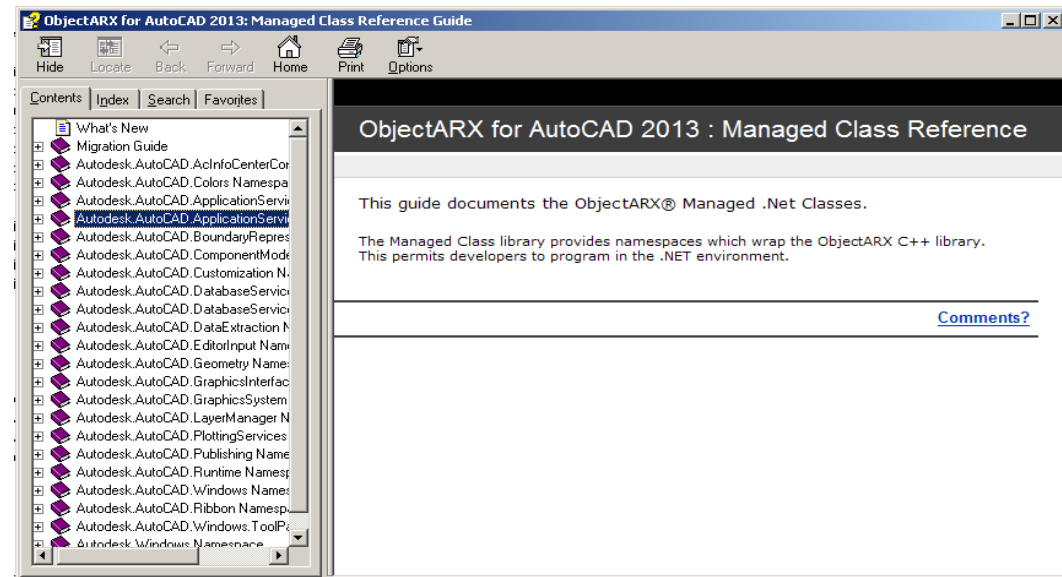




AutoCAD .NET API

- Wie ist die Funktionalität der AutoCAD .NET API dokumentiert?
- ObjectARX SDK :
 - SDK Samples
 - ObjectARX Developer's Guide
 - Managed Reference Guide
 - Arxmgd.chm
- ADN website
 - DevNotes
 - DevHelp Online
- Visual Studio Class Browser




Quelle: Autodesk

- Das ObjArx-Toolkit ist für AutoCAD 2013 nur unter Microsoft Visual Studio 2010 (SP1) installierbar,
- Problem beim Erstellen von non-managed-dll (kein .NET) ist der Einsatz des richtigen Compilers. Bei einer Entwicklung unter 2012 muß der 2010er Compiler eingestellt werden. Die Integration in Studio 2012 ist prinzipiell möglich, aber aufwendig
- Für AutoCAD 2014 wird unter Visual Studio 2012 entwickelt

- Ein .NET-Addin für AutoCAD ist immer eine Klassenbibliothek. Das manuelle Aufsetzen eines Projektes beginnt also mit der Erstellung eines Klassenbibliotheks-Projektes
- In diesem Projekt werden folgende Verweise gesetzt:
 - acdbmgd.dll
 - Database Services und DWG Dateibearbeitungsfunktionalität (wie ObjectDBX)
 - acmgd.dll
 - AutoCAD Applikations-spezifische Dinge
 - accoremgd.dll
 - AutoCAD Kernlogik
- Die verwiesenen Dlls befinden sich im AutoCAD-Installationsordner
 - ACHTUNG: (lokale Kopie in den Verweiseigenschaften auf false setzen, sonst ist das Debuggen nicht möglich)
- Visual Studio wird nur als 32bit Version ausgeliefert. Auf 64bit-Systemen läßt sich AutoCAD nur als 64bit Version installieren, so daß nur debuggt werden kann. Änderungen am Quelltext während des Debuggens sind nicht möglich.

Wie funktioniert ein AutoCAD AddIn?



Project VB.NET

```
1 Imports Autodesk.AutoCAD.Runtime
2 Imports Autodesk.AutoCAD.ApplicationServices
3 Imports Autodesk.AutoCAD.Interop
4
5 Public Class Commands
6     Implements IExtensionApplication
7
8     Private WithEvents docs As DocumentCollection
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

Quelltext in Visual
Basic/C#/C++ .NET

```
Private Sub docs_DocumentActivated(ByVal sender As Object, ByVal e As Autodesk.AutoCAD.ApplicationServices.DocumentEventArgs)
    ThisDrawing = e.Document
End Sub

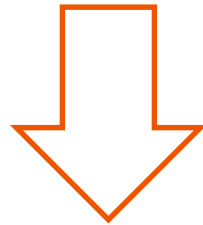
Private Sub ThisDrawing_BeginDocClose(ByVal Cancel As Boolean) Handles ThisDrawing.BeginDocClose
    ThisDrawing = Nothing
End Sub

Private Sub ThisDrawing_EndSave(ByVal FileName As String) Handles ThisDrawing.EndSave
    MsgBox("EndSave called")
End Sub
End Class
```

Verweise auf AutoCAD DLLs.



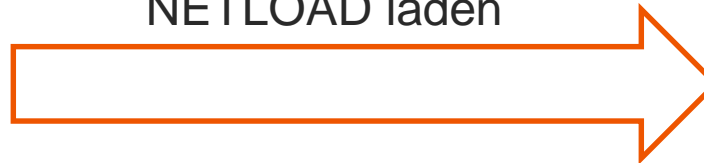
Kompilieren



Assembly-Datei
(.dll)



In AutoCAD mit dem Befehl
NETLOAD laden



Quelle: Autodesk

- Folgende Namespaces können benutzt werden:
 - Autodesk.AutoCAD.ApplicationServices
 - Zugriff auf die AutoCAD Applikation
 - Autodesk.AutoCAD.EditorInput
 - Zugriff auf Benutzerinteraktions-Klassen
 - Autodesk.AutoCAD.Runtime
 - Registration von Befehlen
 - Autodesk.AutoCAD.DatabaseServices
 - Zugriff auf die AutoCAD Database und die Entities

- Über ein Attribut kann ein neuer Befehl für AutoCAD bereitgestellt werden:

```
public class Class1
{
    [CommandMethod("HelloWorld")]
    public void HelloWorld()
    {
    }
    [CommandMethod("OpenDXF", CommandFlags.Session)]
    ...
}
```

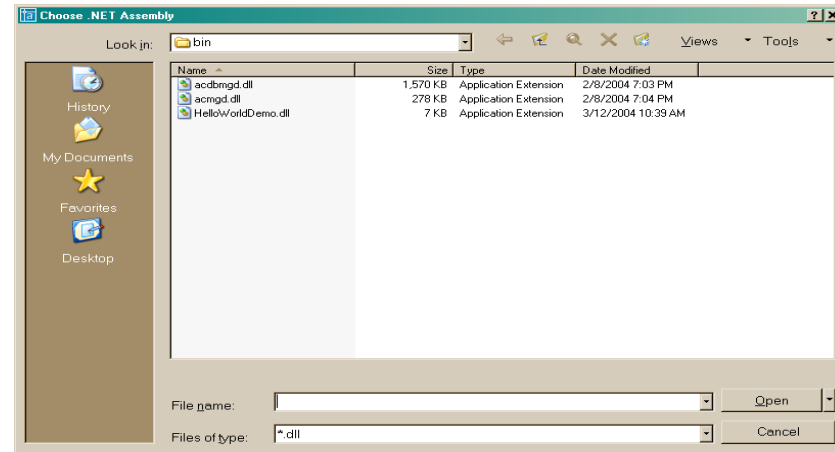
- Das Attribut sorgt für eine Signatur der Funktion, die damit über interne Mechanismen für den Host-Prozess (AutoCAD) verfügbar gemacht wird
- CommandMethod akzeptiert verschiedene CommandMethodAttribute im Konstruktor, mit denen z.B. Internationale Befehlsnamen, Flags für die Befehlsausführung und anderes eingestellt werden können

- Um einen Text in der AutoCAD-Befehlszeile auszugeben, kann folgender Quelltext verwendet werden:

```
public class Class1
{
    [CommandMethod("HelloWorld")]
    public Function HelloWorld()
    {
        Editor ed =
            Application.DocumentManager.MdiActiveDocument.Editor;
        ed.WriteMessage("Hello World");
    }
}
```


Loading .NET assembly

- NETLOAD Befehl
- AUTOLOADER
 - Startup
 - On command invocation
- Demand Load (Registry)
 - Startup
 - Bei Befehlsaufruf
 - Bei Aufruf aus anderer Applikation
 - Bei custom Entities



```
[HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\AutoCAD\R19.0\ACAD-B001:409\Applications\AcLayer]
```

```
"DESCRIPTION"="AutoCAD Layer Manager"
```

```
"LOADER"="C:\\Program Files\\AutoCAD 2013\\aclayer.dll"
```

```
"LOADCTRLS"=dword:0000000e
```

```
"MANAGED"=dword:00000001
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\AutoCAD\R19.0\ACAD-B001:409\Applications\AcLayer\Commands]
```

```
"LAYER"="LAYER"
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\AutoCAD\R19.0\ACAD-B001:409\Applications\AcLayer\Groups]
```

```
"ACLAYER_CMDS"="ACLAYER_CMDS"
```



Die Registry-Schlüssel sollten durch eine Installations-Routine gesetzt werden

Quelle: Autodesk

- AutoCAD lädt Apps, die in %appdata%\Autodesk\ApplicationPlugins gelistet sind
- Jede App hat eine "PackageContents.xml"

```
<?xml version="1.0" encoding="utf-8"?>

<ApplicationPackage SchemaVersion="1.0" AutodeskProduct="AutoCAD" ProductType="Application" Name="MyApp"
    AppVersion="1.0" Description="MyTestApp"
    Author="Autodesk" Icon="./Contents/Help/Resource/TDIcon.jpg"
    OnlineDocumentation="http://www.autodesk.com"
    HelpFile="./Contents/Help/MyApp.chm" ProductCode="{DF51A41E-DC4F-4ad8-8F8A-B6CB7130840F}">

    <RuntimeRequirements OS="Win32|Win64" Platform="AutoCAD*" SeriesMin="R19.0" SeriesMax="R19.0" />

    <CompanyDetails Name="Autodesk" Phone=" " Url="http://www.autodesk.com" Email="Support@autodesk.com" />

    <Components>
        <RuntimeRequirements SupportPath="./Contents/Support" OS="Win32|Win64" SeriesMin="R18.0" />

        <ComponentEntry AppName="MyApp" ModuleName="./Contents/Windows/MyApp.fas" AppDescription="MyTestApp"
            LoadOnAutoCADStartup="True" LoadOnCommandInvocation="True" />

        <ComponentEntry AppName="MyApp" ModuleName="./Contents/Support/MyApp.cuix" />
    </Components>

</ApplicationPackage>
```

DemandLoad mit Registry-Eintrag

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

SOFTWARE

Autodesk

AutoCAD

R19.0

ACAD-B001:409

Applications

YourAppName

Wurzelschlüssel der Benutzer-spez.-Registry-Einträge
Einstellungen für den Rechner – i.d.R. nur mit
Admin-Rechten beschreibbar

R17.0: 2007
.1: 2008
.2: 2009
R18.0: 2010
.1: 2011
.2: 2012
R19.0: 2013

Wenn unter LOCAL_MACHINE
eingetragen wurde, dann steht
die App allen Anwendern zur
Verfügung, sonst nur dem
entsprechenden Benutzer!

X000: Civil3D
X001: AutoCAD

409: Englisch
407: Deutsch
040A: Spanisch ...

Die Schlüssel AutoCAD
und R19.0 haben curVer-
Einträge, mit denen die
zuletzt aktive AutoCAD-Version
ermittelt werden kann.

"DESCRIPTION"="Custom App Name"
"LOADER"="C:\\folder\\appName.dll"
"LOADCTRLS"=dword:0000000e
"MANAGED"=dword:00000001

Quelle: Autodesk

- 0x01
Lädt Application wenn ein Custom Object erkannt wird
- 0x02
Lädt die Applikation wenn AutoCAD startet
- 0x04
Lädt die Applikation bei Aufruf eines enthaltenen Befehls
- 0x08
Lädt die Applikation bei Benutzeranfrage
- 0x10
Lade die Applikation nicht
- 0x20
Lade die Application transparent (während ein Befehl aktiv sein kann)

- Die PromptXXXOptions – Klasse für Optionen zu Benutzereingaben
 - XXX ist der Wertetyp, der vom Benutzer abgefragt werden soll (Punkt, Auswahlsatz, Abstand, ...)
 - Mit Message- und Keyword-Attributen kann die Eingabeaufforderung eingestellt werden, unter anderem mit Darstellung möglicher Schlüsselwörter
 - Mit den AllowYYY-Attributen kann die Gültigkeit der Eingaben eingeschränkt werden
- Die Editor - GetXXX Funktionen zur Benutzereingabe
 - Beispiele für diese Funktionen: GetAngle, GetString, GetDistance, GetCorner etc.
 - Mit den übergebenen PromptXXXOptions kann das Verhalten von GetXXX konfiguriert werden
- Das Ergebnis der Eingabe wird in einem PromptResult-Objekt oder einer davon abgeleiteten Klasse zurückgegeben
 - Beispiele: PromptDoubleResult, PromptIntegerResult etc.

Benutzereingabe am Beispiel eines Schlüsselwortes

- Das Beispiel zeigt die Definition der Benutzeroptionen für die Eingabe eines Schlüsselwortes.
- Schlüsselwörter können auch als Optionen anderer Benutzereingaben verwendet werden (z.B. bei der Auswahl eines Punktes)

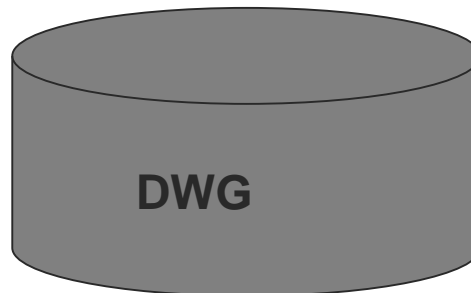
```
PromptKeywordOptions pStrOpts = new PromptKeywordOptions("\nLöschen in  
[K]ontur, [A]lle, [M]anuelle Selektion");  
pStrOpts.Keywords.Add("K", "K", "Kontur");  
pStrOpts.Keywords.Add("A", "A", "Alle");  
pStrOpts.Keywords.Add("M", "M", "Manuelle Selektion");  
pStrOpts.Keywords.Default = "K";  
PromptResult pStrRes = ...Editor.GetKeywords(pStrOpts);  
if (pStrRes.Status == PromptStatus.Cancel) return;  
if ((pStrRes.Status == PromptStatus.OK) || (pStrRes.Status ==  
PromptStatus.Keyword))  
{  
    if (pStrRes.StringResult == "K")  
        ...
```

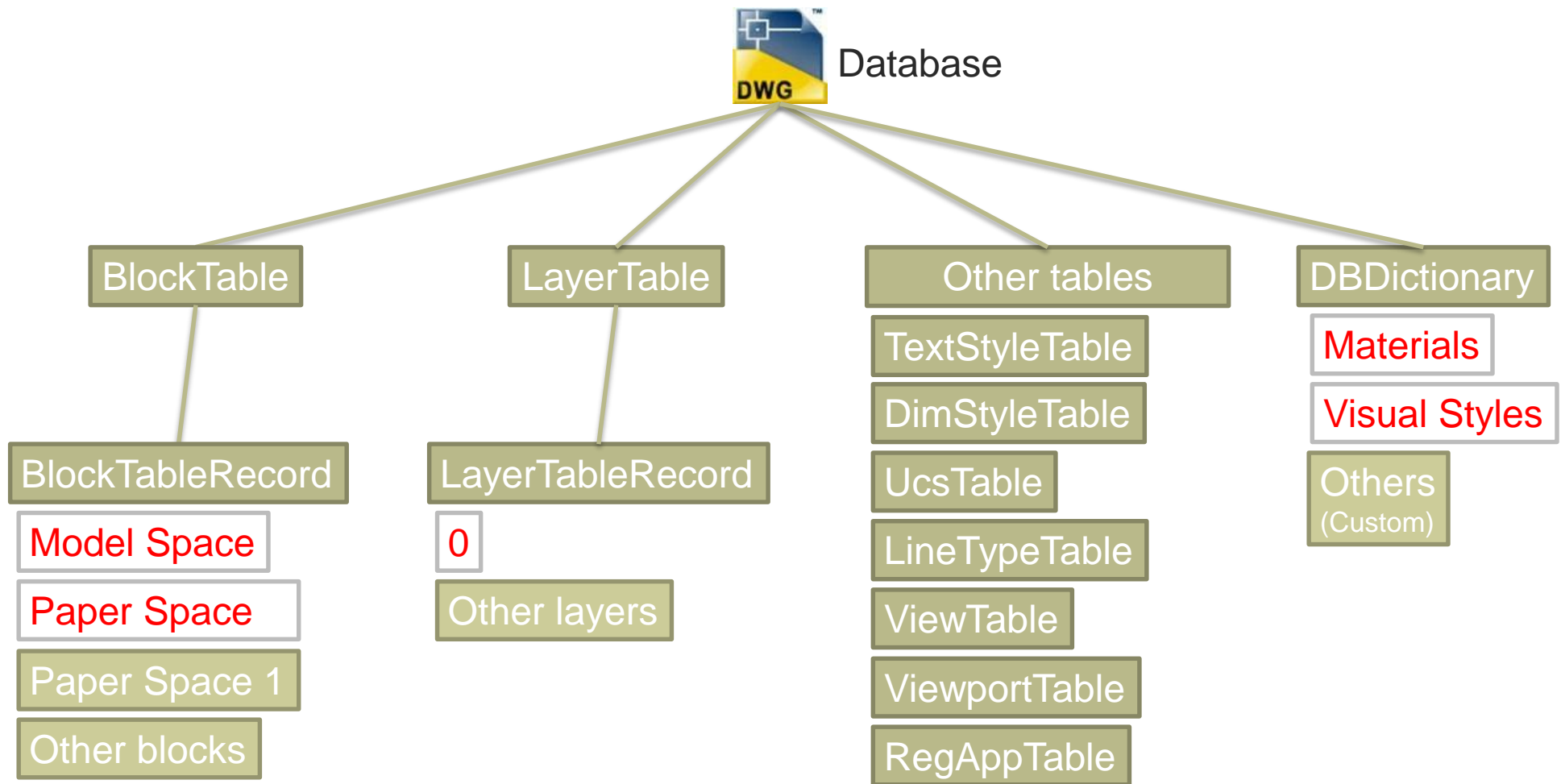
Quelle: Autodesk

- Setzen der Eingabeaufforderung
 - Eingabe der Anzahl der Seiten:
- Setzen von Schlüsselwörtern
 - Eingabe der Anzahl der Seiten [Dreieck/Quadrat/Pentagon]:
- Setzen von Defaultwerten
 - Eingabe der Anzahl der Seiten [Dreieck/Quadrat/Pentagon] <3>:
- Setzen erlaubter Werte
 - Eingabe der Anzahl der Seiten [Dreieck/Quadrat/Pentagon] <3>: -5
 - Der Wert darf nicht negativ oder 0 sein.

- Typen:
 - PromptPointOptions
 - PromptStringOptions
 - PromptDoubleOptions
 - PromptAngleOptions
 - PromptCornerOptions
 - PromptDistanceOptions
 - PromptEntityOptions
 - PromptIntegerOptions
 - PromptKeywordOptions
 - PromptNestedEntityOptions
 - PromptNestedEntityOptions
 - PromptSelectionOptions
 - Usw.

- In-Memory Darstellung der DWG-Datei
 - Die Objekte einer DWG werden hierarchisch in der Datenbank abgebildet – das entspricht einer Datenbank-Struktur
 - Alle Objekte haben Identitäten:
 - ObjectId: die eindeutige ID über alle aktuell geöffneten Dokument, ändert sich nach jedem Öffnen der Datei, nicht in der DWG gespeichert
 - Handle: die eindeutige ID innerhalb eines Dokumentes, kann in mehreren Dokumenten vorkommen, ist aber in der Datei gespeichert und ändert sich nicht, ein einmal in einer DWG vergebenes Handle wird nicht mehr vergeben
 - Auf Objekte wird in Transaktionen zugegriffen
 - Eine Transaktion definiert die Grenzen einer Datenbank-Operation
 - Objekte müssen vor ihrer Benutzung geöffnet werden
 - Objekte können auf andere Objekte verweisen – wie eine Linie auf einen Layer





Quelle: Autodesk

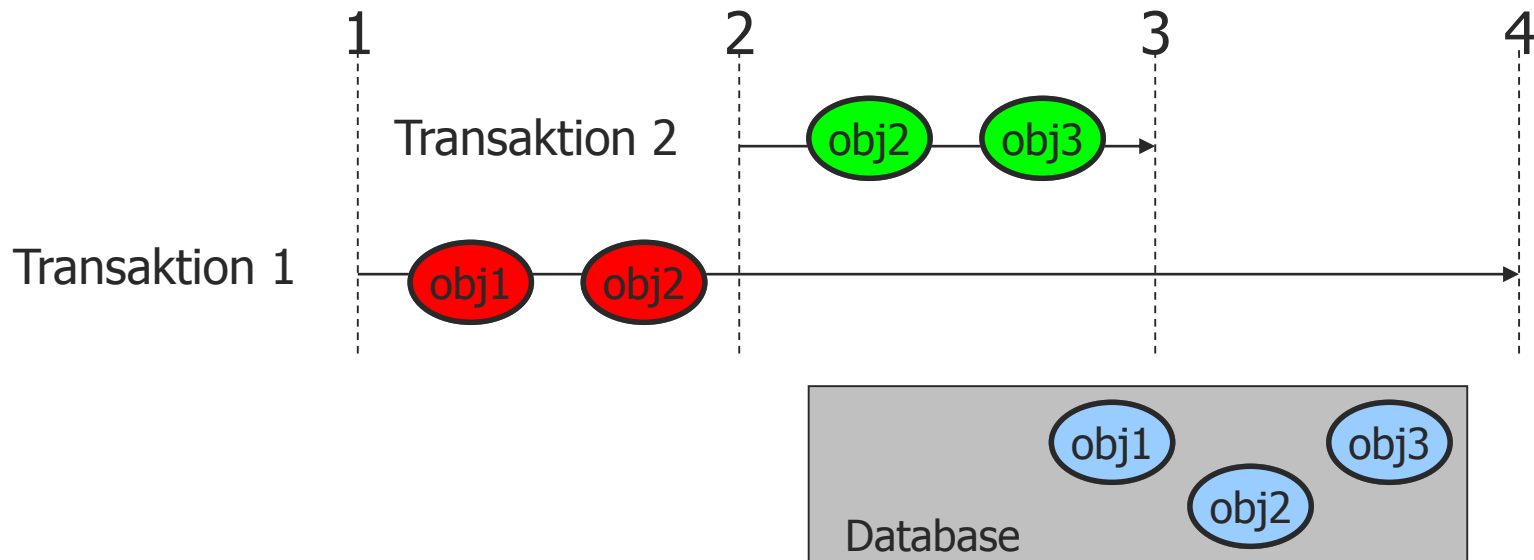
- Symbol Tables
 - Wie Layer Table, Linetype Table, Textstyle Table usw.
 - Die Tables sind Container zur Speicherung von Symbol Table Records
 - Z.B. LayerTableRecord, LinetypeTableRecord etc
 - Alle Symbol Tables haben gemeinsame Container-Methoden wie
 - Add – zum Hinzufügen eines Records
 - Item – zum Abrufen oder Suchen eines Eintrags
 - Has – zum Prüfen, ob ein Eintrag existiert
 - Sind enumerierbar (foreach-fähig)
 - Jeder Symbol Table kann nur Records eines speziellen Typs verwalten
 - So wie ein LayerTable nur LayerTableRecords beinhalten kann
- Mit Tools wie SnoopDb oder ArxDbg kann die Database untersucht werden

- Ein Database-Objekt kann
 - Im Speicher erzeugt oder
 - Anhand einer geöffneten DWG ermittelt werden
 - `HostApplicationServices.WorkingDatabase()` ist die Database des aktuell aktiven Dokuments

- Transaktionen
 - Setzen die Grenzen der Datenbank-Operationen
 - Verfügen über ein Exception-Handling und erlauben die gesamten, innerhalb der Transaktion vorgenommenen Änderungen rückgängig zu machen, wenn ein Fehler auftreten sollte
 - Nutzen einen Undo-Filer. Eine ausgeführte Transaktion ist über ein einziges Undo rückgängig zu machen
 - können
 - committed (Alle database-Operationen werden ausgeführt),
 - rolled back (alle database Operationen werden abgebrochen) und
 - verschachtelt werden

- Alle Database-Objekte haben eine ObjectId (und ein Handle):
 - ObjectId: die eindeutige ID über alle aktuell geöffneten Dokument, ändert sich nach jedem Öffnen der Datei, ist dabei nicht in der DWG gespeichert, gilt also nur für eine AutoCAD-Sitzung, bei Neuladen der Datei erhalten die Objekte andere ObjectIds
 - Handle: die eindeutige ID innerhalb eines Dokumentes, kann in mehreren Dokumenten vorkommen, ist aber in der Datei gespeichert
 - Eine ObjectId wird automatisch generiert sobald ein Objekt der Database hinzugefügt wird
 - Elemente, die nicht in der Database sind, haben keine ObjectId
 - ObjectIds können gemerkt und für ein späteres Laden verwendet werden
 - Mit einer ObjectIdCollection können mehrere ObjectIds gemerkt werden

Verschachtelte Transaktionen



- 1. Client startet Transaktion1 und greift auf Obj1 & Obj2 zu
- 2. Client startet Transaktion2 und greift auf Obj2 & Obj3 zu
- 3. Client committed Transaktion2
 - Änderungen in Transaktion2 werden committed
- 4a. Client committed Transaktion1
 - Trans1 Änderungen werden committed
- 4b. Client bricht Transaktion1 ab
 - Transaktion1 (und Transaktion2) Änderungen werden zurück genommen

Transaktionen

```
Public Function MyFunc()  
{  
    //Aktuelle database in AutoCAD holen  
    Database db = HostApplicationServices.WorkingDatabase();  
    //Starten der Transaktion über den database transaction manager  
    Transaction trans = db.TransactionManager.StartTransaction();  
    try  
    {  
        //Alle database-Operationen im try-Block!  
        //Blocktable der Database holen  
        BlockTable bt = trans.GetObject(db.BlockTableId, OpenMode.ForWrite);  
        //hier geht es nun mit dem Blocktable weiter..  
        //Transaktion durchführen  
        trans.Commit();  
    }  
    catch  
    {  
        trans.Abort();  
    }  
    finally  
    {  
        //Alles ok. Dispose aufrufen, um Transaktionsobjekt zu zerstören  
        trans.Dispose();  
    }  
}
```

■ Standard DB Zugriff mit Transaktionen

```
public void MyFunc()
{
    //Aktuelle Database in AutoCAD
    Database db = HostApplicationServices.WorkingDatabase();
    //Start der Transaktion
    Transaction trans = db.TransactionManager.StartTransaction();
    //Durch das using-Konstrukt ist kein try-catch-finally und Dispose
    //erforderlich
    using(trans)
    {
        //Alle database-Operationen im try-Block!
        //Blocktable der Database holen
        BlockTable bt = trans.GetObject(db.BlockTableId, OpenMode.ForWrite);
        //hier geht es nun mit dem Blocktable weiter...

        //Transaktion durchführen
        trans.commit()
    }
}
```


- Mit der Methode `GetObject` des Transaktions-Objekts wird ein Database-Objekt geöffnet
 - Der erste Parameter ist die `ObjectId`
 - Der zweite Parameter der `OpenMode`
 - `ForRead` – Lesezugriff
 - `ForWrite` – Schreibzugriff
 - `ForNotify` – Wenn das Objekt Nachrichten schickt
- Da `GetObject` ein object liefert, muß i.d.R. gecastet werden. Dies kann wie folgt realisiert werden:

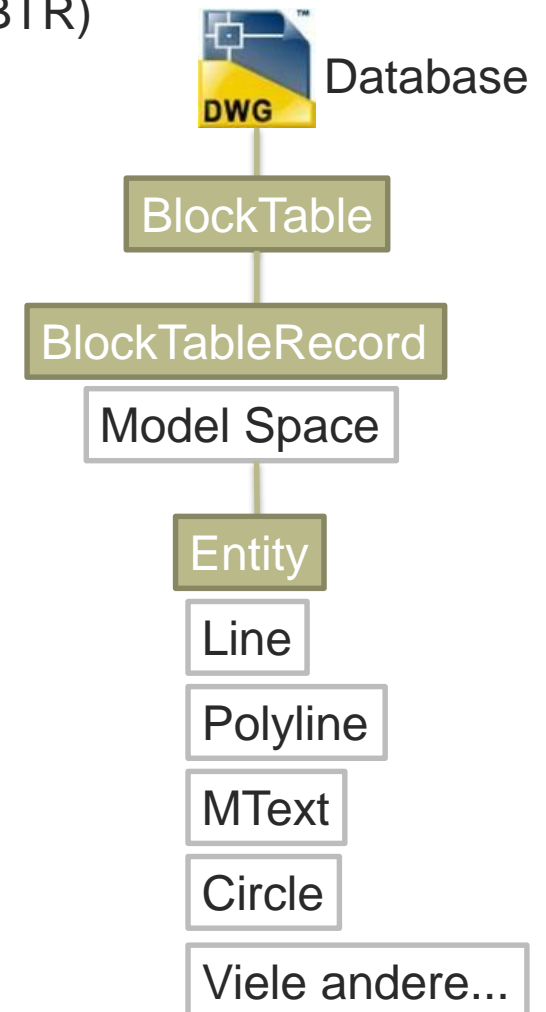
```
BlockTable bt = (BlockTable) trans.GetObject(db.BlockTableId,  
OpenMode.ForWrite); //Casting-Exception möglich -> try-catch
```

```
BlockTable bt = trans.GetObject(db.BlockTableId, OpenMode.ForWrite) as  
BlockTable; //Bei Casting-Fehler ist bt null, keine Exception
```

- Neu erzeugte Objekte werden in Containern (Owner) verwaltet – welches ist der richtige Container für das neue Objekt?
 - Alle Objekte haben genau einen Owner
 - Ein LayerTableRecord kann nur in den LayerTable eingefügt werden, eine neue Linie nur in einen BlockTableRecord
- Mit der Add-Methode werden Symbol Table Records zu ihren Symbol Tables hinzugefügt
- Mit AppendXXX können andere Objekte ihren Ownern zugewiesen werden
 - AppendEntity zum Hinzufügen zum BlockTableRecord
- Wenn ein Object einen Owner bekommt, muß die Transaktion informiert werden

```
newBtr.AppendEntity(circle); `Neuen Kreis in BTR aufnehmen
trans.AddNewlyCreatedDBObject(circle, True); //Transaktion inf.
```

- Im BlockTable werden die BlockTableRecords (BTR) verwaltet
- Der Model Space (Modellbereich) ist ein BlockTableRecord (BTR)
 - Dieses Konzept wird auch für den Layout Bereich und andere internen bzw, benutzerdefinierte Blöcke verwendet
 - Ein eingefügter BlockTableRecord (häufig nur als Block bezeichnet) wird als BlockReference eingefügt (häufig auch nur als Block bezeichnet.
 - Eine als BlockReference eingefügte Datei (externe Referenz) stellt den Modellbereich der referenzierten Datei dar
 - BlockTableRecords ohne Referenzen können mit dem Befehl Bereinigen gelöscht werden (gilt auch für alle andere nicht-referenzierten Elemente in SymbolTables
- Jeder BTR enthält Entities
- Für jeden Geometrietyp gibt es einen entsprechenden Entity-Typ
- Der BTR ist enumerierbar – (foreach möglich)



Quelle: Autodesk

Hinzufügen einer Linien zum Modellbereich

```
Point3d mm_p1 = new Point3d(10.20,0);
Point3d mm_p2 = new Point3d(100,30,0);
Line mm_lin = new Line(mm_p1, mm_p2);
//Aktuelle Database in AutoCAD
Database mm_db = HostApplicationServices.WorkingDatabase();
//Start der Transaktion
Transaction mm_trans = mm_db.TransactionManager.StartTransaction();
using (mm_trans)
{
    BlockTable mm_bt = (BlockTable)
        mm_trans.GetObject(mm_db.BlockTableId,
            OpenMode.ForRead);
    BlockTableRecord mm_btr = (BlockTableRecord)
        mm_trans.GetObject(mm_bt[BlockTableRecord.ModelSpace],
            OpenMode.ForWrite);
    mm_btr.AppendEntity(mm_lin);
    mm_trans.AddNewlyCreatedDBObject(mm_lin, true);
}
```

- Achtung: Verwaltete Objekte ummanteln ein unmanaged C++-Objekt! Der Kern von AutoCAD ist in unmanaged C++ geschrieben.
- Wenn Objekte mit new erzeugt werden – muss Dispose() gerufen werden?
 - Nein – die Garbage Collection ruft Dispose() automatisch
 - Dabei gilt:
 - Wenn das Objekt nicht in der Database ist wird das unmanaged Objekt automatisch mit gelöscht
 - Wenn das Objekt in der Database ist erfolgt ein Close() des nicht-verwalteten Objektes
- Wenn ein Objekt in einer Transaktion geöffnet wird, wird das Objekt beim Dispose or Commit der Transaktion automatisch geschlossen
- Manuelles Dispose nur in wenigen Fällen, z.B. der Transaktion (ohne Using)
 - Ein manuelles Dispose von nicht mehr verwendeten Objekten zur richtigen Zeit entlastet die Garbage Collection und kann die Laufzeit des Programms deutlich verringern

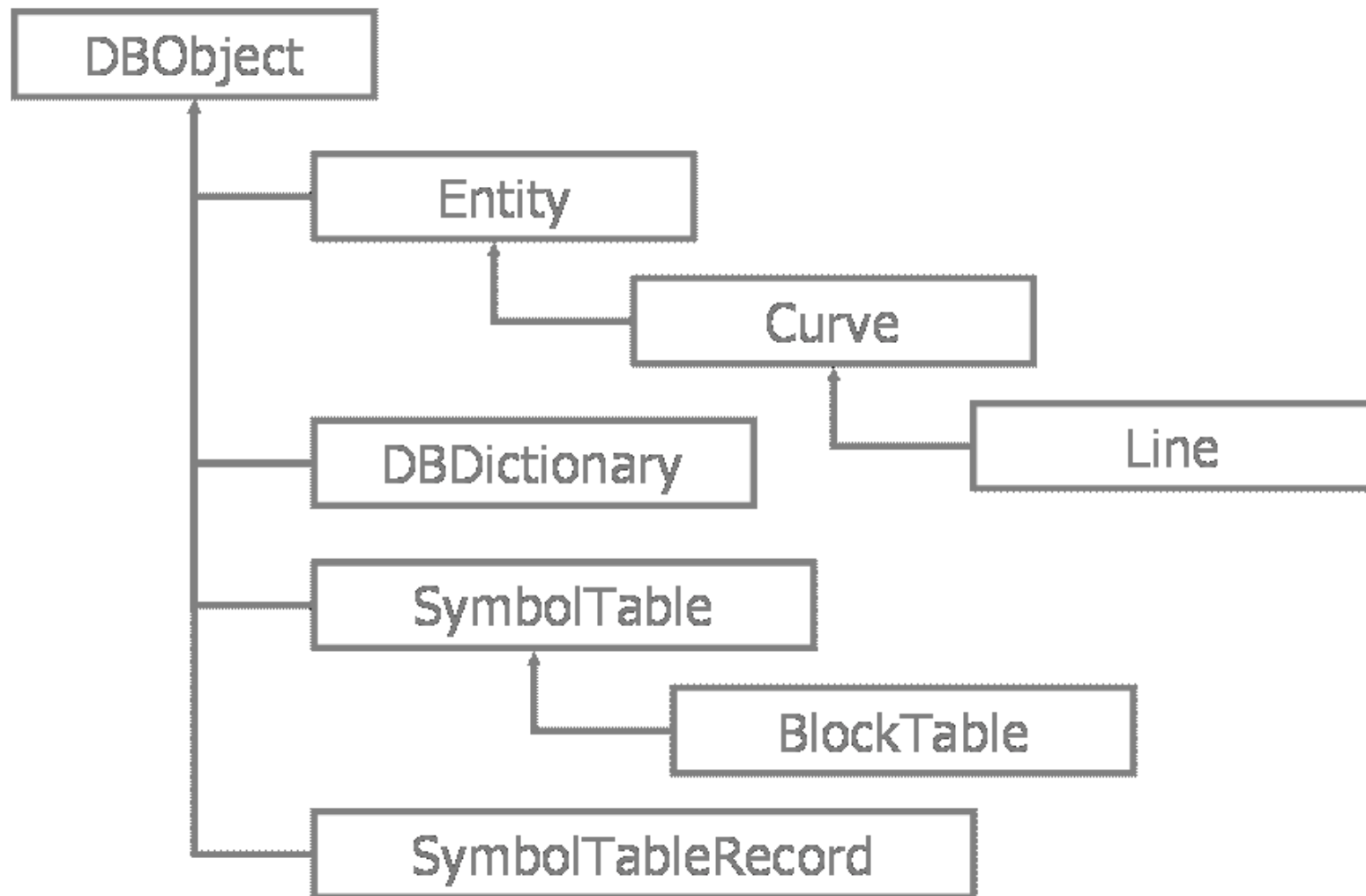
Überblick AutoCAD-Objektmodell

- classmap.dwg (in der ObjectARX Distribution)



Quelle: Autodesk

Wichtige AutoCAD Managed Klassen



Database resident objects

Quelle: Autodesk

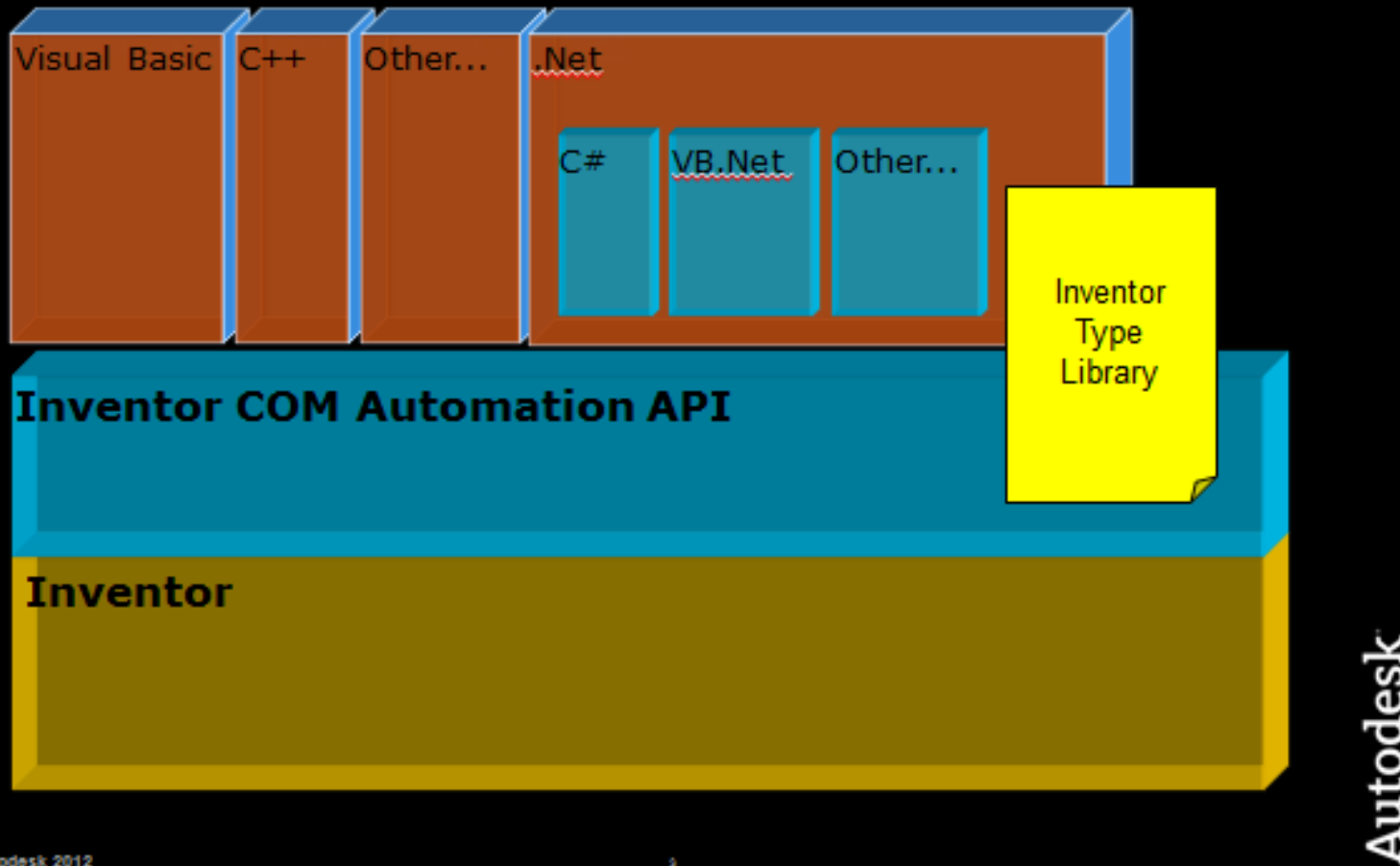
■ Iteration am Beispiel von BlockTableRecords

```
Database mm_db = HostApplicationServices.WorkingDatabase();  
//Blocktable anhand der festen ID in der Database holen  
BlockTable mm_bt = mm_trans.GetObject(mm_db.BlockTableId,  
                                     OpenMode.ForRead) as Blocktable;  
//Den BlockTable durchlaufen  
foreach(ObjectId mm_id in mm_bt)  
{  
    //Den BTR der ID zum Lesen öffnen  
    BlockTableRecord mm_btr = mm_trans.GetObject(mm_id,  
                                                  OpenMode.ForRead);  
}
```




Inventor .NET API

COM API Model



Quelle: Autodesk

- Nach der Inventor-Installation gibt es unter Win7 den Ordner
 - C:\Users\Public\Documents\Autodesk\Inventor 2013\SDK
- In diesem Ordner befinden sich zwei Installationsroutinen:
 - DeveloperTools.msi
 - UserTools.msi
- Für die Entwicklungsumgebung für Inventor 2013 gilt Gleiches wie unter AutoCAD 2013: Visual Studio 2010, unter 2012 lassen sich die Addins für Visual Studio ohne Tricks nicht installieren
 - Einen Hinweis, damit zumindest ein Projekttyp Inventor-AddIn installiert werden kann, findet sich auf <http://forums.autodesk.com/t5/Inventor-Customization/Autodesk-Inventor-Wizards-for-Microsoft-Visual-Studio-2012/td-p/3634414>
 - Es ist aber möglich, auch ohne Wizard Inventor-Addins zu entwickeln.

- Ein .NET-Addin für Inventor ist immer eine Klassenbibliothek. Das manuelle Aufsetzen eines Projektes beginnt also mit der Erstellung eines Klassenbibliotheks-Projektes
- In diesem Projekt werden folgende Verweise gesetzt:
 - Inventor-Type-Library (als COM-Verweis)
- Der Befehlsumfang eines .NET-Addins in Inventor ist gleich dem COM-Funktionsumfang, wenn Inventor also über einen externen Prozess bedient wird. Dies ist in AutoCAD nicht der Fall.
 - Da die Inventor API auf COM basiert, kann so eine externe Exe das zu entwickelnde AddIn abbilden. Der Funktionsumfang der Exe muß dann nur parallel in ein AddIn integriert werden, der Aufwand besteht in der Erstellung zweier Projekte, die in weiten Teilen den gleichen Quelltext benutzen
- Beispiele für Inventor-AddIns oder externe Zugriffe befinden sich in den Developer-Tools
- Für einfache Aufgaben steht der Apprentice-Server zur Verfügung, der im Vergleich zum Inventor schneller lädt, aber stark eingeschränkt hinsichtlich der Modifizierung ist. Die Befehle unterscheiden sich z.T. ebenfalls leicht

- Anders als in AutoCAD, wo eine öffentliche Routine die Schnittstelle zu AutoCAD darstellt, wird in Inventor eine Klasse benötigt, die das Interface `Inventor.ApplicationAddInServer` implementiert und die Kommunikation zu Inventor herstellt. Über das `GuidAttribute` erhält die Klasse eine Identifikation, die mit dem GUID-Tool erstellt werden sollte, um die GUID einzigartig zu machen

```
[GuidAttribute("c8b19dba-21d0-463e-81df-a22a7aaedd68")]  
public class StandardAddInServer : Inventor.ApplicationAddInServer  
{  
    ...  
}
```

- In dieser Klasse müssen folgende Methoden implementiert werden:
 - `public void Activate(Inventor.ApplicationAddInSite addInSiteObject, bool firstTime)`
 - `public void Deactivate()`
 - `public void ExecuteCommand(int commandID) //Leer, Kompatibilität zu alten Versionen, nicht mehr zu verwenden!!`
 - `public object Automation //Möglichkeit von außen auf das AddIn zuzugreifen`

Beispiel für Activate() in VB

```
Public Sub Activate(ByVal AddInSiteObject As Inventor.ApplicationAddInSite, _  
                    ByVal FirstTime As Boolean) Implements  
Inventor.ApplicationAddInServer.Activate  
    ' Save a reference to the Inventor Application object.  
    oApp = AddInSiteObject.Application  
  
    ' Load the icon from an external file. This could be done in many different  
    ' ways, i.e. from a control or a resource file. When read in using .Net you'll  
    ' have an Image object. In VB 6 it is a Picture object. Inventor will expect a  
    ' Picture object (actually an IPictureDisp object) so we'll need to convert the  
    ' image to use it in Inventor.  
    Dim oImage As System.Drawing.Image  
    oImage = System.Drawing.Bitmap.FromFile("C:\Temp\Button.bmp")  
  
    ' Convert the Image to a Picture.  
    Dim oPicture As stdole.IPictureDisp  
    oPicture =  
Microsoft.VisualBasic.Compatibility.VB6.Support.ImageToIPictureDisp(oImage)
```

Beispiel für Activate in VB - 2

```
' Create the button definition object.
oBtnDef = oApp.CommandManager.ControlDefinitions.AddButtonDefinition("Test", _
    "TestButton", Inventor.CommandTypesEnum.kQueryOnlyCmdType, _
    "{76BE5CB6-6778-4e5f-B3F7-5C58DD99EE94}", "A test command.", _
    "Test command", oPicture)

' Check if this is the first time the Add-In's been loaded.
If FirstTime Then
    ' Create a new toolbar.
    Dim oCommandBar As Inventor.CommandBar
    oCommandBar = oApp.UserInterfaceManager.CommandBars.Add("Test", _
        "AddInSampleCommandBar", _
        Inventor.CommandBarTypeEnum.kRegularCommandBar, _
        "{76BE5CB6-6778-4e5f-B3F7-5C58DD99EE94}")

    ' Add the button to the toolbar.
    oCommandBar.Controls.AddButton(oBtnDef)

    ' Make the toolbar visible.
    oCommandBar.Visible = True
End If
End Sub
```

Quelle: Autodesk User Group

- Ein Befehl in einem Inventor-Addin wird immer durch eine Callback-Funktion auf eine GUI-Funktion ausgeführt.
- Dazu muß entweder ein eigener Toolbar-Button erzeugt werden, dessen Callback-Funktion dann die neue Funktion für Inventor implementiert oder ein entsprechendes neues Kontextmenü-Element
- Als Architekturmuster könnte der Befehlscode für den Button in einer externen DLL liegen. Dies hätte den Vorteil, dass während des Debuggens der Code über einen externen Prozess ausgeführt werden kann, Inventor also nicht immer neu gestartet werden muss, aber ebenso über den Verweis direkt in Inventor benutzt werden kann.

- Ein AddIn kann als COM-Komponente registriert werden. Das ist aber wenig nachvollziehbar, weshalb die zweite Version bevorzugt werden sollte.
- .NET AddIns können auch ohne Registry automatisch geladen werden. Durch den Wizard wird eine .addin-Datei erzeugt, die in den Inventor-bin-Ordner (i.d.R. C:\Program Files\Autodesk\Inventor 2013\Bin) kopiert werden muss und etwa so aussehen kann:

```
<Addin Type="Standard">
  <!--Created for Autodesk Inventor Version 17.0-->
  <ClassId>{c8b19dba-21d0-463e-81df-a22a7aaedd68}</ClassId>
  <ClientId>{c8b19dba-21d0-463e-81df-a22a7aaedd68}</ClientId>
  <DisplayName>InventorAddIn1</DisplayName>
  <Description>InventorAddIn1</Description>
  <Assembly>InventorAddIn1.dll</Assembly>
  <LoadOnStartup>1</LoadOnStartup>
  <UserUnloadable>1</UserUnloadable>
  <Hidden>0</Hidden>
  <SupportedSoftwareVersionGreaterThan>16..</SupportedSoftwareVersionGreaterThan>
  <DataVersion>1</DataVersion>
  <UserInterfaceVersion>1</UserInterfaceVersion>
</Addin>
```

- Ein AddIn kann auch von einer externen EXE angesprochen werden:

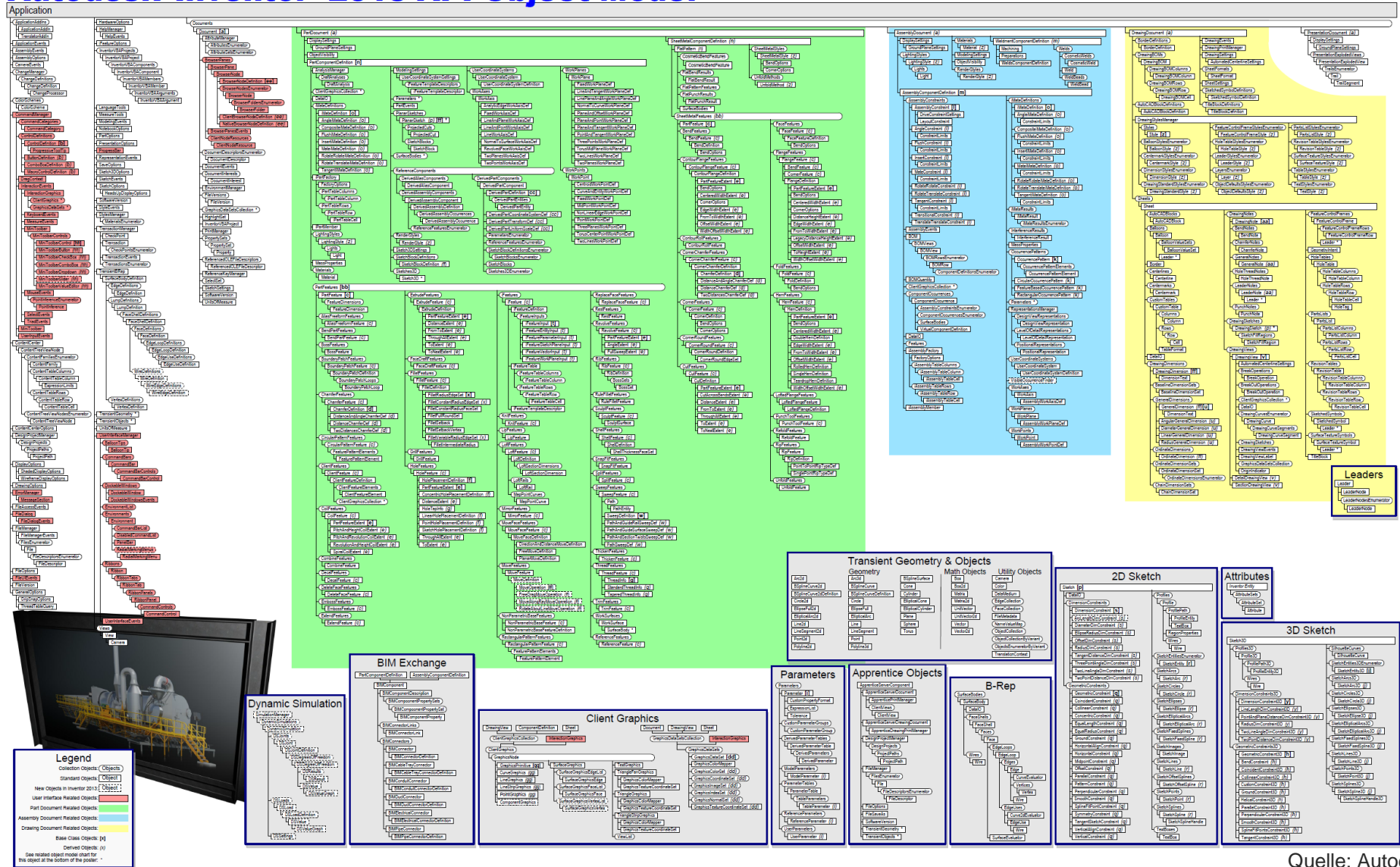
```
try
{
    //Zugriff auf das AddIn selber
    Inventor.ApplicationAddIn addIn =
        InvApp.ApplicationAddIns.ItemById(
            "{7ce7b998-b41b-4ff5-b62b-19e73fc5ec7b}");

    //Zugriff auf das Automation-Objekt des AddIn
    MyInt addInObj = addIn.Automation as MyInt;
}
Catch {Exception ex}
{
}
```

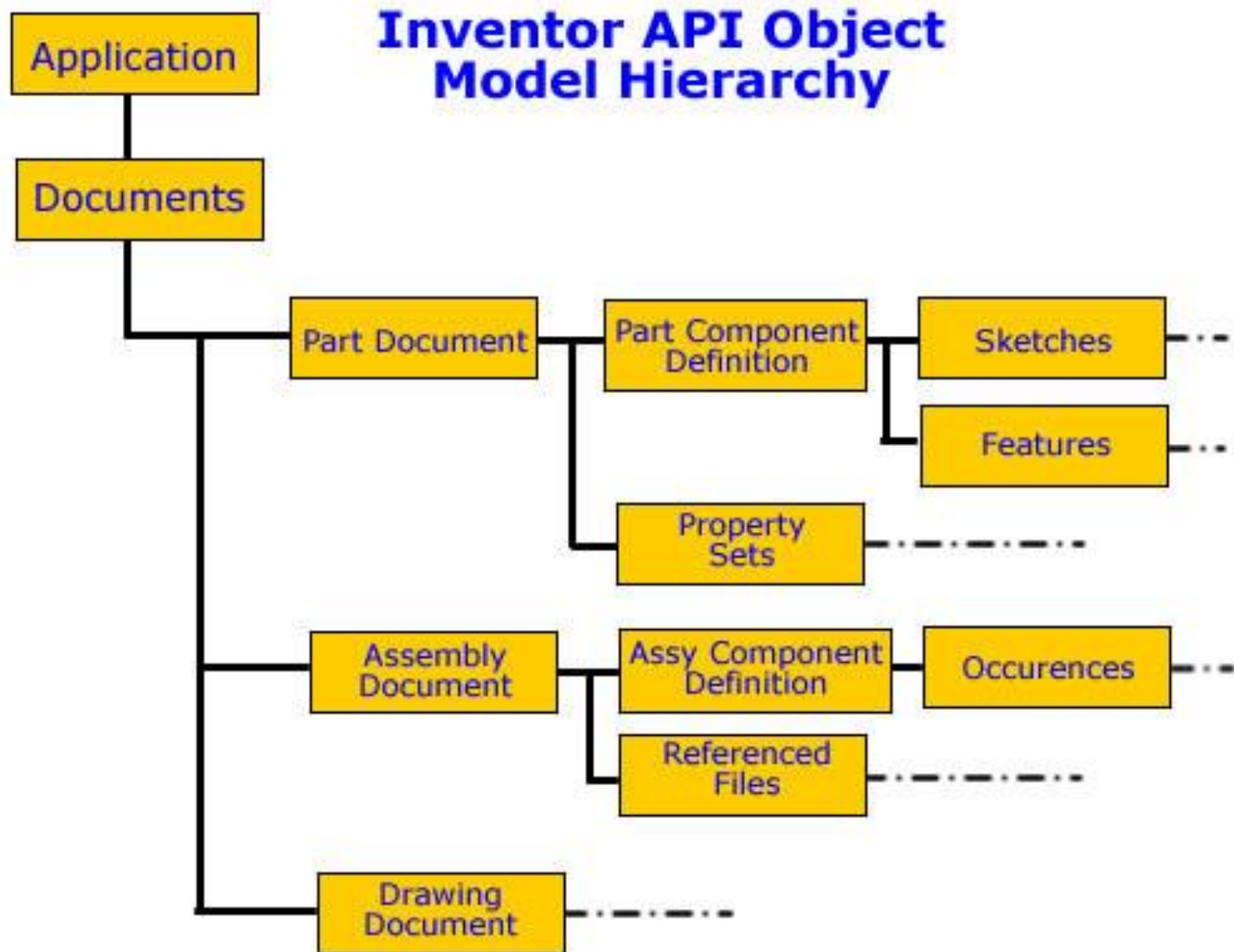
Das Inventor-Objektmodell

Autodesk® Inventor® 2013 API Object Model

Version 2, Feb. 9, 2012



Quelle: Autodesk



Quelle: Autodesk User Group

- Anders als in den AutoCAD-Blocktablerecords werden in Inventor die Elemente in eigenen Typ-eigenen-Containern verwaltet.
- Bei den Features beispielsweise gibt es Container für Revolve, Extrude und andere Features. Den Container zugeordnet sind spezielle Methoden zur Erzeugung der Features

```
//Die möglichen Profile aus einer 2D-Skizze ermitteln
Profiles profiles = planarSketch.Profiles;
//Das erste Profil zur Erzeugung eines Solids holen
//(könnten mehrere sein, daher Skizzen einfach halten)
Profile profile = profiles.AddForSolid(true, sketchLineCollection, 0);
//Liste der Teilefeatures aus der PartComponentDefinition holen
PartFeatures partFeatures = partCompDef.Features;
//Daraus die Extrusionsfeatures holen
ExtrudeFeatures extrudeFeatures = partFeatures.ExtrudeFeatures;
//und ein neues Feature hinzufügen (verschiedene Methoden verfügbar)
ExtrudeFeature newFeat = extrudeFeatures.AddByDistanceExtent(profile,
    m_Extents, m_FeatureExtentDirection,
    PartFeatureOperationEnum.kCutOperation, 0);
```

- Ähnliches wie für das PartDocument und die Features gilt auch für ein DrawingDocument
- Hier gibt es die Sheets-Auflistung, die alle Blätter des Dokumentes enthält
- Jedes Sheet hat einen Container DrawingDimensions
- Darunter gibt es beispielweise die Liste der GeneralDimensions, die normale lineare Bemaßungen enthält (`m_Sheet.DrawingDimensions.GeneralDimensions`)
- In den verschiedenen Dimensions-Listen gilt es folglich nach Bemaßungen zu suchen, die Prüfbemaßungen sind

- Ein Hersteller von Elektromotoren mit angeflanschem Getriebe will einen Konfigurator erstellen lassen, mit dem die unterschiedlichen Getriebe- und Motorenvarianten interaktiv konfiguriert werden können. Dabei soll das jeweilige Ergebnis direkt im 3D-CAD-System visualisiert werden. Stellen Sie die wesentlichen Anwendungsfälle dar und konzipieren Sie das GUI.
Erarbeiten Sie eine mögliche Strukturierung des Produktes.
 - Varianten werden gebildet über:
 - Gehäuseanschluss, Wellendurchmesser, Wellenanschluss, Auftretende Kräfte an der Abtriebswelle
 - Drehzahlen, Momente, Leistungen, Motorentyp
 - Spannung, Frequenz, Spannungsart
 - Isolation, Explosionsschutz, Temperaturbereich
 - Getriebeart, Getriebequalität
 - Welche logischen Zusammenhänge zwischen den einzelnen Komponenten wären bei der Abbildung zu berücksichtigen?
 - Wie könnte eine Benutzeroberfläche aussehen?