t, 268
GUINT64_MAX, 266
GUINTBIG_MAX, 266
GUInt16, 268
GUInt32, 268
GUInt64, 268
GUIntBig, 268
M_PI, 266
MAX, 266
MIN, 266
STARTS_WITH_CI, 266
STARTS_WITH, 266
STRCASECMP, 267
STRNCASECMP, 267
cpl_quad_tree.h, 269
CPLQuadTree, 270
CPLQuadTreeCreate, 270
CPLQuadTreeDestroy, 271
CPLQuadTreeDump, 271
CPLQuadTreeDumpFeatureFunc, 270
CPLQuadTreeForeach, 271
CPLQuadTreeForeachFunc, 270

CPLQuadTreeGetAdvisedMaxDepth, 271
CPLQuadTreeGetBoundsFunc, 270
CPLQuadTreeGetStats, 272
CPLQuadTreeInsert, 272
CPLQuadTreeInsertWithBounds, 272
CPLQuadTreeSearch, 274
CPLQuadTreeSetBucketCapacity, 274
CPLQuadTreeSetMaxDepth, 274
cpl_string.h, 275
CPL_ENC_ASCII, 277
CPL_ENC_ISO8859_1, 277
CPL_ENC_LOCALE, 277
CPL_ENC_UCS2, 277
CPL_ENC_UCS4, 278
CPL_ENC_UTF16, 278
CPL_ENC_UTF8, 278
CPLBase64DecodeInPlace, 280
CPLBase64Encode, 280
CPLBinaryToHex, 280
CPLES_BackslashQuotable, 278
CPLES_CSV_FORCE_QUOTING, 278
CPLES_CSV, 278
CPLES_SQL, 278
CPLES_URL, 278
CPLES_XML_BUT_QUOTES, 279
CPLES_XML, 279
CPLEncodingCharSize, 281
CPLEscapeString, 281
CPLForceToASCII, 282
CPLGetValueType, 283
CPLHexToBinary, 283
CPLIsUTF8, 283
CPLParseNameValue, 284
CPLRecode, 284
CPLRecodeFromWChar, 285
CPLRecodeToWChar, 286
CPLStrlcat, 286
CPLStrlcpy, 287
CPLStrlenUTF8, 288
CPLStrnlen, 288
CPLTestBoolean, 289
CPLUnescapeString, 289
CPLValueType, 280
CSLAddNameValue, 290
CSLAddString, 290
CSLAddStringMayFail, 290
CSLCount, 290
CSLDestroy, 291
CSLDuplicate, 291
CSLFetchBoolean, 291
CSLFetchNameValue, 292
CSLFetchNameValueDef, 292
CSLFetchNameValueMultiple, 292
CSLFindName, 292
CSLFindString, 293
CSLFindStringCaseSensitive, 293
CSLGetField, 294
CSLInsertString, 294
CSLInsertStrings, 294
CSLLoad, 294
CSLLoad2, 296
CSLMerge, 296
CSLParseCommandLine, 297
CSLPartialFindString, 297
CSLPrint, 298
CSLRemoveStrings, 298
CSLSave, 298
CSLSetNameValue, 298
CSLSetNameValueSeparator, 299
CSLT_ALLOWEMPTYTOKENS, 279
CSLT_HONOURSTRINGS, 279
CSLT_PRESERVEESCAPES, 279
CSLT_PRESERVEQUOTES, 279
CSLT_STRIPENDSPACES, 279
CSLT_STRIPLEADSPACES, 279
CSLTestBoolean, 299
CSLTokenizeString, 300
CSLTokenizeString2, 300
CSLTokenizeStringComplex, 301
cpl_virtualmem.h, 301
CPLGetPageSize, 304
CPLIsVirtualMemFileMapAvailable, 304
CPLVirtualMem, 302
CPLVirtualMemAccessMode, 303
CPLVirtualMemCachePageCbk, 302
CPLVirtualMemDeclareThread, 304
CPLVirtualMemDerivedNew, 305
CPLVirtualMemFileMapNew, 306
CPLVirtualMemFree, 306
CPLVirtualMemFreeUserData, 303
CPLVirtualMemGetAccessMode, 307
CPLVirtualMemGetAddr, 307
CPLVirtualMemGetPageSize, 308
CPLVirtualMemGetSize, 308
CPLVirtualMemIsAccessThreadSafe, 309
CPLVirtualMemIsFileMapping, 309
CPLVirtualMemManagerTerminate, 310
CPLVirtualMemNew, 310
CPLVirtualMemPin, 311
CPLVirtualMemUnCachePageCbk, 303
CPLVirtualMemUnDeclareThread, 312
cpl_vsi.h, 312
CPLGetPhysicalRAM, 319
CPLGetUsablePhysicalRAM, 319
CPLReadDir, 316
VSI_CALLOC_VERBOSE, 316
VSI_ISDIR, 316
VSI_ISLNK, 316
VSI_ISREG, 316
VSI_L_OFFSET_MAX, 316
VSI_MALLOC2_VERBOSE, 316
VSI_MALLOC3_VERBOSE, 317
VSI_MALLOC_ALIGNED_AUTO_VERBOSE, 317
VSI_MALLOC_VERBOSE, 317
VSI_REALLOC_VERBOSE, 317
VSI_STAT_EXISTS_FLAG, 317

VSI_STAT_NATURE_FLAG, 317
VSI_STAT_SET_ERROR_FLAG, 318
VSI_STAT_SIZE_FLAG, 318
VSI_STRDUP_VERBOSE, 318
VSICalloc, 320
VSICallocVerbose, 320
VSIFCloseL, 320
VSIFEofL, 321
VSIFFlushL, 321
VSIFGetNativeFileDescriptorL, 322
VSIFGetRangeStatusL, 322
VSIFOpenExL, 323
VSIFOpenL, 324
VSIFPutcL, 325
VSIFReadMultiRangeL, 326
VSIFReadL, 325
VSIFSeekL, 327
VSIFTellL, 328
VSIFTruncateL, 328
VSIFWriteL, 329
VSIFileFromMemBuffer, 323
VSIFree, 326
VSIFreeAligned, 327
VSIGetActualURL, 329
VSIGetDiskFreeSpace, 330
VSIGetFileSystemOptions, 330
VSIGetFileSystemsPrefixes, 331
VSIGetMemFileBuffer, 331
VSIGetSignedURL, 332
VSIHasOptimizedReadMultiRange, 332
VSIIngestFile, 333
VSIInstallGZipFileHandler, 334
VSIInstallMemFileHandler, 334
VSIInstallSparseFileHandler, 334
VSIInstallStdinHandler, 335
VSIInstallStdoutHandler, 335
VSIInstallSubFileHandler, 335
VSIInstallTarFileHandler, 336
VSIInstallZipFileHandler, 336
VSIIsCaseSensitiveFS, 337
VSILFILE, 318
VSIMalloc, 338
VSIMalloc2, 338
VSIMalloc2Verbose, 338
VSIMalloc3, 338
VSIMalloc3Verbose, 338
VSIMallocAligned, 339
VSIMallocAlignedAuto, 339
VSIMallocAlignedAutoVerbose, 340
VSIMallocVerbose, 340
VSIMkdir, 340
VSIMkdirRecursive, 341
VSIRangeStatus, 319
VSIReadDir, 341
VSIReadDirEx, 342
VSIReadDirRecursive, 342
VSIRealloc, 343
VSIReallocVerbose, 343
VSIRename, 343
VSIRewindL, 344
VSIRmdir, 344
VSIRmdirRecursive, 345
VSIStatBufL, 318
VSIStatExL, 345
VSIStatL, 346
VSIStdoutSetRedirection, 346
VSIStrdup, 347
VSIStrdupVerbose, 347
VSIStrerror, 347
VSISupportsSparseFiles, 347
VSIUnlink, 348
VSIWriteFunction, 318
vsi_l_offset, 318
cpl_worker_thread_pool.h, 348
CreateDefaultFieldInstance
DDFRecord, 111
ctb, 99
curfile_info, 100

DDFBinaryFormat
DDFSubfieldDefn, 119
DDFField, 100
Dump, 100
GetData, 101
GetDataSize, 101
GetFieldDefn, 101
GetInstanceData, 101
GetRepeatCount, 101
GetSubfieldData, 102
DDFFieldDefn, 102
Dump, 103
FindSubfieldDefn, 103
GetDefaultValue, 104
GetDescription, 104
GetFixedWidth, 104
GetName, 104
GetSubfield, 104
GetSubfieldCount, 105
IsRepeating, 105
SetRepeatingFlag, 105
DDFModule, 105
~DDFModule, 106
AddField, 106
Close, 107
DDFModule, 106
Dump, 107
FindFieldDefn, 107
GetField, 107
GetFieldCount, 108
Open, 108
ReadRecord, 108
Rewind, 109
DDFRecord, 109
AddField, 110
Clone, 110
CloneOn, 111
CreateDefaultFieldInstance, 111

DeleteField, 112
Dump, 112
FindField, 112
GetData, 113
GetDataSize, 113
GetField, 113
GetFieldCount, 113
GetFloatSubfield, 113
GetIntSubfield, 114
GetModule, 115
GetStringSubfield, 115
ResizeField, 115
SetFieldRaw, 116
SetFloatSubfield, 116
SetIntSubfield, 117
SetStringSubfield, 117
Write, 118
DDFSubfieldDefn, 118
DDFBinaryFormat, 119
Dump, 119
DumpData, 119
ExtractFloatData, 120
ExtractIntData, 120
ExtractStringData, 121
FormatFloatValue, 122
FormatIntValue, 122
FormatStringValue, 122
GetDataLength, 122
GetDefaultValue, 123
GetFormat, 123
GetName, 123
GetType, 124
GetWidth, 124
DefaultCSVFileNameTLS, 124
Deinit
CPLJSONObject, 62
Delete
CPLJSONObject, 62
DeleteField
DDFRecord, 112
dfX
SDTSRawPoint, 145
dfY
SDTSRawPoint, 146
dfZ
SDTSRawPoint, 146
Dump
DDFField, 100
DDFFieldDefn, 103
DDFModule, 107
DDFRecord, 112
DDFSubfieldDefn, 119
SDTSAttrRecord, 130
SDTSFeature, 132
SDTSRawLine, 143
SDTSRawPoint, 145
SDTSRawPolygon, 148
DumpData
DDFSubfieldDefn, 119
DumpResult
CPLODBCStatement, 82

EQUALN
cpl_port.h, 265
EQUAL
cpl_port.h, 265
eType
CPLXMLNode, 98
Eof
VSISparseFileHandle, 159
VSISubFileHandle, 163
VSIVirtualHandle, 167
errHandler, 124
EstablishSession
CPLODBCSession, 77
ExecuteSQL
CPLODBCStatement, 83
ExtractFloatData
DDFSubfieldDefn, 120
ExtractIntData
DDFSubfieldDefn, 120
ExtractStringData
DDFSubfieldDefn, 121

Failed
CPLODBCSession, 78
Fetch
CPLODBCStatement, 83
file_in_zip_read_info_s, 125
FillIndex
SDTSIndexedReader, 133
FindField
DDFRecord, 112
FindFieldDefn
DDFModule, 107
FindFileTLS, 125
FindLayer
SDTSTransfer, 150
FindSubfieldDefn
DDFFieldDefn, 103
Flush
VSIVirtualHandle, 168
Format
CPLJSONObject, 63
FormatFloatValue
DDFSubfieldDefn, 122
FormatIntValue
DDFSubfieldDefn, 122
FormatStringValue
DDFSubfieldDefn, 122
fp
CPLSharedFileInfo, 93

GBool
cpl_port.h, 267
GByte
cpl_port.h, 267

GDALScaledProgressInfo, 125
GINT64_MAX
cpl_port.h, 265
GINT64_MIN
cpl_port.h, 265
GINTBIG_MAX
cpl_port.h, 265
GINTBIG_MIN
cpl_port.h, 265
GInt16
cpl_port.h, 267
GInt32
cpl_port.h, 268
GInt64
cpl_port.h, 268
GIntBig
cpl_port.h, 268
GOA2GetAccessTokenFromCloudEngineVM
cpl_http.h, 237
GOA2GetAccessTokenFromServiceAccount
cpl_http.h, 238
GOA2GetAuthorizationURL
cpl_http.h, 239
GOA2GetRefreshToken
cpl_http.h, 239
GPtrDiff_t
cpl_port.h, 268
GUINT64_MAX
cpl_port.h, 266
GUINTBIG_MAX
cpl_port.h, 266
GUInt16
cpl_port.h, 268
GUInt32
cpl_port.h, 268
GUInt64
cpl_port.h, 268
GUIntBig
cpl_port.h, 268
GetArray
CPLJSONObject, 63
GetAttr
SDTSTransfer, 151
GetBlock
SDTSRasterReader, 140
GetBlockXSize
SDTSRasterReader, 140
GetBlockYSize
SDTSRasterReader, 140
GetBool
CPLJSONObject, 64
GetBounds
SDTSTransfer, 151
GetCATD
SDTSTransfer, 152
GetChildren
CPLJSONObject, 64
GetColColumnDef
CPLODBCStatement, 84
GetColCount
CPLODBCStatement, 84
GetColData
CPLODBCStatement, 84, 85
GetColDataLength
CPLODBCStatement, 85
GetColId
CPLODBCStatement, 85
GetColName
CPLODBCStatement, 86
GetColNullable
CPLODBCStatement, 86
GetColPrecision
CPLODBCStatement, 86
GetColSize
CPLODBCStatement, 88
GetColType
CPLODBCStatement, 88
GetColTypeName
CPLODBCStatement, 88
GetColumns
CPLODBCStatement, 89
GetCommand
CPLODBCStatement, 89
GetConnection
CPLODBCSession, 78
GetData
DDFField, 101
DDFRecord, 113
GetDataLength
DDFSubfieldDefn, 122
GetDataSize
DDFField, 101
DDFRecord, 113
GetDefaultValue
DDFFieldDefn, 104
DDFSubfieldDefn, 123
GetDescription
DDFFieldDefn, 104
GetDouble
CPLJSONObject, 64
GetEntryFilePath
SDTS_CATD, 126
GetEntryType
SDTS_CATD, 126
GetEntryTypeDesc
SDTS_CATD, 127
GetEnvironment
CPLODBCSession, 78
GetField
DDFModule, 107
DDFRecord, 113
GetFieldCount
DDFModule, 108
DDFRecord, 113
GetFieldDefn
DDFField, 101

GetFixedWidth
DDFFieldDefn, 104
GetFloatSubfield
DDFRecord, 113
GetFormat
DDFSubfieldDefn, 123
GetIndexedFeatureRef
SDTSIndexedReader, 133
GetInstanceData
DDFField, 101
GetIntSubfield
DDFRecord, 114
GetInteger
CPLJSONObject, 65
GetLastError
CPLODBCDriverInstaller, 75
CPLODBCSession, 78
GetLastErrorCode
CPLODBCDriverInstaller, 75
GetLayerCATDEntry
SDTSTransfer, 152
GetLayerIndexedReader
SDTSTransfer, 152
GetLayerRasterReader
SDTSTransfer, 153
GetLayerType
SDTSTransfer, 153
GetLong
CPLJSONObject, 65
GetMinMax
SDTSRasterReader, 140
GetModule
DDFRecord, 115
GetName
DDFFieldDefn, 104
DDFSubfieldDefn, 123
GetNativeFileDescriptor
VSIVirtualHandle, 168
GetNextFeature
SDTSIndexedReader, 134
GetObj
CPLJSONObject, 66
GetPathOut
CPLODBCDriverInstaller, 75
GetPrimaryKeys
CPLODBCStatement, 89
GetRangeStatus
VSIVirtualHandle, 168
GetRasterType
SDTSRasterReader, 141
GetRepeatCount
DDFField, 101
GetRoot
CPLJSONDocument, 53
GetRowCountAffected
CPLODBCStatement, 90
GetStatement
CPLODBCStatement, 90
GetString
CPLJSONObject, 66
GetStringSubfield
DDFRecord, 115
GetSubfield
DDFFieldDefn, 104
GetSubfieldCount
DDFFieldDefn, 105
GetSubfieldData
DDFField, 102
GetTables
CPLODBCStatement, 90
GetThreadCount
CPLWorkerThreadPool, 96
GetTransform
SDTSRasterReader, 141
GetType
CPLJSONObject, 67
DDFSubfieldDefn, 124
GetTypeMapping
CPLODBCStatement, 91
GetTypeName
CPLODBCStatement, 91
GetUsageCount
CPLODBCDriverInstaller, 75
GetWidth
DDFSubfieldDefn, 124
GetXREF
SDTSTransfer, 154
GetXSize
SDTSRasterReader, 141
GetYSize
SDTSRasterReader, 142

InstallDriver
CPLODBCDriverInstaller, 75
IsInTransaction
CPLODBCSession, 78
IsIndexed
SDTSIndexedReader, 134
IsRepeating
DDFFieldDefn, 105
IsSecondary
SDTSAttrReader, 129
IsValid
CPLJSONObject, 67

linkedlist_data_s, 125
linkedlist_datablock_internal_s, 125
Load
CPLJSONDocument, 53
LoadChunks
CPLJSONDocument, 54
LoadMemory
CPLJSONDocument, 54, 55
LoadUrl
CPLJSONDocument, 55

M_PI

cpl_port.h, 266
MAX
cpl_port.h, 266
MIN
cpl_port.h, 266
maxx
CPLRectObj, 92
maxy
CPLRectObj, 92
minx
CPLRectObj, 92
miny
CPLRectObj, 92

nAttributes
SDTSFeature, 132
nDataAlloc
CPLHTTPResult, 46
nDataLen
CPLHTTPResult, 46
CPLMimePart, 74
nMimePartCount
CPLHTTPResult, 46
nRecord
SDTSModId, 136
nRefCount
CPLSharedFileInfo, 93
nRings
SDTSRawPolygon, 148
nStatus
CPLHTTPResult, 46
nVertices
SDTSRawLine, 143
SDTSRawPolygon, 148
nZone
SDTS_XREF, 128

oAreaId
SDTSRawPoint, 146
oEndNode
SDTSRawLine, 143
oLeftPoly
SDTSRawLine, 143
oModId
SDTSFeature, 132
oRightPoly
SDTSRawLine, 144
oStartNode
SDTSRawLine, 144
Open
DDFModule, 108
SDTSTransfer, 154
operator[]
CPLJSONArray, 50, 52
CPLJSONObject, 67

pData
_CPLList, 43
pabyData
CPLHTTPResult, 46
CPLMimePart, 74
padfX
SDTSRawLine, 144
SDTSRawPolygon, 148
padfY
SDTSRawLine, 144
SDTSRawPolygon, 148
padfZ
SDTSRawLine, 144
SDTSRawPolygon, 149
panRingStart
SDTSRawPolygon, 149
paoATID
SDTSFeature, 132
papszHeaders
CPLHTTPResult, 46
CPLMimePart, 74
ParseContext, 125
pasMimePart
CPLHTTPResult, 46
poATTR
SDTSAttrRecord, 130
poWholeRecord
SDTSAttrRecord, 130
PrettyFormat
CPLJSONObject, 58
psChild
CPLXMLNode, 98
psNext
_CPLList, 44
CPLXMLNode, 98
pszAccess
CPLSharedFileInfo, 94
pszContentType
CPLHTTPResult, 47
pszDatum
SDTS_XREF, 128
pszErrBuf
CPLHTTPResult, 47
pszFilename
CPLSharedFileInfo, 94
pszSystemName
SDTS_XREF, 128
pszValue
CPLXMLNode, 99

Read
VSISparseFileHandle, 160
VSISubFileHandle, 164
VSIVirtualHandle, 169
ReadMultiRange
VSIVirtualHandle, 170
ReadRecord
DDFModule, 108
RemoveDriver
CPLODBCDriverInstaller, 76
ResizeField
DDFRecord, 115

Rewind
DDFModule, 109
SDTSIndexedReader, 134
RollbackTransaction
CPLODBCSession, 79

SDTS_CATDEntry, 127
SDTS_CATD, 125
GetEntryFilePath, 126
GetEntryType, 126
GetEntryTypeDesc, 127
SDTS_IREF, 127
SDTS_XREF, 128
nZone, 128
pszDatum, 128
pszSystemName, 128
SDTSAttrReader, 129
IsSecondary, 129
SDTSAttrRecord, 129
Dump, 130
poATTR, 130
poWholeRecord, 130
SDTSDataset, 131
SDTSFeature, 131
Dump, 132
nAttributes, 132
oModId, 132
paoATID, 132
SDTSIndexedReader, 133
ClearIndex, 133
FillIndex, 133
GetIndexedFeatureRef, 133
GetNextFeature, 134
IsIndexed, 134
Rewind, 134
ScanModuleReferences, 134
SDTSLineReader, 135
AttachToPolygons, 135
SDTSModId, 136
nRecord, 136
szModule, 136
szName, 136
szOBRP, 137
SDTSPointReader, 137
SDTSPolygonReader, 138
AssembleRings, 138
SDTSRasterBand, 139
SDTSRasterReader, 139
GetBlock, 140
GetBlockXSize, 140
GetBlockYSize, 140
GetMinMax, 140
GetRasterType, 141
GetTransform, 141
GetXSize, 141
GetYSize, 142
SDTSRawLine, 142
Dump, 143
nVertices, 143
oEndNode, 143
oLeftPoly, 143
oRightPoly, 144
oStartNode, 144
padfX, 144
padfY, 144
padfZ, 144
SDTSRawPoint, 145
dfX, 145
dfY, 146
dfZ, 146
Dump, 145
oAreaId, 146
SDTSRawPolygon, 146
AssembleRings, 147
Dump, 148
nRings, 148
nVertices, 148
padfX, 148
padfY, 148
padfZ, 149
panRingStart, 149
SDTSTransfer, 149
FindLayer, 150
GetAttr, 151
GetBounds, 151
GetCATD, 152
GetLayerCATDEntry, 152
GetLayerIndexedReader, 152
GetLayerRasterReader, 153
GetLayerType, 153
GetXREF, 154
Open, 154
SFRegion, 154
STARTS_WITH_CI
cpl_port.h, 266
STARTS_WITH
cpl_port.h, 266
STRCASECMP
cpl_port.h, 267
STRNCASECMP
cpl_port.h, 267
Save
CPLJSONDocument, 56
SaveAsString
CPLJSONDocument, 56
ScanModuleReferences
SDTSIndexedReader, 134
sdtsdataset.cpp, 349
Seek
VSISparseFileHandle, 160
VSISubFileHandle, 164
VSIVirtualHandle, 170
Set
CPLJSONObject, 68–70
SetFieldRaw
DDFRecord, 116
SetFloatSubfield

DDFRecord, 116
SetIntSubfield
DDFRecord, 117
SetNull
CPLJSONObject, 70
SetRepeatingFlag
DDFFieldDefn, 105
SetStringSubfield
DDFRecord, 117
Setup
CPLWorkerThreadPool, 96
Size
CPLJSONArray, 52
StackContext, 155
SubmitJob
CPLWorkerThreadPool, 96
SubmitJobs
CPLWorkerThreadPool, 97
szModule
SDTSModId, 136
szName
SDTSModId, 136
szOBRP
SDTSModId, 137

Tell
VSISparseFileHandle, 161
VSISubFileHandle, 165
VSIVirtualHandle, 171
tm_unz_s, 155
tm_zip_s, 155
ToArray
CPLJSONObject, 70
ToBool
CPLJSONObject, 71
ToDouble
CPLJSONObject, 71
ToInteger
CPLJSONObject, 72
ToLong
CPLJSONObject, 72
ToString
CPLJSONObject, 72
Truncate
VSIVirtualHandle, 171
TupleEnvVarOptionName, 155
Type
CPLJSONObject, 59

unz_file_info_internal_s, 155
unz_file_info_s, 155
unz_file_pos_s, 156
unz_global_info_s, 156
unz_s, 156

VALIDATE_POINTER0
cpl_error.h, 216
VALIDATE_POINTER1
cpl_error.h, 216
VSI_CALLOC_VERBOSE
cpl_vsi.h, 316
VSI_ISDIR
cpl_vsi.h, 316
VSI_ISLNK
cpl_vsi.h, 316
VSI_ISREG
cpl_vsi.h, 316
VSI_L_OFFSET_MAX
cpl_vsi.h, 316
VSI_MALLOC2_VERBOSE
cpl_vsi.h, 316
VSI_MALLOC3_VERBOSE
cpl_vsi.h, 317
VSI_MALLOC_ALIGNED_AUTO_VERBOSE
cpl_vsi.h, 317
VSI_MALLOC_VERBOSE
cpl_vsi.h, 317
VSI_REALLOC_VERBOSE
cpl_vsi.h, 317
VSI_STAT_EXISTS_FLAG
cpl_vsi.h, 317
VSI_STAT_NATURE_FLAG
cpl_vsi.h, 317
VSI_STAT_SET_ERROR_FLAG
cpl_vsi.h, 318
VSI_STAT_SIZE_FLAG
cpl_vsi.h, 318
VSI_STRDUP_VERBOSE
cpl_vsi.h, 318
VSIArchiveContent, 156
VSIArchiveEntry, 156
VSIArchiveEntryFileOffset, 156
VSIArchiveFilesystemHandler, 157
VSIArchiveReader, 157
VSICalloc
cpl_vsi.h, 320
VSICallocVerbose
cpl_vsi.h, 320
VSIDIR, 157
VSIDummyCryptFilesystemHandler, 157
VSIErrorContext, 158
VSIFCloseL
cpl_vsi.h, 320
VSIFEofL
cpl_vsi.h, 321
VSIFFlushL
cpl_vsi.h, 321
VSIFGetNativeFileDescriptorL
cpl_vsi.h, 322
VSIFGetRangeStatusL
cpl_vsi.h, 322
VSIFOpenExL
cpl_vsi.h, 323
VSIFOpenL
cpl_vsi.h, 324
VSIFPutcL
cpl_vsi.h, 325

VSIFReadMultiRangeL
cpl_vsi.h, 326
VSIFReadL
cpl_vsi.h, 325
VSIFSeekL
cpl_vsi.h, 327
VSIFTellL
cpl_vsi.h, 328
VSIFTruncateL
cpl_vsi.h, 328
VSIFWriteL
cpl_vsi.h, 329
VSIFileFromMemBuffer
cpl_vsi.h, 323
VSIFileManager, 158
VSIFilesystemHandler, 158
VSIFree
cpl_vsi.h, 326
VSIFreeAligned
cpl_vsi.h, 327
VSIGetActualURL
cpl_vsi.h, 329
VSIGetDiskFreeSpace
cpl_vsi.h, 330
VSIGetFileSystemOptions
cpl_vsi.h, 330
VSIGetFileSystemsPrefixes
cpl_vsi.h, 331
VSIGetMemFileBuffer
cpl_vsi.h, 331
VSIGetSignedURL
cpl_vsi.h, 332
VSIHasOptimizedReadMultiRange
cpl_vsi.h, 332
VSIIngestFile
cpl_vsi.h, 333
VSIInstallGZipFileHandler
cpl_vsi.h, 334
VSIInstallMemFileHandler
cpl_vsi.h, 334
VSIInstallSparseFileHandler
cpl_vsi.h, 334
VSIInstallStdinHandler
cpl_vsi.h, 335
VSIInstallStdoutHandler
cpl_vsi.h, 335
VSIInstallSubFileHandler
cpl_vsi.h, 335
VSIInstallTarFileHandler
cpl_vsi.h, 336
VSIInstallZipFileHandler
cpl_vsi.h, 336
VSIIsCaseSensitiveFS
cpl_vsi.h, 337
VSILFILE
cpl_vsi.h, 318
VSIMalloc
cpl_vsi.h, 338
VSIMalloc2
cpl_vsi.h, 338
VSIMalloc2Verbose
cpl_vsi.h, 338
VSIMalloc3
cpl_vsi.h, 338
VSIMalloc3Verbose
cpl_vsi.h, 338
VSIMallocAligned
cpl_vsi.h, 339
VSIMallocAlignedAuto
cpl_vsi.h, 339
VSIMallocAlignedAutoVerbose
cpl_vsi.h, 340
VSIMallocVerbose
cpl_vsi.h, 340
VSIMkdir
cpl_vsi.h, 340
VSIMkdirRecursive
cpl_vsi.h, 341
VSIRangeStatus
cpl_vsi.h, 319
VSIReadDir
cpl_vsi.h, 341
VSIReadDirEx
cpl_vsi.h, 342
VSIReadDirRecursive
cpl_vsi.h, 342
VSIReadDirRecursiveTask, 158
VSIRealloc
cpl_vsi.h, 343
VSIReallocVerbose
cpl_vsi.h, 343
VSIRename
cpl_vsi.h, 343
VSIRewindL
cpl_vsi.h, 344
VSIRmdir
cpl_vsi.h, 344
VSIRmdirRecursive
cpl_vsi.h, 345
VSISparseFileFilesystemHandler, 158
VSISparseFileHandle, 159
Close, 159
Eof, 159
Read, 160
Seek, 160
Tell, 161
Write, 161
VSIStatBufL
cpl_vsi.h, 318
VSIStatExL
cpl_vsi.h, 345
VSIStatL
cpl_vsi.h, 346
VSIStdoutSetRedirection
cpl_vsi.h, 346
VSIStrdup

cpl_vsi.h, 347
VSIStrdupVerbose
cpl_vsi.h, 347
VSIStrerror
cpl_vsi.h, 347
VSISubFileFilesystemHandler, 162
VSISubFileHandle, 163
Close, 163
Eof, 163
Read, 164
Seek, 164
Tell, 165
Write, 165
VSISupportsSparseFiles
cpl_vsi.h, 347
VSIUnlink
cpl_vsi.h, 348
VSIVirtualHandle, 166
Close, 167
Eof, 167
Flush, 168
GetNativeFileDescriptor, 168
GetRangeStatus, 168
Read, 169
ReadMultiRange, 170
Seek, 170
Tell, 171
Truncate, 171
Write, 172
VSIWriteFunction
cpl_vsi.h, 318
vsi_l_offset
cpl_vsi.h, 318

WaitCompletion
CPLWorkerThreadPool, 97
Write
DDFRecord, 118
VSISparseFileHandle, 161
VSISubFileHandle, 165
VSIVirtualHandle, 172

zip_fileinfo, 172
zip_internal, 173
zlib_filefunc_def_s, 173