Mapping Specification for DWG/DXF (MSD)

C# .NET 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 ObjectARX™ 2007 using Microsoft Visual Studio 2005, C# .NET. 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

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.EditorInput;


namespace C_SharpSamples
{
  class PRJSamples
  {
    static String PRJ_RecordName = "ESRI_PRJ";

    public void Get_WKT(Database db)
    {
      Transaction t = db.TransactionManager.StartTransaction();
      // get the current editor for output
      Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
      try
      {

        // open the named object dictionary
        DBDictionary dict =
           (DBDictionary)t.GetObject(db.NamedObjectsDictionaryId, OpenMode.ForRead,
           false);

        // grab the xrecord with the WKT string, if it exists
        ObjectId objID = dict.GetAt(PRJ_RecordName);
        Xrecord rec =
           (Xrecord)t.GetObject(dict.GetAt(PRJ_RecordName), OpenMode.ForRead);

        // write out the string
        ResultBuffer data = rec.Data;
        if (data != null)
        {
          ed.WriteMessage(data.ToString());
        }
        t.Commit();

      }
      catch
      {
        ed.WriteMessage("No PRJ found");
        t.Dispose();
      }
    }

  }
}
```

Working with Feature Classes

```
using System;
using System.Collections.Generic;
using System.Text;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
```

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.EditorInput;


namespace C_SharpSamples
{
  class FCSamples
  {
    static string FC_DictName = "ESRI_FEATURES";
    static string FC_Type = "FeatureType";
    static string FC_Query = "FeatureQuery";
    static string FC_Attr = "ESRI_Attributes";


            // match the feature types to the entity types used in ArcGIS.
            // note:  ArcGIS looks into individual elements of block inserts
            // this code does not.
            static TypedValue[] tvPoint =
                    {
        new TypedValue(-4, "<or"),
                        new TypedValue(0, "POINT"),
                        new TypedValue(0, "INSERT"),
                        new TypedValue(0, "SHAPE"),
                        new TypedValue(0, "HATCH"),
                        new TypedValue(0, "PROXY"),
                        new TypedValue(-4, "or>")
                    };

            static TypedValue[] tvPolyline =
                    {
        new TypedValue(-4, "<or"),
                        new TypedValue(0, "ARC"),
                        new TypedValue(0, "CIRCLE"),
                        new TypedValue(0, "ELLIPSE"),
                        new TypedValue(0, "LINE"),
                        new TypedValue(0, "MLINE"),
                        new TypedValue(0, "*POLYLINE"),
                        new TypedValue(0, "RAY"),
                        new TypedValue(0, "SPLINE"),
                        new TypedValue(0, "XLINE"),
                        new TypedValue(0, "TRACE"),
                        new TypedValue(0, "SOLID"),
                        new TypedValue(0, "FACE"),
        new TypedValue(-4,"or>")
                    };

            static TypedValue[] tvPolygon =
                    {
        new TypedValue(-4, "<or"),
                        new TypedValue(0, "CIRCLE"),
                        new TypedValue(0, "SOLID"),
                        new TypedValue(0, "ELLIPSE"),
                        new TypedValue(0, "FACE"),
                        new TypedValue (-4, "<and"),
                        new TypedValue(0, "*POLYLINE"),
                        new TypedValue(70, 1),
                        new TypedValue(-4, "and>"),
                        new TypedValue (-4, "<and"),
                        new TypedValue(0, "MLINE"),
                        new TypedValue(70, 1),
                        new TypedValue(-4, "and>"),
                        new TypedValue(-4, "or>")
                    };
```

```
            TypedValue[] tvAnnotation =
                    {
        new TypedValue(-4, "<or"),
                            new TypedValue(0,          "TEXT"),
                            new TypedValue(0,    "MTEXT"),
                            new TypedValue(0,    "ATTRIBUTE"),
                            new TypedValue(0,    "ATTDEF"),
                            new TypedValue(-4, "or>")
                    };

            TypedValue[] tvMultiPatch = tvPolygon;

    /*
     * Create a feature class in the specified drawing
     * - assume feature class name = MyFeatureClass
     *          feature class type = Polyline
     *          query = all features on Layer1 or Layer2
    */
    public void CreateFeatureClass(Database db)
    {
                    Transaction t = db.TransactionManager.StartTransaction();

      Document ThisDrawing =Application.DocumentManager.GetDocument(db);
      DocumentLock docLock = ThisDrawing.LockDocument();

      DBDictionary fc =
        OpenFeatureClassDictionary(db, "MyFeatureClass", t, OpenMode.ForWrite);

      // add the type
      // Valid values for type are
     // “Polyline”, “Point”, “Annotation”, “Point”, “MultiPatch”
      Xrecord recType = new Xrecord();
      recType.Data =
        new ResultBuffer(new TypedValue((int)DxfCode.Text, "Polyline"));
      fc.SetAt(FC_Type, recType);
      t.AddNewlyCreatedDBObject(recType, true);

      // add the query
      Xrecord recQuery =  new Xrecord();
      recQuery.Data = new ResultBuffer(new TypedValue(8, "Layer1, Layer2"));
      fc.SetAt(FC_Query, recQuery);
      t.AddNewlyCreatedDBObject(recQuery, true);

      // commit the transaction
      docLock.Dispose();
      t.Commit();

    }


    /*
     * 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
    */
    public void AddAttributeDefinitions(Database db)
    {
```

```
  Transaction t = db.TransactionManager.StartTransaction();

  Document ThisDrawing = Application.DocumentManager.GetDocument(db);
  DocumentLock docLock = ThisDrawing.LockDocument();

  // open/Create the attribute Dictionary
  DBDictionary fcAttr =
    OpenFeatureClassAttributeDictionary(db, "MyFeatureClass",
      t, OpenMode.ForWrite);

  // add an integer field with default value of 99
  Xrecord recIntField = new Xrecord();
  recIntField.Data = new ResultBuffer(new TypedValue(70, 99));
  fcAttr.SetAt("IntField", recIntField);
  t.AddNewlyCreatedDBObject(recIntField, true);

  // add a long field with default value of 900000
  Xrecord recLongField = new Xrecord();
  recLongField.Data = new ResultBuffer(new TypedValue(90, 900000));
  fcAttr.SetAt("longField", recLongField);
  t.AddNewlyCreatedDBObject(recLongField, true);

  // add a real field with default value of 99.999
  Xrecord recRealField = new Xrecord();
  recRealField.Data = new ResultBuffer(new TypedValue(40, 99.999));
  fcAttr.SetAt("realField", recRealField);
  t.AddNewlyCreatedDBObject(recRealField, true);

  // add a string field default value of "Sample String" and a length of 64
  Xrecord recStringField = new Xrecord();
  recStringField.Data =
    new ResultBuffer(new TypedValue(1, "Sample String"));
  recStringField.Data.Add(new ResultBuffer(new TypedValue(90, 64)));
  fcAttr.SetAt("TextField", recStringField);
  t.AddNewlyCreatedDBObject(recStringField, true);

  // release the lock and transaction
  docLock.Dispose();
  t.Commit();
}



/*
 * List all the feature classes in the specified drawing
 * Lists the name, type, and query to the command line
 */
public void ListFeatureClasses(Database db)
{

  // get the current editor for output
  Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;

  Transaction t = db.TransactionManager.StartTransaction();
  Document ThisDrawing = Application.DocumentManager.GetDocument(db);
  DocumentLock docLock = ThisDrawing.LockDocument();

  // open the feature data (obviously, you'd want to handle
  // the case where the feature collection doesn't exist)
  DBDictionary fcCollection =
    OpenFeatureDatasetDictionary(db, t, OpenMode.ForRead);
```

```
    foreach (DBDictionaryEntry curDict in fcCollection)
    {
      ed.WriteMessage("Name: " + curDict.Key + Environment.NewLine);
      DBDictionary thisDict =
         OpenFeatureClassDictionary(db, curDict.Key, t, OpenMode.ForRead);

      // get the feature type
      Xrecord xType =
         (Xrecord)t.GetObject(thisDict.GetAt("FeatureType"),
           OpenMode.ForRead, false);

      foreach (TypedValue tv in xType.Data.AsArray())
        ed.WriteMessage(String.Format("FeatureType = {1}",
           tv.TypeCode, tv.Value) + Environment.NewLine);

      // get the feature query
      Xrecord xQuery =
         (Xrecord)t.GetObject(thisDict.GetAt("FeatureQuery"),
           OpenMode.ForRead, false);
      ResultBuffer rbQuery = xQuery.Data;
      foreach (TypedValue tv in xQuery.Data.AsArray())
        ed.WriteMessage(String.Format("TypeCode={0}, Value={1}",
           tv.TypeCode, tv.Value) + Environment.NewLine);

    }

    // release the lock and transaction
    docLock.Dispose();
    t.Dispose();

  }


  /*
   * Select all the features in the feature class "MyFeatureClass"
   *
   */
  public void SelectFeatures(Database db)
  {

    Transaction t = db.TransactionManager.StartTransaction();
    Document ThisDrawing = Application.DocumentManager.GetDocument(db);
    DocumentLock docLock = ThisDrawing.LockDocument();
    String fcName = "MyFeatureClass";


    // get the query from the dictionary
    DBDictionary thisDict =
        OpenFeatureClassDictionary(db, fcName, t, OpenMode.ForRead);
    if (thisDict != null)
    {
      string typeString = "";
      // get the feature type
      Xrecord xType = (Xrecord)t.GetObject(thisDict.GetAt("FeatureType"),
          OpenMode.ForRead, false);
      foreach (TypedValue tv in xType.Data.AsArray())
        typeString = tv.Value.ToString();

      // get the feature query
      Xrecord xQuery = (Xrecord)t.GetObject(thisDict.GetAt("FeatureQuery"),
        OpenMode.ForRead, false);
```

```
      ResultBuffer rbFilter = new ResultBuffer(new TypedValue(-4, "<and"));

      // first select the feature type
      // Valid values for type are “Polyline”, "Polygon", “Point”,
      // “Annotation”, “Point”, “MultiPatch”
      if (typeString == "Polyline")
      {
        for (int j = 0; j < tvPolyline.Length; j++)
          rbFilter.Add(tvPolyline[j]);
      }
      else if (typeString == "Point")
      {
        for (int j = 0; j < tvPoint.Length; j++)
          rbFilter.Add(tvPoint[j]);
      }

      else if (typeString == "Polygon")
      {
        for (int j = 0; j < tvPolygon.Length; j++)
          rbFilter.Add(tvPolygon[j]);
      }

      else if (typeString == "Annotation")
      {
        for (int j = 0; j < tvAnnotation.Length; j++)
          rbFilter.Add(tvAnnotation[j]);
      }

      else if (typeString == "MultiPatch")
      {
        for (int j = 0; j < tvMultiPatch.Length; j++)
          rbFilter.Add(tvMultiPatch[j]);
      }

      TypedValue[] values = xQuery.Data.AsArray();
      for (int i = 0; i < values.Length; i++)
        rbFilter.Add(values[i]);

      rbFilter.Add(new TypedValue(-4, "and>"));

      SelectionFilter filterSS = new SelectionFilter(rbFilter.AsArray());

      Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
      PromptSelectionResult selResults = ed.SelectAll(filterSS);
      if (selResults.Status == PromptStatus.OK && selResults.Value.Count > 0)
      {
        ObjectId[] ids = selResults.Value.GetObjectIds();
        Autodesk.AutoCAD.Internal.Utils.SelectObjects(ids);
      }

    }
    t.Dispose();
    docLock.Dispose();
  }



  /*
   * Delete the "MyFeatureClass" definition
   * Note that this does not remove any attributes from entities
   * participating in the MyFeatureClass query
   */
  void DeleteFeatureClass(Database db)
```

```
{
  // begin the transaction
  Transaction t = db.TransactionManager.StartTransaction();
  Document ThisDrawing = Application.DocumentManager.GetDocument(db);
  DocumentLock docLock = ThisDrawing.LockDocument();


  // open the dictionary
  DBDictionary thisDict =
    OpenFeatureClassDictionary(db, "MyFeatureClass", t, OpenMode.ForWrite);

  // erase it
  thisDict.Erase();

  // commit the transaction
  t.Commit();
  docLock.Dispose();

}

//============================================================================
// HELPER FUNCTIONS
/*
 * Helper function to open or create the Feature Class dictionary
*/
DBDictionary
OpenFeatureDatasetDictionary(Database db, Transaction t, OpenMode mode)
{
  DBDictionary fcCollection;
  DBDictionary NOD =
    (DBDictionary)t.GetObject(db.NamedObjectsDictionaryId, mode, false);

  // if it exists, just open it and return
  try
  {
    fcCollection =
       (DBDictionary)t.GetObject(NOD.GetAt(FC_DictName), mode, false);
    return fcCollection;
  }

  // otherwise, if we are to open for write, add it
  catch
  {
    if (mode != OpenMode.ForWrite)
      return null;
    fcCollection = new DBDictionary();
    NOD.SetAt(FC_DictName, fcCollection);
    t.AddNewlyCreatedDBObject(fcCollection, true);
    fcCollection =
       (DBDictionary)t.GetObject(NOD.GetAt(FC_DictName), mode, false);
    return fcCollection;
  }
}

/*
 * Helper function to open or create a feature class dictionary
*/
DBDictionary
OpenFeatureClassDictionary(Database db, string name,
  Transaction t, OpenMode mode)
{
  DBDictionary ffCollection =
    OpenFeatureDatasetDictionary(db, t, mode);
```

```
      try
      {
        DBDictionary fc =
           (DBDictionary)t.GetObject(ffCollection.GetAt(name), mode, false);
        return fc;
      }
      catch
      {
        if (mode == OpenMode.ForWrite)
        {
          DBDictionary fc = new DBDictionary();
          ffCollection.SetAt(name, fc);
          t.AddNewlyCreatedDBObject(fc, true);
          fc =
           (DBDictionary)t.GetObject(ffCollection.GetAt(name), mode, false);
          return fc;
        }
        return null;
      }

    }

    /*
     * Helper function to open or create a attribute definition
     * dictionary in a specified feature class
    */
    DBDictionary
    OpenFeatureClassAttributeDictionary(Database db, string name,
    Transaction t, OpenMode mode)
    {
      DBDictionary featureClass =
        OpenFeatureClassDictionary(db, name, t, mode);

      try
      {
        DBDictionary fcAttr =
                          (DBDictionary)t.GetObject(featureClass.GetAt(name), mode,
false);
        return fcAttr;
      }
      catch
      {
        if (mode == OpenMode.ForWrite)
        {
          DBDictionary fcAttr = new DBDictionary();
          featureClass.SetAt(FC_Attr, fcAttr);
          t.AddNewlyCreatedDBObject(fcAttr, true);
          fcAttr =
              (DBDictionary)t.GetObject(featureClass.GetAt(FC_Attr),
              mode, false);
          return fcAttr;
        }
        return null;
      }
    }

  }
}
```

Working with Attributes on Entities

```
using System;
```

```
using System.Collections;
using System.Collections.Generic;
using System.Text;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.EditorInput;

namespace C_SharpSamples
{
  class EntityAttributeSamples
  {

    /*
     * To attach attributes to a specific entity, follow the
     * example that attaches attribute definitions to a
     * feature class.  The only difference is that you use the
     * ESRI_Attributes dictionary in the entity's extension
     * dictionary instead of the feature class dictionary.
     */


    /*
     * 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(Database db, ObjectId entid)
    {

      Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
      using (Transaction t = db.TransactionManager.StartTransaction())
      {

        // open the entity and get the extension dictionary

        Entity entity = (Entity)t.GetObject(entid, OpenMode.ForRead, true);
        ObjectId extDictId = entity.ExtensionDictionary;
        if ((extDictId.IsNull == false) && (extDictId.IsErased == false))
        {

          // open the extension dictionary
          DBObject tmpObj = t.GetObject(extDictId, OpenMode.ForRead);
          DBDictionary dbDict = tmpObj as DBDictionary;
          if (dbDict != null)
          {

            // if the extension dictionary contains attributes, open the
            // attributes dictionary
            if (dbDict.Contains("ESRI_Attributes"))
            {
              ObjectId attrDictId = dbDict.GetAt("ESRI_Attributes");
              if ((attrDictId.IsNull == false) &&
                   (attrDictId.IsErased == false))
              {
                DBObject anObj = t.GetObject(attrDictId, OpenMode.ForRead);
                DBDictionary attrDict = anObj as DBDictionary;
```

```
                if (attrDict != null)
                {

                  // iterate through the attributes in the dictionary
                  // and write out their values to the command line
                  foreach (DBDictionaryEntry curEntry in attrDict)
                  {
                    ObjectId curEntryId = curEntry.Value;
                    DBObject curEntryObj =
                     t.GetObject(curEntryId, OpenMode.ForRead);
                    Xrecord curX = curEntryObj as Xrecord;
                    if (curX != null)
                    {
                      if (curX.Data != null)
                      {
                        IEnumerator iter = curX.Data.GetEnumerator();
                        iter.MoveNext();
                        TypedValue tmpVal = (TypedValue)iter.Current;
                        if (tmpVal != null)
                        {
                          ed.WriteMessage("\n");
                          ed.WriteMessage(curEntry.m_key);
                          ed.WriteMessage("(");
                          ed.WriteMessage(tmpVal.TypeCode.ToString());
                          ed.WriteMessage(") = ");
                          ed.WriteMessage(tmpVal.Value.ToString());
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
        t.Dispose();
      }
    }
  }
}
```