Autodesk University 2012

Using .NET in the Land of NOD

James E. Johnson – Synergis Software

CP2146 This class is about working with the Named Object Dictionary (NOD) and other dictionaries that can exist in an AutoCAD® software drawing database using .NET applications. We will look at getting and using dictionary keys such as get/add an UnderlayDefinition key for adding DGN, DWF™ and PDF file underlay references to the active document. The Named Object Dictionary can contain other dictionaries like the ACAD_MLINESTYLE and ACAD_MATERIAL dictionaries, of which we will have code samples to show you how to use the data in those and other drawing database dictionaries. You will leave this class with a good understanding and with code samples for working with AutoCAD drawing database dictionaries and know what is typically stored in the Named Object Dictionary.

Learning Objectives

At the end of this class, you will be able to:

- Describe what the Named Object Dictionary contains and how to work with those dictionary keys
- Add and get underlay definitions in the Named Object Dictionary
- Work with drawing database dictionaries in a .NET application
- Use XRecords in dictionaries to store application information within a drawing

About the Speaker

James has worked with CAD products for more than 25 years, involved in many positions from being a CAD drafter to writing automation applications. In his current position, he is doing CAD integration for Adept document management system. In previous positions, he has used Autodesk® RealDWG® to write custom automation to create AutoCAD® drawings of industrial kitchen equipment, and has worked at Autodesk resellers in software development groups doing custom applications for Inventor® and AutoCAD®. He has taught AutoCAD® and VBA classes while working for resellers, and was a CAD instructor at two different community colleges.

Email: james.e.johnson@me.com

AutoCAD® dictionaries...

In an AutoCAD® drawing database a dictionary is a container object which can contain any AutoCAD® object or an XRecord.

Dictionaries are stored either in the database under the named object dictionary or as an extension dictionary of a table record or graphical entity. The named object dictionary is the master table for all of the dictionaries associated with a database. Unlike symbol tables, new dictionaries can be created and added to the named object dictionary.

Describe what the Named Object Dictionary contains and how to work with those dictionary keys...

The Named Object Dictionary contains all dictionaries other than extension dictionaries in the drawing database. Dictionary objects cannot contain drawing entities.

The Named Object Dictionary is where data is placed for the entire drawing to access. This dictionary is always present in all drawing databases. AutoCAD® uses this dictionary for items such as Material styles and MLine styles.

An Extension dictionary is a dictionary that is attached to an entity to store Objects/XRecords which are typically entity specific.

Default dictionary keys in the Named Object Dictionary:

ACAD_CIP_PREVIOUS_PRODUCT_INFO
ACAD_COLOR
ACAD_DETAILVIEWSTYLE
ACAD_GROUP
ACAD_LAYOUT
ACAD_MATERIAL
ACAD_MLEADERSTYLE
ACAD_MLINESTYLE
ACAD_PLOTSETTINGS
ACAD_PLOTSTYLENAME
ACAD_SCALELIST
ACAD_SECTIONVIEWSTYLE
ACAD_TABLESTYLE
ACAD_VISUALSTYLE
AcDbVariableDictionary

In code the Dictionary entries are handled with the `DBDictionaryEntry` class by enumerating through the contents of a dictionary...

The dictionary is accessed in .NET as an AutoCAD DBDictionary object. To get an instance from the active drawing database:

```
DBDictionary nod = tr.GetObject(HostApplicationServices
                                .WorkingDatabase
                                .NamedObjectsDictionaryId,
                                 OpenMode.ForRead) as DBDictionary;
```

Sample code to retrieve Dictionary entries:

```
[CommandMethod("GetNOD")]
public void getNOD()
{
    Editor ed = Application.DocumentManager
                            .MdiActiveDocument
                            .Editor;

    using (Transaction tr = HostApplicationServices
                                    .WorkingDatabase
                                    .TransactionManager
                                    .StartTransaction())
    {
        DBDictionary nod = tr.GetObject(HostApplicationServices
                                            .WorkingDatabase
                                            .NamedObjectsDictionaryId,
                                             OpenMode.ForRead) as DBDictionary;

        foreach (DBDictionaryEntry de in nod)
        {
            ed.WriteMessage("Key: " + de.Key.ToString() + "\n");
        }

    }
}
```

In code the Dictionary entries are handled with the `DBDictionaryEntry` class by enumerating through the contents of a dictionary...

The Named Object dictionary can also be accessed at the AutoCAD Command line with AutoLISP:

```
Command: (setq mainDict (namedobjdict))
Command: (entget mainDict)
```

Returns:

```
((-1 . <Entity name: 7ffff7038c0>) (0 . "DICTIONARY") (330 . <Entity name: 0>) (5 . "C")
(100 . "AcDbDictionary") (280 . 0) (281 . 1) (3 . "ACAD_CIP_PREVIOUS_PRODUCT_INFO")
(350 . <Entity name: 7ffff705ad0>) (3 . "ACAD_COLOR") (350 . <Entity name: 7ffff703c30>)
(3 . "ACAD_DETAILVIEWSTYLE") (350 . <Entity name: 7ffff705b30>) (3 . "ACAD_GROUP") (350 .
<Entity name: 7ffff7038d0>) (3 . "ACAD_LAYOUT") (350 . <Entity name: 7ffff7039a0>) (3 .
"ACAD_MATERIAL") (350 . <Entity name: 7ffff703c20>) (3 . "ACAD_MLEADERSTYLE") (350 .
<Entity name: 7ffff7050f0>) (3 . "ACAD_MLINESTYLE") (350 . <Entity name: 7ffff703970>)
(3 . "ACAD_PLOTSETTINGS") (350 . <Entity name: 7ffff703990>) (3 . "ACAD_PLOTSTYLENAME")
(350 . <Entity name: 7ffff7038e0>) (3 . "ACAD_SCALELIST") (350 . <Entity name:
7ffff705060>) (3 . "ACAD_SECTIONVIEWSTYLE") (350 . <Entity name: 7ffff705b00>) (3 .
"ACAD_TABLESTYLE") (350 . <Entity name: 7ffff703ce0>) (3 . "ACAD_VISUALSTYLE") (350 .
<Entity name: 7ffff703e10>) (3 . "AcDbVariableDictionary") (350 . <Entity name:
7ffff703b60>))
```

Sample code to add a Dictionary entry:

```
[CommandMethod("addToNOD")]
public void addToNOD()
{
    Editor ed = Application.DocumentManager
                           .MdiActiveDocument
                           .Editor;
    using (Transaction tr = HostApplicationServices
                                   .WorkingDatabase
                                   .TransactionManager
                                   .StartTransaction())
    {
        DBDictionary nod = tr.GetObject(HostApplicationServices
                                           .WorkingDatabase
                                           .NamedObjectsDictionaryId,
                                           OpenMode.ForWrite) as DBDictionary;
        string newDictName = "LandOfNod";
        DBDictionary newDict = new DBDictionary();
        if (!nod.Contains(newDictName))
        {
            nod.SetAt(newDictName, newDict);
            tr.AddNewlyCreatedDBObject(newDict, true);
        }
        tr.Commit();
    }
}
```

Creates a new DBDictionary and adds it to the database...

Sample code to remove a Dictionary entry:

```
[CommandMethod("removeNOD")]
public void removeNOD()
{
    Editor ed = Application.DocumentManager
                           .MdiActiveDocument
                           .Editor;
    using (Transaction tr = HostApplicationServices
                                   .WorkingDatabase
                                   .TransactionManager
                                   .StartTransaction())
    {
        DBDictionary nod = tr.GetObject(HostApplicationServices
                                           .WorkingDatabase
                                           .NamedObjectsDictionaryId,
                                           OpenMode.ForWrite) as DBDictionary;
        string newDictName = "LandOfNod";
        DBDictionary newDict = new DBDictionary();
        if (nod.Contains(newDictName))
        {
            nod.Remove(newDictName);
        }
        tr.Commit();
    }
}
```

Removes the DBDictionary and all of its contents from the database...

The following example illustrates getting an ObjectID of a dictionary entry in the NOD then iterates through that dictionary. In this example the “ACAD_LAYOUT” name is used to get the proper dictionary entry then the value (ObjectID) is used to open the dictionary. This could have also been accomplished using the layout dictionary ID from the database:

```
        HostApplicationServices.WorkingDatabase.LayoutDictionaryId
```

```
Other Default dictionaries can be acquired in the same way:
```

```
        HostApplicationServices.WorkingDatabase.MaterialDictionaryId
        HostApplicationServices.WorkingDatabase.MLeaderStyleDictionaryId
        HostApplicationServices.WorkingDatabase.MLStyleDictionaryId
        HostApplicationServices.WorkingDatabase.ColorDictionaryId
```

Sample code to get entries in a sub Dictionary:

```
[CommandMethod("GetNODLayout")]
public void getNODLayout()
{
    Editor ed = Application.DocumentManager
                           .MdiActiveDocument
                           .Editor;

    using (Transaction tr = HostApplicationServices
                                   .WorkingDatabase
                                   .TransactionManager
                                   .StartTransaction())
    {

        DBDictionary nod = tr.GetObject(HostApplicationServices
                                           .WorkingDatabase
                                           .NamedObjectsDictionaryId,
                                           OpenMode.ForRead) as DBDictionary;

        if (nod.Contains("ACAD_LAYOUT"))
        {
            foreach (DBDictionaryEntry de in nod)
            {
                if (de.Key == "ACAD_LAYOUT")
                {
                    DBDictionary layoutDB = (DBDictionary)tr.GetObject(
                                               de.Value,
                                               OpenMode.ForRead);

                    foreach (DBDictionaryEntry LOde in layoutDB)
                    {
                        ed.WriteMessage("Key: " + LOde.Key.ToString() + "\n");
                    }
                }
            }
        }
    }
}
```

Add and get underlay definitions in the Named Object Dictionary...

In AutoCAD® Underlays are similar to raster images,but their content is snappable. The UnderlayDefinition is an abstract class that handles the linkage to external underlay content. An UnderlayDefinition object is referenced by zero or more UnderlayReferences. The UnderlayReference class is an abstract class that represents underlays in the drawing.

The UnderlayReference object is responsible for the placement of the content within the drawing, while the UnderlayDefinition object handles the linkage to the underlay content.

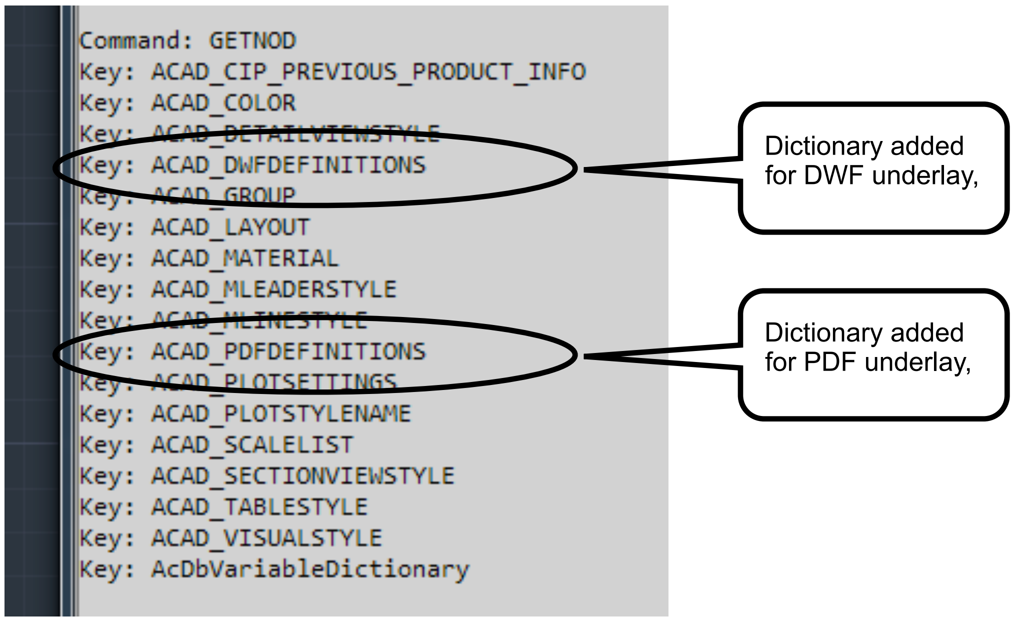
There are three primary classes that Inherit from the UnderlayDefinition class, Instances of UnderlayDefinition-derived concrete classes are inserted into a dictionary within the named object dictionary...

- Autodesk.AutoCAD.DatabaseServices.UnderlayDefinition
  - Autodesk.AutoCAD.DatabaseServices.DgnDefinition
  - Autodesk.AutoCAD.DatabaseServices.DwfDefinition
  - Autodesk.AutoCAD.DatabaseServices.PdfDefinition

There are three primary classes that Inherit from the UnderlayReference class, Instances of UnderlayReference-derived concrete classes are inserted into a block table record...

- Autodesk.AutoCAD.DatabaseServices.UnderlayReference
  - Autodesk.AutoCAD.DatabaseServices.DgnReference
  - Autodesk.AutoCAD.DatabaseServices.DwfReference
  - Autodesk.AutoCAD.DatabaseServices.PdfReference

When an UnderlayDefinition is added to the drawing the associated dictionaries are added to the named object dictionary...



```
Command: GETNOD
Key: ACAD_CIP_PREVIOUS_PRODUCT_INFO
Key: ACAD_COLOR
Key: ACAD_DETAILVIEWSTYLE
Key: ACAD_DWFDEFINITIONS
Key: ACAD_GROUP
Key: ACAD_LAYOUT
Key: ACAD_MATERIAL
Key: ACAD_MLEADERSTYLE
Key: ACAD_MLINESTYLE
Key: ACAD_PDFDEFINITIONS
Key: ACAD_PLOTSETTINGS
Key: ACAD_PLOTSTYLENAME
Key: ACAD_SCALELIST
Key: ACAD_SECTIONVIEWSTYLE
Key: ACAD_TABLESTYLE
Key: ACAD_VISUALSTYLE
Key: AcDbVariableDictionary
```

Dictionary added for DWF underlay,
Dictionary added for PDF underlay,

Sample code to insert a PDF underlay:

```
[CommandMethod("pdfInsert")]
static public void pdfInsert()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;

    using (Transaction tr =  doc.TransactionManager.StartTransaction())
    {
        DBDictionary nod = (DBDictionary)tr
                                    .GetObject(db.NamedObjectsDictionaryId,
                                                        OpenMode.ForWrite);
        string defPDFDictKey = UnderlayDefinition
                                    .GetDictionaryKey(typeof(PdfDefinition));

        if (!nod.Contains(defPDFDictKey))
        {
            using (DBDictionary dict = new DBDictionary())
            {
                nod.SetAt(defPDFDictKey, dict);
                tr.AddNewlyCreatedDBObject(dict, true);
            }
        }
        ObjectId defObjID;
        DBDictionary pdfDict = (DBDictionary)tr
                                            .GetObject(nod.GetAt(defPDFDictKey),
                                                            OpenMode.ForWrite);

        using (PdfDefinition pdfDef = new PdfDefinition())
        {
            pdfDef.SourceFileName = @"C:\AU2012\CP2146\LandOfNOD.pdf";
            defObjID = pdfDict.SetAt("LANDOFNOD_PDF", pdfDef);
            tr.AddNewlyCreatedDBObject(pdfDef, true);
        }

        BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId, OpenMode.ForRead);
        BlockTableRecord btr = (BlockTableRecord)tr
                                              .GetObject(bt[BlockTableRecord.ModelSpace],
                                                    OpenMode.ForWrite);

        using (PdfReference pdf = new PdfReference())
        {
            pdf.DefinitionId = defObjID;
            btr.AppendEntity(pdf);
            tr.AddNewlyCreatedDBObject(pdf, true);
        }

        tr.Commit();
    }
}
```

Adds the definition to the dictionary...

Path is preset for this example for the file to load...

Adds a reference to the Instance in the BlockTable...

Sample code to insert a DWF underlay:

```
[CommandMethod("dwfInsert")]
static public void dwfInsert()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;

    using (Transaction tr = doc.TransactionManager.StartTransaction())
    {
        DBDictionary nod = (DBDictionary)tr
                                        .GetObject(db.NamedObjectsDictionaryId,
                                            OpenMode.ForWrite);
        string defDWFDictKey = UnderlayDefinition
                                        .GetDictionaryKey(typeof(DwfDefinition));

        if (!nod.Contains(defDWFDictKey))
        {
            using (DBDictionary dict = new DBDictionary())
            {
                nod.SetAt(defDWFDictKey, dict);
                tr.AddNewlyCreatedDBObject(dict, true);
            }
        }

        ObjectId defObjID;
        DBDictionary dwfDict = (DBDictionary)tr
                                            .GetObject(nod.GetAt(defDWFDictKey),
                                                OpenMode.ForWrite);

        using (DwfDefinition dwfDef = new DwfDefinition())
        {
            dwfDef.SourceFileName = @"C:\AU2012\CP2146\LandOfNOD.dwf";
            defObjID = dwfDict.SetAt("LANDOFNOD_DWF", dwfDef);
            tr.AddNewlyCreatedDBObject(dwfDef, true);
        }

        BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId, OpenMode.ForRead);
        BlockTableRecord btr = (BlockTableRecord)tr
                                            .GetObject(bt[BlockTableRecord.ModelSpace],
                                                OpenMode.ForWrite);

        using (DwfReference dwf = new DwfReference())
        {
            dwf.DefinitionId = defObjID;
            btr.AppendEntity(dwf);
            tr.AddNewlyCreatedDBObject(dwf, true);
        }

        tr.Commit();
    }
}
```

Sample code to search for PDF underlays:

```
[CommandMethod("pdfSearch")]
static public void pdfSearch()
{
    Editor ed = Application.DocumentManager
                .MdiActiveDocument
                .Editor;

    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;

    using (Transaction tr = doc.TransactionManager.StartTransaction())
    {
        DBDictionary nod = (DBDictionary)tr
                                    .GetObject(db.NamedObjectsDictionaryId,
                                                        OpenMode.ForWrite);
        string defPDFDictKey = UnderlayDefinition
                                    .GetDictionaryKey(typeof(PdfDefinition));

        if (nod.Contains(defPDFDictKey))
        {
            foreach (DBDictionaryEntry de in nod)
            {
                try
                {
                    if (de.Key == defPDFDictKey)
                    {
                        DBDictionary pdfDict = (DBDictionary)de.Value
                                                            .GetObject(OpenMode.ForRead);
                        foreach (DBDictionaryEntry pdfDE in pdfDict)
                        {
                            PdfDefinition pdf = (PdfDefinition)pdfDE.Value
                                                            .GetObject(OpenMode.ForRead);

                            string pdfPath = pdf.SourceFileName;
                            if ((pdf.ActiveFileName != null) &&
                                    (pdf.ActiveFileName != string.Empty))
                            {
                                pdfPath = pdf.ActiveFileName;
                            }

                            ed.WriteMessage("PDF File: " + pdfPath);
                        }
                    }
                }
                catch
                {
                }
            }

        }

    }
}
```

Steps through dictionary looking for PdfDefinition...

Sample code to search for DWF underlays:

```
[CommandMethod("dwfSearch")]
static public void dwfSearch()
{
    Editor ed = Application.DocumentManager
                .MdiActiveDocument
                .Editor;

    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;

    using (Transaction tr = doc.TransactionManager.StartTransaction())
    {
        DBDictionary nod = (DBDictionary)tr
                            .GetObject(db.NamedObjectsDictionaryId,
                                                    OpenMode.ForWrite);
        string defDWFDictKey = UnderlayDefinition
                            .GetDictionaryKey(typeof(DwfDefinition));

        if (nod.Contains(defDWFDictKey))
        {
            foreach (DBDictionaryEntry de in nod)
            {
                try
                {
                    if (de.Key == defDWFDictKey)
                    {
                        DBDictionary dwfDict = (DBDictionary)de.Value
                                                .GetObject(OpenMode.ForRead);
                        foreach (DBDictionaryEntry dwfDE in dwfDict)
                        {
                            DwfDefinition dwf = (DwfDefinition)dwfDE.Value
                                                .GetObject(OpenMode.ForRead);

                            string dwfPath = dwf.SourceFileName;
                            if ((dwf.ActiveFileName != null) &&
                                (dwf.ActiveFileName != string.Empty))
                            {
                                dwfPath = dwf.ActiveFileName;
                            }

                            ed.WriteMessage("PDF File: " + dwfPath);
                        }
                    }
                }
                catch
                {
                }
            }

        }

    }
}
```

Work with drawing database dictionaries in a .NET application...

In an AutoCAD® drawing database dictionaries are accessed with the DBDictionary class which is a database-resident object dictionary. A DBDictionary maintains a map between text strings and database objects.

An instance of this class represents a single object...

```
DBDictionary nod = (DBDictionary)tr.GetObject(db.NamedObjectsDictionaryId, OpenMode.ForWrite);
```

Dictionary Entries in an DBDictionary must be unique.

When objects are placed in a dictionary, the dictionary becomes the object's owner.

Dictionary names honor the rules for symbol names.

1. Names may be as long as you need them to be (longer than 32 characters, as defined by previous versions of AutoCAD).
2. Names may contain additional characters, such as the space character (' '), the apostrophe ('''), and so on.
3. Names are treated case-insensitively.
4. Names can not contain the following illegal characters:

    vertical bar ('|')

    asterisk ('*'), except as noted above

    backslash ('')

    colon (':')

    semicolon (';')

    angle brackets ('>', '<')

    question mark ('?')

    double quote ('"')

    comma (',')

    equal sign ('=')

    grave accent ('`')

Objects added to a dictionary must have NULL handles and must not have a presence in the database.

When a dictionary is erased, all the objects within it are erased.

When a dictionary is unerased, all of its contents are unerased.

Adds a dictionary to the Named object dictionary:

```
using (DBDictionary dict = new DBDictionary())
{
    nod.SetAt(defPDFDictKey, dict);
    tr.AddNewlyCreatedDBObject(dict, true);
}
```

Removes a dictionary to the Named object dictionary:

```
nod.Remove(newDictName);
```

Use XRecords in dictionaries to store application information within a drawing...

XRecords provide storage for information in AutoCAD entities and dictionaries. The XRecord class is a data storage class to allow programs a means to store data. Each XRecord object is capable of storing up to 2GB. By establishing an object's extension dictionary as the XRecord's owner, it's possible to associate large amounts of data with that object.

An XRecords structure for data input and output is done with a linked list of Resultbuffer structures, which is a list of TypedValue items. TheTypedValue object pairs a DXF Code with its associated data.

Typical Resultbuffer code sample:

```
ResultBuffer resbuf = new ResultBuffer(
    new TypedValue((int)DxfCode.Text, "HELLO"),
    new TypedValue((int)DxfCode.Int16, 256),
    new TypedValue((int)DxfCode.Real, 25.4));
```

Autodesk.AutoCAD.DatabaseServices.DxfCode Enumeration values:

Alpha = 440, Angle = 50, ArbitraryHandle = 320, AttributePrompt = 3, AttributeTag = 2, BinaryChunk = 310, BlockName = 2, Bool = 290, CircleSides = 0x48, CLShapeName = 4, CLShapeText = 9, Color = 0x3e, ColorName = 430, ColorRgb = 420, Comment = 0x3e7, ControlString = 0x66, DashLength = 0x31, Description = 3, DimBlk1 = 6, DimBlk2 = 7, DimensionAlternativePrefixSuffix = 4, DimensionBlock = 5, DimPostString = 3, DimStyleName = 3, DimVarHandle = 0x69, Elevation = 0x26, EmbeddedObjectStart = 0x65, End = -1, ExtendedDataAsciiString = 0x3e8, ExtendedDataBinaryChunk = 0x3ec, ExtendedDataControlString = 0x3ea, ExtendedDataDist = 0x411, ExtendedDataHandle = 0x3ed, ExtendedDataInteger16 = 0x42e, ExtendedDataInteger32 = 0x42f, ExtendedDataLayerName = 0x3eb, ExtendedDataReal = 0x410, ExtendedDataRegAppName = 0x3e9, ExtendedDataScale = 0x412, ExtendedDataWorldXCoordinate = 0x3f3, ExtendedDataWorldXDir = 0x3f5, ExtendedDataWorldXDisp = 0x3f4, ExtendedDataWorldYCoordinate = 0x3fd, ExtendedDataWorldYDir = 0x3ff, ExtendedDataWorldYDisp = 0x3fe, ExtendedDataWorldZCoordinate = 0x407, ExtendedDataWorldZDir = 0x409, ExtendedDataWorldZDisp = 0x408, ExtendedDataXCoordinate = 0x3f2, ExtendedDataYCoordinate = 0x3fc, ExtendedDataZCoordinate = 0x406, ExtendedInt16 = 400, FirstEntityId = -2, GradientAngle = 460, GradientColCount = 0x1c5, GradientColVal = 0x1cf, GradientName = 470, GradientObjType = 450, GradientPatType = 0x1c3, GradientShift = 0x1cd, GradientTintType = 0x1c4, GradientTintVal = 0x1ce, Handle = 5, HardOwnershipId = 360, HardPointerId = 340, HasSubentities = 0x42, HeaderId = -2, **Int16 = 70**, Int32 = 90, Int64 = 160, Int8 = 280, Invalid = -9999, LayerLinetype = 0x3d, LayerName = 8, LayoutName = 410, LinetypeAlign = 0x48, LinetypeElement = 0x31, LinetypeName = 6, LinetypePdc = 0x49, LinetypeProse = 3, LinetypeScale = 0x30, LineWeight = 370, MlineOffset = 0x31, MlineStyleName = 2, NormalX = 210, NormalY = 220, NormalZ = 230, Operator = -4, PixelScale = 0x2f, PlotStyleNameId = 390, PlotStyleNameType = 380, PReactors = -5, **Real = 40**, RegAppFlags = 0x47, RenderMode = 0x119, ShapeName = 2, ShapeScale = 0x2e, ShapeXOffset = 0x2c, ShapeYOffset = 0x2d, SoftOwnershipId = 350, SoftPointerId = 330, Start = 0, Subclass = 100, SymbolTableName = 2, SymbolTableRecordComments = 4, SymbolTableRecordName = 2, **Text = 1**, TextBigFontFile = 4, TextFontFile = 3, TextStyleName = 7, Thickness = 0x27, TxtSize = 40, TxtStyleFlags = 0x47, TxtStylePSize = 0x2a, TxtStyleXScale = 0x29, UcsOrg = 110, UcsOrientationX = 0x6f, UcsOrientationY = 0x70, ViewBackClip = 0x2c, ViewBrightness = 0x8d, ViewContrast = 0x8e, ViewFrontClip = 0x2b, ViewHeight = 0x2d, ViewLensLength = 0x2a, ViewMode = 0x47, ViewportActive = 0x44, ViewportAspect = 0x29, ViewportGrid = 0x4c, ViewportHeight = 40, ViewportIcon = 0x4a, ViewportNumber = 0x45, ViewportSnap = 0x4b, ViewportSnapAngle = 50, ViewportSnapPair = 0x4e, ViewportSnapStyle = 0x4d, ViewportTwist = 0x33, ViewportVisibility = 0x43, ViewportZoom = 0x49, ViewWidth = 0x29, Visibility = 60, XCoordinate = 10, XDataStart = -3, XDictionary = -6, XInt16 = 170, XReal = 140, XRefPath = 1, XTextString = 300, XXInt16 = 270, YCoordinate = 20, ZCoordinate = 30

Code sample to write an XRecord to an extension dictionary:

```
public void writeXRecord(ObjectId id, string key, ResultBuffer resbuf)
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    using (Transaction tr = db.TransactionManager.StartTransaction())
    {
        Entity ent = tr.GetObject(id, OpenMode.ForWrite) as Entity;
        if (ent != null)
        {
            if (ent.ExtensionDictionary == ObjectId.Null)
                ent.CreateExtensionDictionary();
            }
            DBDictionary xDict = (DBDictionary)
                                                .GetObject(ent.ExtensionDictionary,
                                                    OpenMode.ForWrite);
            Xrecord xRec = new Xrecord();
            xRec.Data = resbuf;
            xDict.SetAt(key, xRec);
            tr.AddNewlyCreatedDBObject(xRec, true);
        }
        tr.Commit();
    }
}
```

Looks for an Extension dictionary, which is where XRecords are stored...

Code sample to read an XRecord from an extension dictionary:

```
public ResultBuffer readXRecord(ObjectId id, string key)
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    ResultBuffer result = new ResultBuffer();
    using (Transaction tr = db.TransactionManager.StartTransaction())
    {
        Xrecord xRec = new Xrecord();
        Entity ent = tr.GetObject(id, OpenMode.ForRead, false) as Entity;
        if (ent != null)
        {
            try
            {
                DBDictionary xDict = (DBDictionary)tr
                                                .GetObject(ent.ExtensionDictionary,
                                                        OpenMode.ForRead, false);
                xRec = (Xrecord)tr.GetObject(xDict.GetAt(key),
                                                OpenMode.ForRead, false);
                return xRec.Data;
            }
            catch
            {
                return null;
            }
        }
        else return null;
    }
}
```

Gets the XRecord object and returns its Data property which is a ResultBuffer...

Code sample of command to write an XRecord to an extension dictionary:

```
[CommandMethod("addXRecordToEntity")]
public void addXRecord()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    Editor ed = doc.Editor;
    PromptEntityOptions pEntOp = new PromptEntityOptions("\nSelect an entity: ");
    pEntOp.AllowNone = false;
    PromptEntityResult per = ed.GetEntity(pEntOp);
    if (per.Status == PromptStatus.OK)
    {
        ObjectId id = per.ObjectId;
        ResultBuffer resbuf = new ResultBuffer(
            new TypedValue((int)DxfCode.Text, "HELLO"),
            new TypedValue((int)DxfCode.Int16, 256),
            new TypedValue((int)DxfCode.Real, 25.4));
        writeXRecord(id, "hello", resbuf);
    }
}
```

Creates a new ResultBuffer and assigns some values...

Code sample of command to read an XRecord to an extension dictionary:

```
[CommandMethod("getXRecordFromEntity")]
public void getXRecord()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    Editor ed = doc.Editor;
    PromptEntityOptions pEntOp = new PromptEntityOptions("\nSelect an entity: ");
    pEntOp.AllowNone = false;
    PromptEntityResult per = ed.GetEntity(pEntOp);
    if (per.Status == PromptStatus.OK)
    {
        ObjectId id = per.ObjectId;
        ResultBuffer resbuf = readXRecord(id, "hello");
        if (resbuf == null)
            ed.WriteMessage("\nNo XRecord found...");
        else
        {
            TypedValue[] result = resbuf.AsArray();
            ed.WriteMessage("{0} {1} {2}", result[0], result[1], result[2]);
        }
    }
}
```

The Dynamic Language Runtime (DLR) is used to implement dynamic languages like Python and Ruby on the .NET Framework.

In Visual Studio 2010 C# a new type of 'dynamic' was introduced. This is a static type that bypasses static type checking and functions like it has a type of object. A dynamic type at compile time assumes support for any operation, whether its value comes from a COM API, a dynamic language, an HTML Document Object Model (DOM), reflection, or somewhere in the program. Errors are caught at run time for invalid code.

The equivalent to the dynamic keyword is object in VB.NET but with ‘Option Strict Off’, with ‘Option Strict On’ there is no equivalent.

Code sample to read an XRecord using Dynamic Type:

```
static public void dynamicXRecordRead(string dictName, string xRecName)
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    Editor ed = doc.Editor;

    dynamic nod = db.NamedObjectsDictionaryId;

    try
    {
        var xr = nod[dictName][xRecName];
        ed.WriteMessage("\n - " + xr.Data.ToString());
    }
    catch (Autodesk.AutoCAD.Runtime.Exception acadEx)
    {
        if (acadEx.ErrorStatus == ErrorStatus.KeyNotFound)
            ed.WriteMessage("\nDictionary or record does not exist");
    }
}
```

Code sample to write an XRecord using Dynamic Type:

```
static public void dynamicXRecordWrite(string dictName, string xRecName, string value)
{
    Database db = HostApplicationServices.WorkingDatabase;

    dynamic nod = db.NamedObjectsDictionaryId;

    dynamic newdict = nod.SetAt(dictName, new DBDictionary());
    dynamic record = newdict.SetAt(xRecName, new Xrecord());

    ResultBuffer rb = new ResultBuffer();

    rb.Add(new TypedValue((int)DxfCode.Text, value));

    record.Data = rb;
}
```

Code sample of command to write an XRecord using Dynamic Type:

```
[CommandMethod("addXRecordDynamic")]
public void addXRecordDynamic()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    Editor ed = doc.Editor;
    PromptEntityOptions pEntOp = new PromptEntityOptions("\nSelect an entity: ");
    pEntOp.AllowNone = false;
    PromptEntityResult per = ed.GetEntity(pEntOp);
    if (per.Status == PromptStatus.OK)
    {
        dynamic ent = per.ObjectId;
        dynamicXRecordEntWrite(ent, "hello", "Hello");
    }
}
```

Code sample of command to read an XRecord using Dynamic Type:

```
[CommandMethod("getXRecordDynamic")]
public void getXRecordDynamic()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Database db = doc.Database;
    Editor ed = doc.Editor;
    PromptEntityOptions pEntOp = new PromptEntityOptions("\nSelect an entity: ");
    pEntOp.AllowNone = false;
    PromptEntityResult per = ed.GetEntity(pEntOp);
    if (per.Status == PromptStatus.OK)
    {
        dynamic ent = per.ObjectId;
        dynamic xrec = dynamicXRecordEntRead(ent, "hello");
        if (xrec == null)
            ed.WriteMessage("\nNo XRecord found...");
        else
        {
            ed.WriteMessage("\n - " + xrec.Data.ToString());
        }
    }
}
```