Mapping Specification for DWG/DXF (MSD)

DWGdirect Code Samples

March, 2008

The code samples contained herein are intended as learning tools and demonstrate basic coding techniques used to implement MSD. They were created with Open Design's DWGdirect™ using Microsoft Visual Studio 2005, C++. Error handling has been omitted for clarity.

This document assumes the reader has a working knowledge of reading and writing to the DWG and or DXF format. All samples are provided "as-is" and without warranty of any kind, expressed or implied. ESRI does not assume any responsibility for their successful operation.

The Open Design Alliance (ODA) is a non-profit organization funded by dues from its members. The ODA offers OpenDWG, which does not replace the DWG file format but is instead compatible with the DWG file format. The ODA is committed to maintaining compatibility between the DWGdirect libraries and the DWG file format. The aim of the ODA is to provide its membership libraries that read and write CAD files, such as DWG and DGN, or other formats where appropriate. More information regarding ODA can be found at www.opendesign.com.

Contents

Reporting the PRJ string embedded in the DWG

```
#include "OdaCommon.h"
#include "DbEntity.h"
#include "DbDictionary.h"
#include "DbXRecord.h"
#include "DbSSet.h"


#define STL_USING_IOSTREAM
#include "OdaSTL.h"
#define  STD(a)  std:: a

void Get_WKT(OdDbDatabasePtr pDwgDb)
{

       OdDbDictionaryPtr pNamedObjectsDic =
    pDwgDb->getNamedObjectsDictionaryId().safeOpenObject();

       OdDbXrecordPtr pXRec =
     OdDbXrecord::cast(pNamedObjectsDic->getAt("ESRI_PRJ", OdDb::kForRead));

       if (!pXRec.isNull())
       {
              OdDbXrecordIteratorPtr pxrecIter = pXRec->newIterator();
              if (!pxrecIter.isNull() && !pxrecIter->done())
              {
                     OdResBufPtr rb = pxrecIter->getCurResbuf();
                     OdString textString = rb->getString();
      STD(cout) << "WKT is: " << textString << STD(endl);

              }
       }

}
```

Working with Feature Classes

```
#include "OdaCommon.h"
#include "DbEntity.h"
#include "DbDictionary.h"
#include "DbXRecord.h"
#include "DbSSet.h"

#define STL_USING_IOSTREAM
#include "OdaSTL.h"
#define  STD(a)  std:: a

/*
 * Create a feature class in the specified drawing
 * - assume feature class name = MyFeatureClass
 *          feature class type = Polyline
 *          query = all features on Layer1 or Layer2
*/
void CreateFeatureClass(OdDbDatabasePtr pDwgDb)
{
  OdString fcName("MyFeatureClass");
```

```
  //Open the named objects dictionary
  OdDbDictionaryPtr pMain =
    pDwgDb->getNamedObjectsDictionaryId().safeOpenObject(OdDb::kForWrite);

  // open (or create) the ESRI_FeatureClass dictionary
  OdDbDictionaryPtr fcCollection;
  if (pMain->has("ESRI_FEATURES"))
  {
    fcCollection = pMain->getAt("ESRI_FEATURES", OdDb::kForWrite);
  }
  else
  {
    fcCollection = OdDbDictionary::createObject();
    OdDbObjectId dicId = pMain->setAt("ESRI_FEATURES", fcCollection);
  }

  // Create the specific feature class dictionary
  OdDbDictionaryPtr fClassDict = OdDbDictionary::createObject();
  OdDbObjectId fcId = fcCollection->setAt("NewFeatureClassName", fClassDict);

  // Create a new xrecord object for the type
  OdDbXrecordPtr pXRecType = OdDbXrecord::createObject();
  OdDbObjectId xrId = fClassDict->setAt("FeatureType", pXRecType);

  // Valid values for type are
  // “Polyline”, “Point”, “Annotation”, “Point”, “MultiPatch”
  pXRecType->setFromRbChain(OdResBuf::newRb(OdResBuf::kDxfXdAsciiString, "Polyline"));

  // add an xrecord for the query based on two layer values
  OdString layerList("Layer1,Layer2");
  OdDbXrecordPtr pXRecQuery = OdDbXrecord::createObject();
  OdResBufPtr pRb = OdResBuf::newRb(8, layerList);
  fClassDict->setAt("FeatureQuery", pXRecQuery);
  pXRecQuery->setFromRbChain(pRb);


}

/*
 * Add attribute definitions to the previously defined feature class
 * - assume feature class name = MyFeatureClass
 * - add fields for:
 *    Integer named IntField with a default value of 99
 *    Long named LongField with a default value of 900000
 *    Real named RealField with a default value of 99.999
 *    Text named TextField with a default value of Sample String and a length of 64
characters
*/
void AddAttributeDefinitions(OdDbDatabasePtr pDwgDb)
{
  static OdString s_ESRIDictionary("ESRI_FEATURES");
  static OdString s_AttributeDictionary("ESRI_Attributes");
  static OdString s_FeatureClassName("MyFeatureClass");

  //Open the main dictionary
  OdDbDictionaryPtr pMain =
    pDwgDb->getNamedObjectsDictionaryId().safeOpenObject(OdDb::kForWrite);

  // open the ESRI_FeatureClass dictionary
  OdDbDictionaryPtr fcCollection =
    pMain->getAt(s_ESRIDictionary, OdDb::kForWrite);

  // open the specific feature class dictionary
```

```
  OdDbDictionaryPtr fClassDict =
    OdDbDictionary::cast(
      fcCollection->getAt(s_FeatureClassName, OdDb::kForWrite));

  // create the attribute dictionary
  OdDbDictionaryPtr fClassAttribs = OdDbDictionary::createObject();
  OdDbObjectId dicId = fClassDict->setAt(s_AttributeDictionary, fClassAttribs);

  // build a field definition for Integer
  // field name="IntField", field type = Int16, default value = 99
  OdResBufPtr pRbInt16   = OdResBuf::newRb(OdResBuf::kDxfInt16, OdInt16(99));
  OdDbXrecordPtr pXRecInt = OdDbXrecord::createObject();
  fClassAttribs->setAt("IntField", pXRecInt);
  pXRecInt->setFromRbChain(pRbInt16);

  // build a field definition for Long
  // field name="LongField", field type = Int32, default value = 900000
  OdResBufPtr pRbLongVal =
     OdResBuf::newRb(OdResBuf::kDxfInt32, OdInt32(900000));
  OdDbXrecordPtr pXRecLong = OdDbXrecord::createObject();
  fClassAttribs ->setAt("LongField", pXRecLong);
  pXRecLong->setFromRbChain(pRbLongVal);

  // build a field definition for Real
  // field name="RealField", field type = Real, default value = 99.999
  OdResBufPtr pRbRealVal = OdResBuf::newRb(OdResBuf::kDxfReal, 99.999);
  OdDbXrecordPtr pXRecReal = OdDbXrecord::createObject();
  fClassAttribs->setAt("RealField", pXRecReal);
  pXRecReal->setFromRbChain(pRbRealVal);

  // build a field definition for Text
  // field name="TextField", field type = Text, default value = "Sample String"
  // include an optional string length on the field of 64
  OdResBufPtr pRbString = OdResBuf::newRb(OdResBuf::kDxfText, "Sample String");
  pRbString->setNext(OdResBuf::newRb(OdResBuf::kDxfInt32, OdInt32(64)));
  OdDbXrecordPtr pXRecText = OdDbXrecord::createObject();
  fClassAttribs->setAt("TextField", pXRecText);
  pXRecText->setFromRbChain(pRbString);
}

/*
 * List all the feature classes in the specified drawing
 * Lists only the name and type to the std output
 */
void ListAllFeatureClasses(OdDbDatabasePtr pDwgDb)
{
  static OdString s_ESRIDictionary("ESRI_FEATURES");
  static OdString s_AttributeDictionary("ESRI_Attributes");
  static OdString s_FeatureType("FeatureType");

  //Open the named object dictionary
  OdDbDictionaryPtr pMain =
      pDwgDb->getNamedObjectsDictionaryId().safeOpenObject(OdDb::kForRead);

  // read feature classes defined in the CAD drawing
  OdDbDictionaryPtr pFFDict =
    pMain->getAt(s_ESRIDictionary, OdDb::kForRead);

  if (!pFFDict.isNull())
  {
    OdDbDictionaryIteratorPtr pIter = pFFDict->newIterator();
    for (; !pIter->done(); pIter->next())
    {
```

```
      OdDbDictionaryPtr pEntry = pIter->objectId().safeOpenObject();
      if (pEntry.isNull()) continue;

      // get fc type
      OdDbXrecordPtr pXRecType =
        OdDbXrecord::cast(pEntry->getAt(s_FeatureType, OdDb::kForRead));
      if (pXRecType.isNull()) continue;
      OdResBufPtr rb = pXRecType->rbChain();
      if (rb.isNull()) continue;
      int code = rb->restype();
      OdString fcType("Unknown");
      if (OdDxfCode::String == OdDxfCode::_getType(code))
      {
        fcType = rb->getString();
      }

      // finally, print the feature class name and type
      STD(cout) << "Name =" << pIter->name() << "; Type = " << fcType << "\n";
    }
  }
}

/*
 * Helper Function to build a selection set
 * In practice, you want to take into account the entity types as well to
 * inlcude only polyine features in polyline feature classes.
 * To handle more complex queries, you'll want to include the BagFiler
 * tools provided in DWGDirect.
 */
OdDbSelectionSetPtr BuildSelectionSet(OdDbDatabase* pDwgDb, OdResBuf* rb)
{
  // if query is empty, then +
  if (rb == 0)
    return OdDbSelectionSet::select(pDwgDb);

  // look for invalid codes
  OdResBuf* temp = rb;
  while (temp)
  {
    if (temp->restype() == -4 || temp->restype() == 65532)
    {
      // ignore any queries involving conditionals since they
      // are not supported by DWGDirect
      return OdDbSelectionSet::select(pDwgDb);
    }
    temp = temp->next();
  }

  // otherwise execute the query
  return OdDbSelectionSet::select(pDwgDb, rb);
}

/*
 * Select all the features belonging to "MyFeatureClass" in the specified drawing
 */
void SelectFeatures(OdDbDatabasePtr pDwgDb)
{
  OdDbDictionaryPtr pNamedObjectsDic =
      pDwgDb->getNamedObjectsDictionaryId().safeOpenObject(OdDb::kForRead);

  // open the ESRI_FEATURES dictionary
  OdDbDictionaryPtr pFFDict =
    pNamedObjectsDic->getAt("ESRI_FEATURES", OdDb::kForRead);
```

```
  // open the dictionary for the “MyFeatureClass” feature class (assume it is there)
  OdDbDictionaryPtr fClassDict =
    OdDbDictionary::cast(pFFDict->getAt("MyFeatureClass", OdDb::kForRead));

  // get XRecord for the query and call the helper function to build the selection set
  OdDbXrecordPtr pXRecQuery;
  pXRecQuery = OdDbXrecord::cast(fClassDict->getAt("FeatureQuery", OdDb::kForRead));
  OdDbSelectionSetPtr pSSet;
  if (!pXRecQuery.isNull())
  {
    try
    {
      OdResBufPtr pRb = pXRecQuery->rbChain();
      pSSet = BuildSelectionSet(pDwgDb, pRb);
    }
    // if there's an invalid resbuf, then build a selection set of all entities
    catch(const OdError& )
    {
      pSSet = BuildSelectionSet(pDwgDb, 0);
    }
   }
  else
  {
    pSSet = OdDbSelectionSet::select(pDwgDb);
  }
}

/*
 * Delete the "MyFeatureClass" feature class in the specified DWG
 * Note that this does not impact any entities (or attributes attached
 * to entities).  This only deletes the definition of the feature class.
 */
void DeleteFeatureClass(OdDbDatabasePtr pDwgDb)
{
  static OdString s_dictName = "ESRI_FEATURES";

  //Open the main dictionary
  OdDbDictionaryPtr pMain =
      pDwgDb->getNamedObjectsDictionaryId().safeOpenObject(OdDb::kForRead);

  // open the ESRI_FeatureClass dictionary
  OdDbDictionaryPtr fcCollection =
   pMain->getAt(OdString(s_dictName), OdDb::kForRead);

  // open the specific feature class dictionary
  OdDbDictionaryPtr fClassDict =
      OdDbDictionary::cast(
        fcCollection->getAt("MyFeatureClass", OdDb::kForWrite));

  // erase it
  fClassDict->erase();
}
```

Working with Attributes on Entities

```
#include "OdaCommon.h"
#include "DbEntity.h"
#include "DbDictionary.h"
```

```
#include "DbXRecord.h"
#include "DbSSet.h"

#define STL_USING_IOSTREAM
#include "OdaSTL.h"
#define  STD(a)  std:: a


/*
 * List all the attributes attached to the specified entity
 *
 * Important note:  attributes exist independent of feature
 * classes.  To list the attributes associated with
 * a specific feature class, read the attribute
 * definitions from the feature class, and then query the
 * ESRI_Attributes dictionary on this entity for the existance of
 * each defined attribute.  If not found, the value is the
 * default stored in the feature class definition.
 */
void ListAttributes(OdDbObjectId id)
{
  OdDbEntityPtr pEnt = id.safeOpenObject();

  // open the extension dictionary
  OdDbDictionaryPtr dict =
    pEnt->extensionDictionary().safeOpenObject(OdDb::kForRead);

  // open the ESRI_Attributes dictionary (this assumes it exists)
  OdDbDictionaryPtr attrDict = dict->getAt("ESRI_Attributes", OdDb::kForRead);

  // iterate through all the XRecords and display
  OdDbDictionaryIteratorPtr pIter = attrDict->newIterator();
  for (; !pIter->done(); pIter->next())
  {
    // the dictionary entry name is the field name
    STD(cout) << pIter->name() << " = ";

    OdDbXrecordPtr pXRec =
      OdDbXrecord::cast(pIter->objectId().safeOpenObject());
    OdResBufPtr resbuf = pXRec->rbChain();

    switch (OdDxfCode::_getType(resbuf->restype()))
    {
      case OdDxfCode::Name:
      case OdDxfCode::String:
      case OdDxfCode::LayerName:
        STD(cout) << resbuf->getString() << "\n";
            break;

      case OdDxfCode::Bool:
        STD(cout) << resbuf->getBool() << "\n";
        break;

      case OdDxfCode::Integer8:
        STD(cout) << resbuf->getInt8() << "\n";
        break;

      case OdDxfCode::Integer16:
        STD(cout) << resbuf->getInt16() << "\n";
        break;

      case OdDxfCode::Integer32:
        STD(cout) << resbuf->getInt32() << "\n";
```

```
        break;

      case OdDxfCode::Double:
        STD(cout) << resbuf->getDouble() << "\n";
        break;
    }
  }
}
```