ETO Potlatch

Using .Net from Intent Rules

Vol. 2, 11-Oct-2012

Online Resources:

Forum: http://forums.autodesk.com/t5/Autodesk-Inventor-Engineer-to/bd-p/184
Wiki help: http://wikihelp.autodesk.com/Inventor_ETO/enu/2013

Autodesk

Welcome / Agenda

- Introductory Remarks
- 30 minutes presentation on selected topic:
  - "Using .Net from Intent Rules" – Jon Balgley
- 20 minutes Q&A and discussion "on-topic"
- 10 minutes "Three Tips"
- 20 minutes Q&A and discussion "on any topic"

Autodesk

Introductory Remarks

Autodesk

Overview: Using .Net from Intent Rules

- Why use .Net?
- How to call standard libraries
- How to write and use custom libraries
- Limitations

Autodesk

Why Use .Net?

- Many capabilities already implemented
  - File i/o and management, networking, image manipulation, etc
  - Legacy/external systems – ERP, CRM, etc.
  - Inventor API
- Wrapper around legacy apps
- High performance for algorithmic computations
- Easy to call from Intent rules

Calling Standard Libraries

Home | Library | Learn | Downloads | Support | Community | Sign in | United States - English

Search MSDN with Bing

- MSDN Library
- .NET Development
- .NET Framework 4.5
  - .NET Framework Class Library
    - System
    - System.Activities Namespaces
    - System.AddIn Namespaces
    - System.CodeDom Namespaces
    - System.Collections Namespaces
    - System.ComponentModel Namespaces
    - System.Configuration Namespaces
    - System.Data Namespaces
    - System.Deployment Namespaces
    - System.Device.Location
    - System.Diagnostics Namespaces
    - System.DirectoryServices Namespaces
    - System.Drawing Namespaces
    - System.Dynamic
    - System.EnterpriseServices Namespaces
    - System.Globalization
    - System.IdentityModel Namespaces
    - System.IO Namespaces
    - System.Linq Namespaces
    - System.Management Namespaces
    - System.Media
    - System.Messaging Namespaces
    - System.Net Namespaces
    - System.Numerics
    - System.Printing Namespaces
    - System.Reflection Namespaces
    - System.Resources Namespaces
    - System.Runtime Namespaces
    - System.Security Namespaces
    - System.ServiceModel Namespaces

msdn

.NET Framework Class Library

.NET Framework 4.5 | Other Versions | 45 out of 52 rated this helpful - Rate this topic

The .NET Framework class library is a library of classes, interfaces, and value types that provide access to system functionality. It is the foundation on which .NET Framework applications, components, and controls are built. The namespaces and namespace categories in the class library are listed in the following table and documented in detail in this reference. (Note that the table of contents lists the namespaces and categories by usage, with the most frequently used namespaces appearing first. The following list is ordered alphabetically, to provide an alternate way to navigate the class library.)

Namespaces

Namespace	Description
System	The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
System.Activities	The System.Activities namespaces contain all the classes necessary to create and work with activities in Window Workflow Foundation.
System.AddIn	The System.AddIn namespaces contain types used to identify, register, activate, and control add-ins, and to allow add-ins to communicate with a host application.
System.CodeDom	The System.CodeDom namespaces contain classes that represent the elements of a source code document and that support the generation and compilation of source code in supported programming languages.
System.Collections	The System.Collections namespaces contain types that define various standard, specialized, and generic collection objects.

- Etc.

Autodesk

Calling Standard Libraries (aka Assemblies)

- “System” assemblies, or other special pre-loaded assemblies
- Call with fully-qualified reference, no ‘using’
  - e.g. System.IO.File.Exists

```
<%%Category("Simple")> _
Rule fileDoesExist As Boolean = system.io.file.exists("c:\temp\output.txt")


<%%Category("Simple")> _
Rule fileDoesNotExist As Boolean = system.io.file.exists("c:\temp\outputxxxxxxx.txt")
```

Simple	
fileDoesExist	True
fileDoesNotExist	False

- Simple data types (strings, numbers, booleans) “just work”
- NOT case-sensitive

Autodesk

Using .Net Objects

- “As Any”
- Use as normal Intent reference

```
<%%Category("File IO")> _
Rule newDirInfo As Any = system.io.directory.createDirectory("c:\temp\newDir")

<%%Category("File IO")> _
Rule newDirTime As Any = newDirInfo.CreationTime

<%%Category("File IO")> _
Rule weekDay As Any = newDirTime.DayOfWeek
```

File IO

newDirInfo	<newDir>
newDirTime	<10/8/2012 11:13:49 AM>
weekDay	<Monday>

Directory.CreateDirectory Method (String)

.NET Framework 4.5 | Other Versions | This topic has not yet been rated - Rate this topic

Creates all directories and subdirectories in the specified path.

Namespace: System.IO
Assembly: mscorlib (in mscorlib.dll)

Syntax

C# | C++ | F# | VB

```
public static DirectoryInfo CreateDirectory(
        string path
)
```

Parameters

path

Type: System.String

The directory path to create.

Return Value

Type: System.IO.DirectoryInfo
An object that represents the directory for the specified path.

Autodesk

Using .Net Enumerators

- Use “For Each”

```
<%%Category("Enumerator")> _
Rule tempFileCount As Integer
    Dim tempDirEnum As Any = system.io.directory.enumerateFiles("c:\temp")
    tempFileCount = 0
    For Each f In tempDirEnum
        tempFileCount = tempFileCount + 1
    Next f
End Rule
```

Enumerator	
tempFileCount	70

Autodesk

Using “New”

- Use the “New” operator to make .Net objects
  - [StringBuilder is an efficient way to make long strings]

```
<%%Category("Struct")> _
Rule strAllCustNames As String
    Dim sb As Any = New system.text.stringbuilder(100)
    For Each c In customers
        sb.Append(c.firstname)
        sb.append(" ")
        sb.append(c.lastname)
        sb.append(", ")
    Next c
    sb.Remove(sb.Length-2, 2) 'Remove trailing comma-space
    strAllCustNames = sb.ToString()
End Rule
```

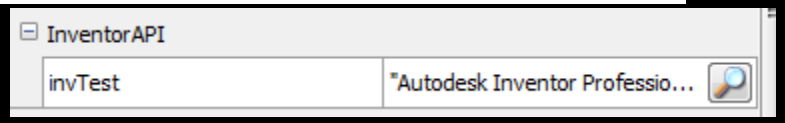
- Use “.” to call methods and get property values

Calling Inventor API

- Same as any other .Net interface
  - Class methods
  - Get objects back and refer to properties/methods
  - Use ‘new’ when necessary
  - Use enumerators when necessary
  - Start with ‘%%InventorApplication’

- Simple example:

```
<%%Category("InventorAPI")> _
Rule invTest As Any = %%inventorApplication.Caption
```



- Too many details – topic for other Potlatch sessions
  - Can mess up Intent!

Autodesk

Issues with .Net Libraries

- Caching
- Side-effects / state / order-of-evaluation
- Contextual Actions
  - Writing to a file
  - Adding to a sketch
- “Finishing” actions (e.g., closing a file/sketch)
- Returning ‘null’ from .Net is NoValue [with all the ramifications]

Summary of Using Standard .Net Libraries

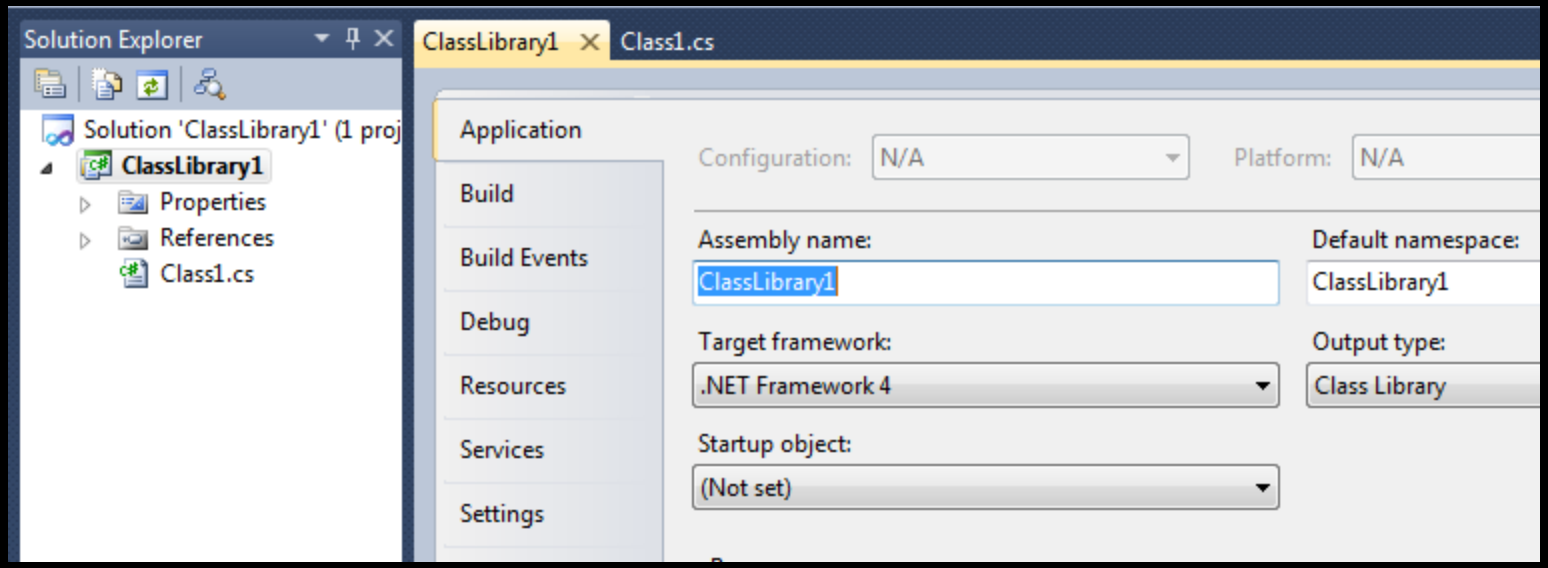
- Call directly from Intent rules
- Use basic datatypes
- Use “For Each”
- Use “New”

Autodesk

Creating Custom .Net Libraries

Autodesk

Custom .Net Class Libraries - Project



Autodesk

Custom .Net Class Libraries - Implementation

- Write classes, methods, and properties
  - Inventor must be stopped
- Install DLL to “Design Files” folder
  - Restart Inventor

ClassLibrary1 | Class1.cs

ClassLibrary1.Class1 | public class Class1

ClassLibrary1.Class1 | GetTheString(string

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassLibrary1
{
    public class Class1
    {
        private static string m_theString = "This is the string";

        public static string TheString
        {
            get { return m_theString; }
        }

        public static string GetTheString(string suffix)
        {
            return m_theString + suffix;
        }
```

centerSupporAssy.iks	9/27/2012 4:36 PM	IKS File	1 KB
centerSupporAssyAdopt.iks	9/27/2012 4:36 PM	IKS File	2 KB
ClassLibrary1.dll	10/8/2012 1:08 PM	Application extens...	5 KB
curvedBeam.iks	9/29/2012 1:39 PM	IKS File	2 KB
curvedBeamAdopt.iks	9/29/2012 4:44 PM	IKS File	4 KB

Autodesk

Custom .Net Class Libraries - Usage

- Call in all the usual ways

```
<%%Category("Custom")> _
Rule theString_method As String = classLibrary1.class1.getTheString(" that I wanted")
<%%Category("Custom")> _
Rule theString_property As String = classLibrary1.class1.TheString
```

Custom	
theString_method	"This is the string that I wanted"
theString_property	"This is the string"

- Intent finds DLL via reflection

Autodesk

Datatype Translation

- Intent datatypes ← → .Net datatypes
- Arguments, return value

Intent Datatype	.Net Datatype
Boolean, Integer, Number, String	Same
List	Object[ ]
Point, Vector, Frame	Autodesk.Intent.Point Autodesk.Intent.Vector Autodesk.Intent.Frame
Part	Risky to use this
NoValue	Null
(.Net object) "Any"	Other .Net objects
Name	Use strings instead

Example: Simple Customer List

Autodesk

```
<%%Category("Customers")> _
Rule cus1 As Any = {"Jon", "Balgley", "503-555-6789"}
<%%Category("Customers")> _
Rule cus2 As Any = {"Jon", "Yelglab", "503-555-9876"}
<%%Category("Customers")> _
Rule custs As List = {cus1, cus2}

<%%Category("Customers")> _
Rule allCusPhoneNumbers As List
    For Each c In custs
       allCusPhoneNumbers = allCusPhoneNumbers + {third(c)}
    Next c
End Rule
```

```
<%%Category("Customers")> _
Rule strAllCusNames As String
    Dim sb As Any = New system.text.stringbuilder(100)
    For Each c In custs
        sb.Append(first(c))
        sb.append(" ")
        sb.append(second(c))
        sb.append(", ")
    Next c
    sb.Remove(sb.Length-2, 2) 'Remove trailing comma-space
    strAllCusNames = sb.ToString()
End Rule
```

- Implemented with Lists

Customers

allCusPhoneNumbers	{"503-555-6789", "503-555-9876"}
cus1	{"Jon", "Balgley", "503-555-6789"}
cus2	{"Jon", "Yelglab", "503-555-9876"}
custs	{{"Jon", "Balgley", "503-555-6789"}, {"Jon", "Yelglab", "503-555-9876...
strAllCusNames	"Jon Balgley, Jon Yelglab"

Autodesk

Example: Lightweight "Struct"

- Class with a few "simple" properties, use instead of (sub) List. Why?
  - Data encapsulation – minimizes mistakes
  - Probably faster

- Members (private)

- Constructor (public)

```
namespace cs
{
    public class Customer
    {
        private string m_firstName;
        private string m_lastName;
        private string m_phone;

        public Customer(string firstName, string lastName, string phone)
        {
            m_firstName = firstName;
            m_lastName = lastName;
            m_phone = phone;
        }
```

- Properties:

```
public string FirstName
{
    get { return m_firstName; }
}
public string LastName
{
    get { return m_lastName; }
}
public string Phone
{
    get { return m_phone; }
}
```

- ToString (optional)

```
public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    sb.Append(FirstName);
    sb.Append(" ");
    sb.Append(LastName);
    return sb.ToString();
}
```

Use in Intent

- Make some instances

```
Rule c1 As Any = New cs.customer("Jon", "Balgley", "503-555-6789")
Rule c2 As Any = New cs.customer("Jon", "Yelglab", "503-555-9876")
Rule customers As List = {c1, c2}
```

c1	<Jon Balgley>
c2	<Jon Yelglab>
customers	{<Jon Balgley>, <Jon Yelglab>}

- Use the instances

```
Rule allPhoneNumbers As List
    For Each c In customers
        allPhoneNumbers = allPhoneNumbers + {c.phone}
    Next c
End Rule
```

allPhoneNumbers	{"503-555-6789", "503-555-9876"}

```
<%%Category("Struct")> _
Rule strAllCustNames As String
    Dim sb As Any = New system.text.stringbuilder(100)
    For Each c In customers
        sb.Append(c.firstname)
        sb.append(" ")
        sb.append(c.lastname)
        sb.append(", ")
    Next c
    sb.Remove(sb.Length-2, 2) 'Remove trailing comma-space
    strAllCustNames = sb.ToString()
End Rule
```

strAllCustNames	"Jon Balgley, Jon Yelglab"

Autodesk

Summary: Custom Library for “Struct”

- More “structured” than using a List
- Faster & more convenient than using a Part

Limitations

- No “index” operator … get_Item()
- get_PropertyName, set_PropertyName
- “Ref” arguments (for output)
- Template methods
- Casts, operators
- Not all “System” libraries available
- Workaround: Write a wrapper / auxiliary class

Summary: Custom .Net Libraries

- Easy to make and use
- Many good reasons for doing so
- Don’t forget why you’re using Intent!

Autodesk

When/Why to use .Net vs. Intent?

- Use Intent for modeling and for convenience

- Use .Net for interfacing to legacy and/or external systems
  - Wrap .Net in Intent designs for best integration

- Use .Net for complex (CPU-intensive), self-contained calculations
- Use .Net to optimize performance after finding bottlenecks

Summary

- Easy to call into .Net assemblies
  - Standard … “System” & Inventor & legacy apps
  - Custom … to augment Intent, wrap legacy apps

Autodesk

“On Topic” Q&A

Autodesk

Random Tips

1. Use IvBlock for debugging. This design uses a standard, simple factory file, and goes through all the factory- and member-file processing. If you have some unexpected behavior, use this design to eliminate your IPT as one of the possible causes. If it turns out to be an ETO defect, it will then be easier to report.

2. AutoSaveChanges? parameter. This parameter of IvAssemblyDocument controls whether or not member files are saved immediately, or only when the top-level assembly is saved. When set to False, it minimizes the number of possibly-extraneous member files.

3. Test rule? Prefix it with "_". This will put it at the beginning of the category, where it's easy to find.

Q&A – Open Discussion

Autodesk

Thanks!

- Send us suggestions for future topics
- Send us your favorite little “tips”
- See you next time!

Autodesk

Autodesk®