WebSphere Message Broker V8 .NET Integration

2012 Connectivity STEW

Agenda

- Why .NET?
- .NET Overview
  – Framework and CLR
- Integration with Broker
  – .NET Compute node
- Visual Studio Integration
  – Plugins
  – Debugging
- The Broker Plugin API
  – Navigation and Tree access
- Integrating .Net and COM applications
- ESQL Calling .NET
- Hosting the CLR
- App Domains
  – Creation and Configuration
- Summary

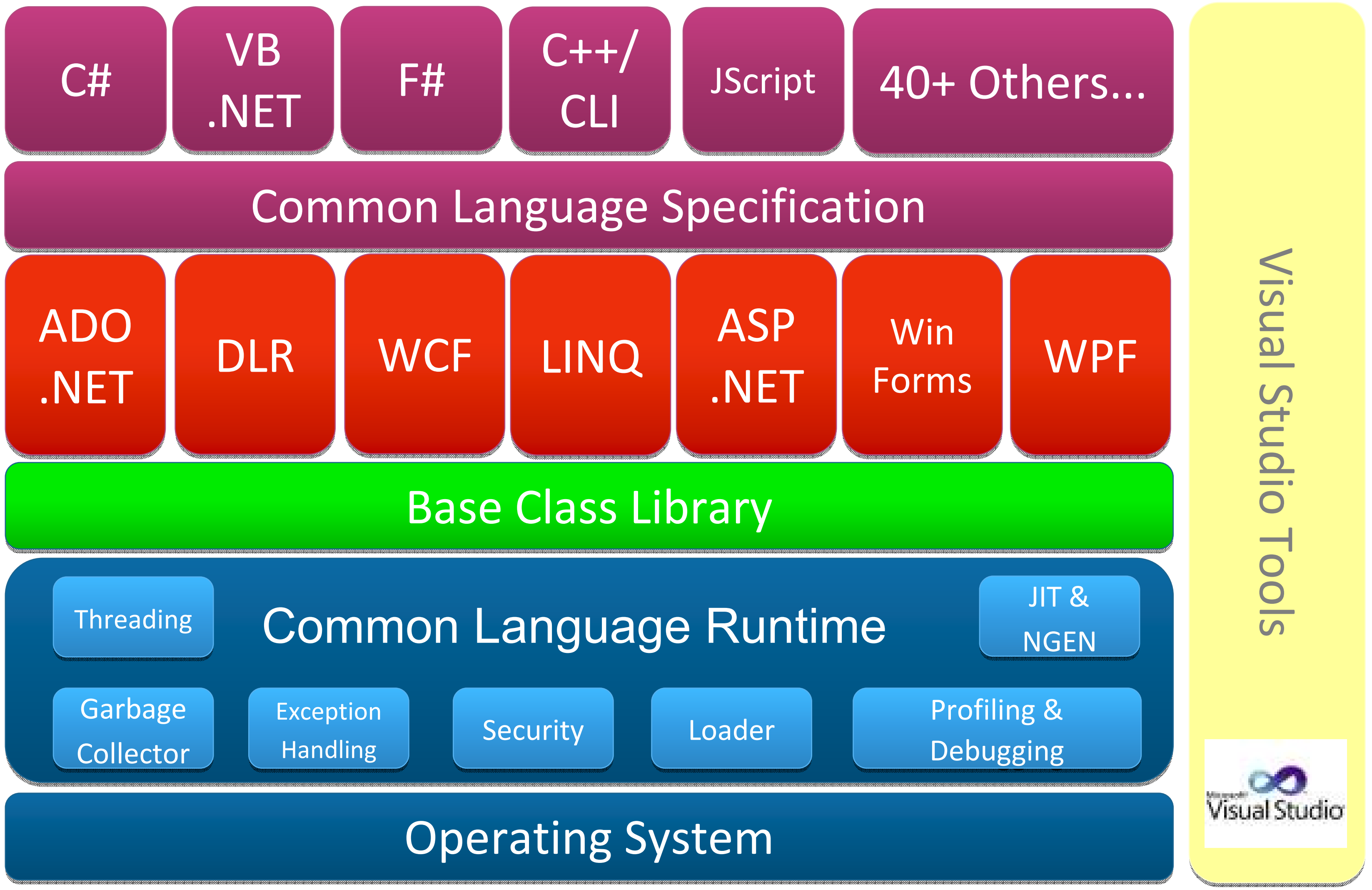
Why .NET?

- **Many clients have a large investment in Microsoft, .NET and related technologies**
  - Dynamics for CRM / ERP
  - SharePoint for collaboration
  - Visual Studio for development
  - Custom .NET applications
- **.NET is a very popular environment for developers.**
  - TIOBE Programming Community Index for June 2011. *[www.tiobe.com]*

Language	Access and Transform
Java	JavaCompute Node
C	C Plugin Node
C++	C Plugin Node
C#	.NET Compute Node
PHP	PHPCompute Node
VB	.NET Compute Node

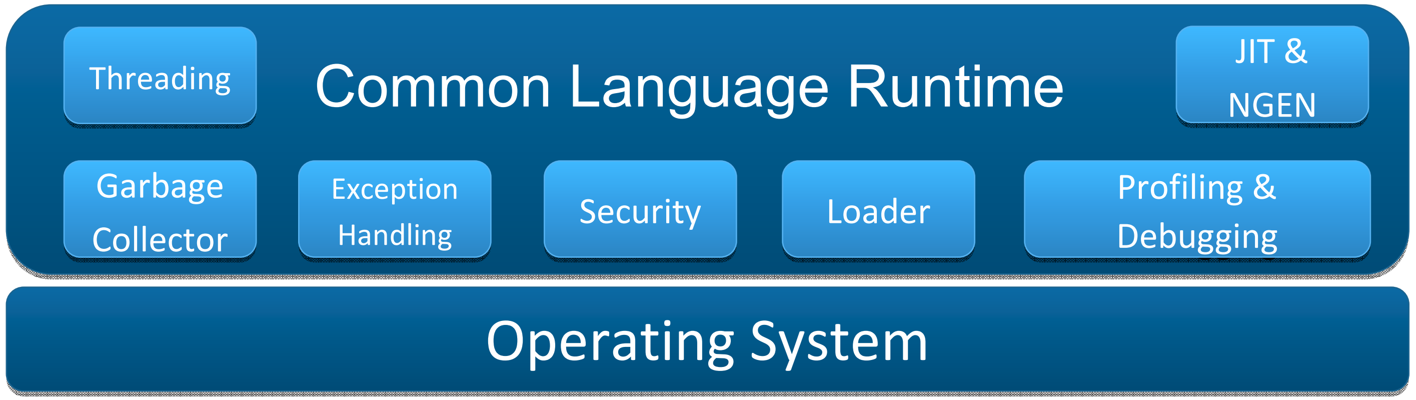
  - WMB now provides transformation capability for all of the top 6 languages

.NET Framework Overview

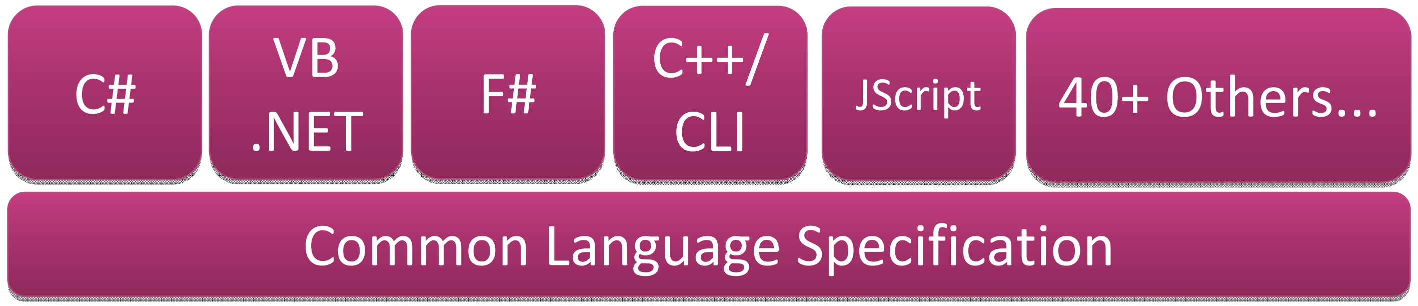


The .NET Common Language Runtime (CLR)

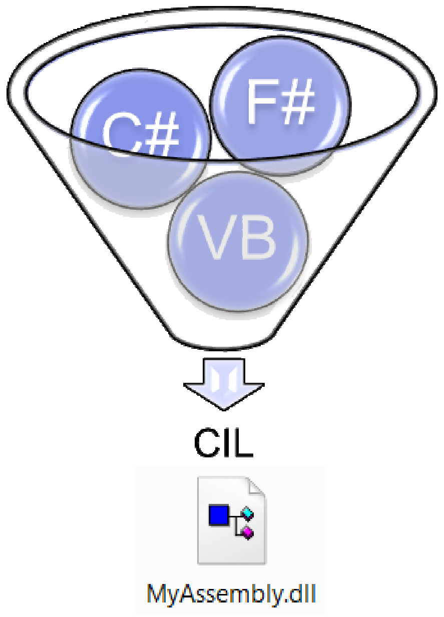
- **The CLR provides an environment inside which “managed” code is executed**
  - Similar to a JVM
- **Can be hosted inside another process**
  - Gives the ability to run managed code
    - SQLServer does this to run managed Stored Procedures
- **Provides key services to all code running inside it**
  - Loading, GC, Debugging etc.



From Source Code to Byte Code

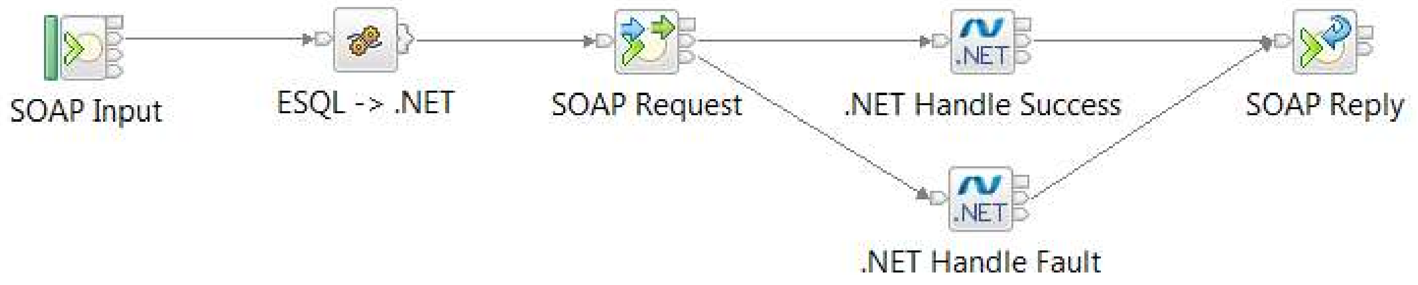


- **All .NET code is compiled from the source language into Managed “CIL” (MSIL) code**
- **Common Type System (CTS) and Common Language Spec (CLS)**
- **The CIL code lives in a .DLL or .EXE and is called an Assembly**
  - The Assembly is loaded into the CLR to be executed
  - Code is “JITted” before it is executed
    - Can be JITted before hand with NGEN
- **At runtime the CLR does not care what the source language was**



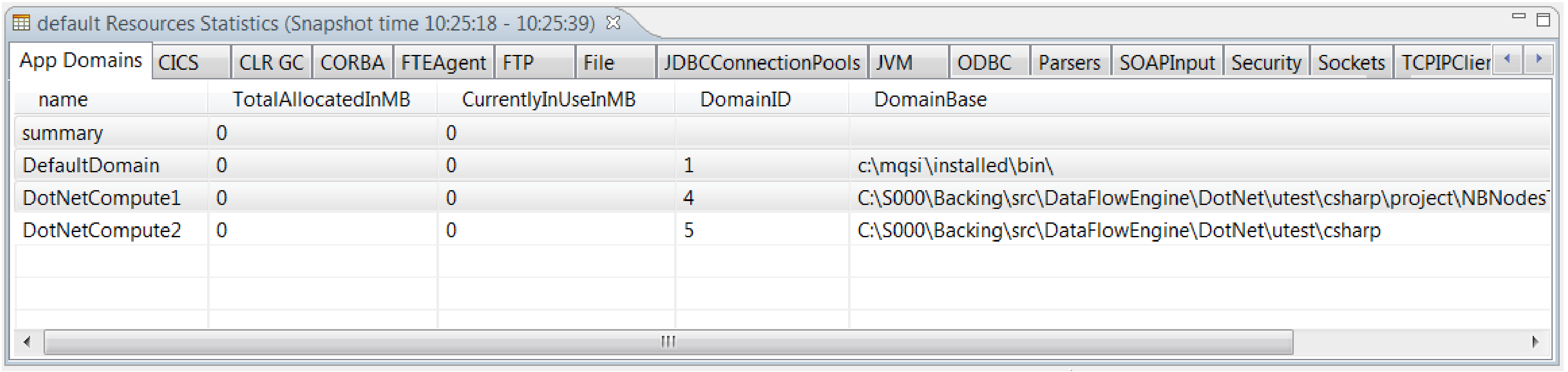
Integrating .NET with Message Broker

- **Extremely tight language agnostic integration**
  - Integrates any CLR language at a very low level with the broker
- **Create your own .NET Compute Nodes using Visual Studio**
  - Integrate new or existing .NET applications directly with your Message Flow
    - Write nodes in C#, VB, F#, C++/CLI, and many more
- **Tightly integrated with Visual Studio**
  - Broker toolkit can launch Visual Studio
  - Visual studio plugin to simplify node development
- **Call .NET code directly from ESQL**
  - Jump straight from ESQL into .NET code
- **Integrate with existing COM applications**



Managing .NET Integration

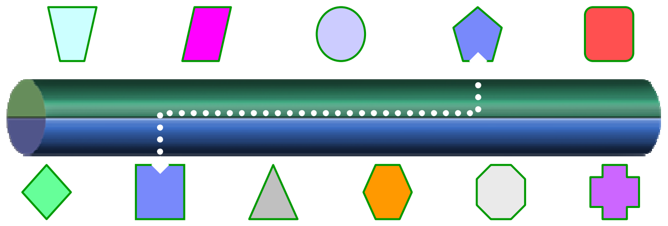
- **Fine grained operation control**
  - Configurable Service, Resource Stats
- **The CLR is hosted inside each Execution Group**
  - One CLR per E.G.
- **Each CLR is split into different App Domains**
  - Choose which App Domain your code runs in



name	TotalAllocatedInMB	CurrentlyInUseInMB	DomainID	DomainBase
summary	0	0		
DefaultDomain	0	0	1	c:\mqsi\installed\bin\
DotNetCompute1	0	0	4	C:\S000\Backing\src\DataFlowEngine\DotNet\utest\csharp\project\NBNodes
DotNetCompute2	0	0	5	C:\S000\Backing\src\DataFlowEngine\DotNet\utest\csharp

Running .NET in your Enterprise

- **.NET in Broker is supported on the Windows platform**
- **.NET functionality is available in all editions of Broker 8 (Express, Standard, Advanced, etc)**
- **Broker's existing connectivity options give flexible deployment options**
  - You can position .NET at the "edge" and connect to your main infrastructure
  - You can position .NET in the "middle" as part of your core infrastructure
  - Use any Broker transport option to make the links
    - MQ, WebService, etc



.NET Compute Node - What do you want to integrate today?

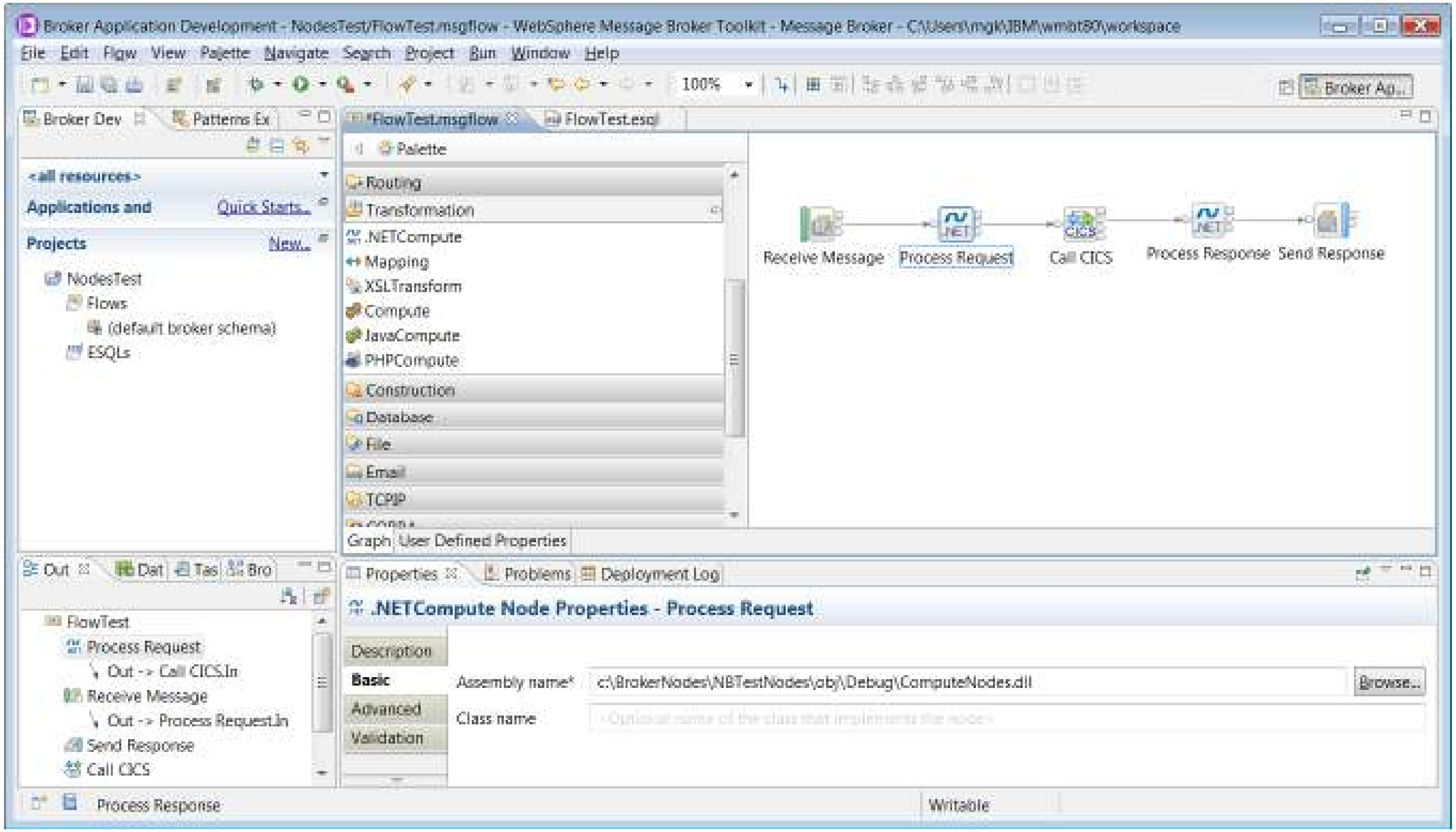
Insert into CRM
Change Excel
Call COM
Update SharePoint
Invoke Dynamics

- **First Class Broker transformation node**
  - similar to JavaCompute
- **Write your transformations in any CLR compliant language**
  - Build transformations in: C#, VB, F#, C++/CLI, Jscript, etc…
- **Allows you to integrate your .NET code directly with your Flow**
  - Three code "templates" to get you started
    - Filter Message
    - Modify Message
    - Create Message
- **Implement a single method "Evaluate"**
  - Stub is auto implemented in Visual Studio
- **Provides full access to the Broker Trees**
  - Message,
  - LocalEnvironment,
  - Environment,
  - ExceptionList
- **Dynamic Terminals**
  - As many as you need

```
using SharePointClient = Microsoft.SharePoint.Cl
  //Update SharePoint with details from Message
  private void UpdateSharepoint(NBElement fileIn
    string fileSource = (string)fileInfo["Locati
    string fileName = "/sites/pp/Documents/" + (
    ClientContext context = new ClientContext("h
    using (FileStream fs = new FileStream(fileSo
    {
      SharePointClient.File.SaveBinaryDirect(con
    }
  }
```

.NET Compute Node Configuration

- **Browse to choose Assembly**
- **Drag / Drop an assembly on the node to configure**



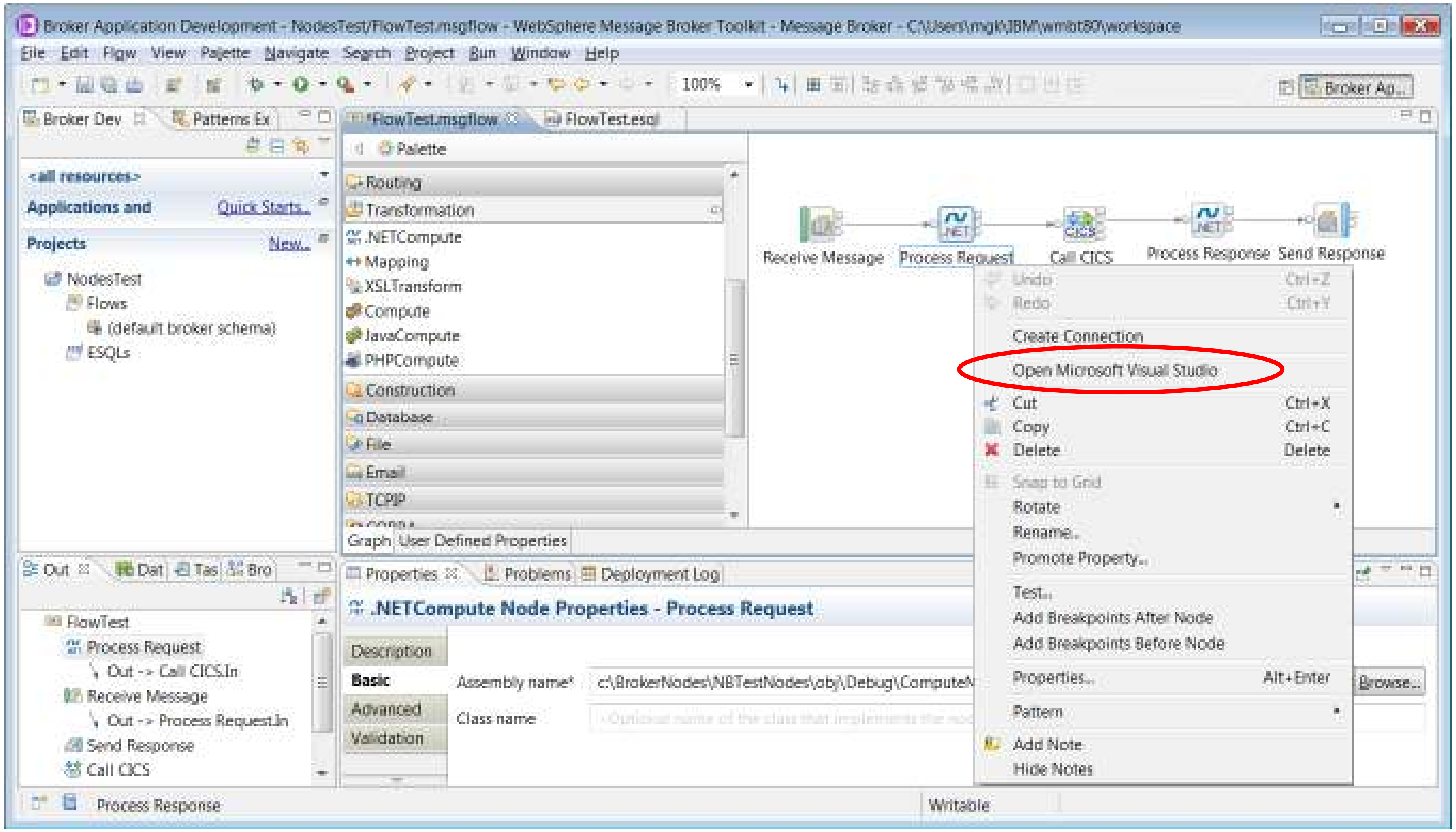
.NETCompute Node Properties - Process Request

Assembly name* c:\BrokerNodes\NBTestNodes\obj\Debug\ComputeNodes.dll

Class name <Optional name of the class that implements the node>

Launch Visual Studio directly from the .NET Compute node

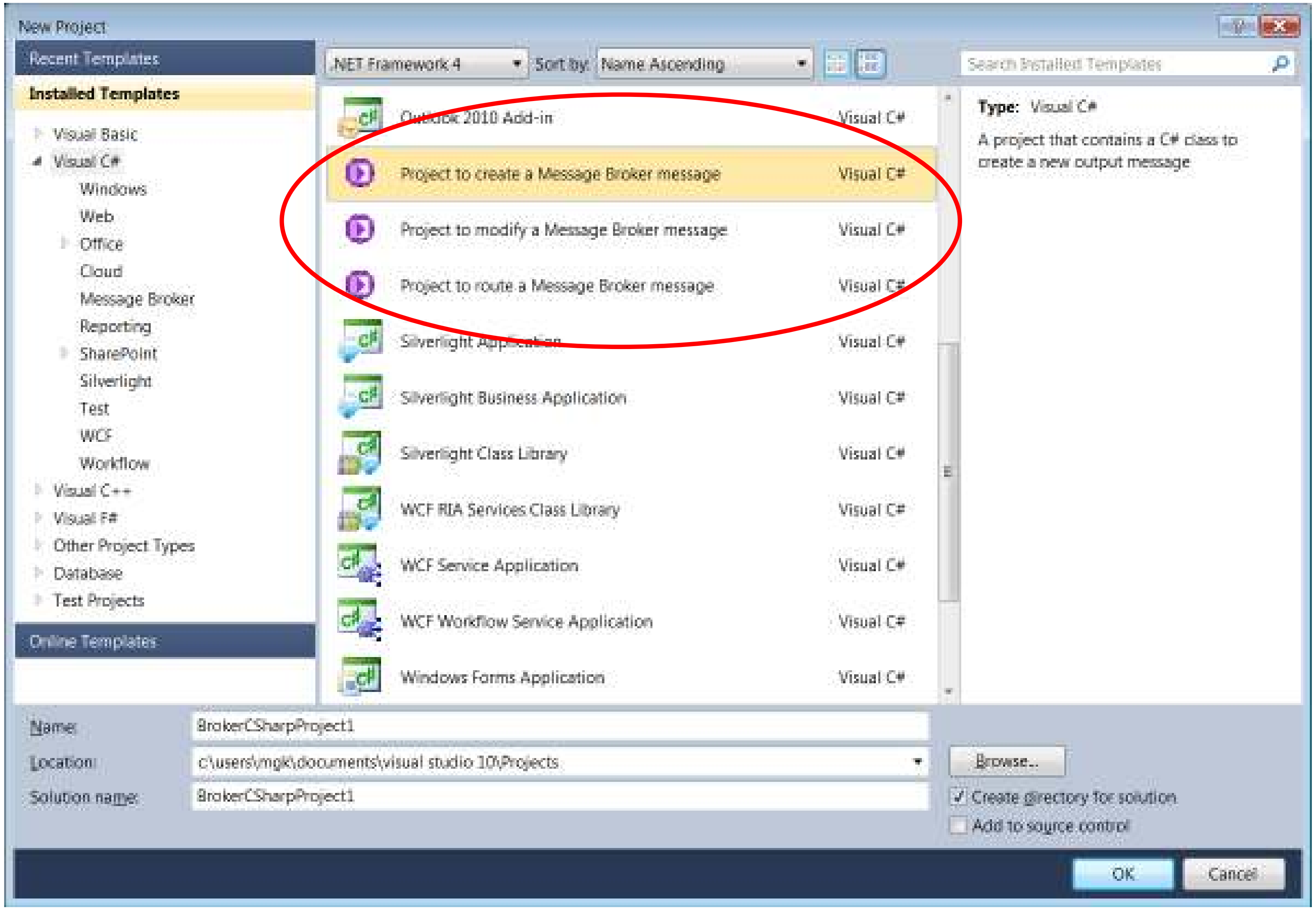
- **Simply double-click on the node**
  - Or right click "Open Microsoft Visual Studio"



Visual Studio 2010 Integration

- **Visual Studio is the "toolkit" when developing .NET assemblies**
  - Visual Studio is the "industry standard" for .NET development
- **Tight integration with the Broker runtime**
  - "Double Click" on a .NETCompute Node to launch Visual Studio
  - Node can be configured with a "Solution" to launch automatically
- **Plugin for Visual Studio to generate skeleton .NET Compute code**
  - Specific to the language choice and the node type (Filter / Modify / Create)
- **Use Visual Studio debugger to debug your .NET assemblies.**
  - "Attach" the debugger to the runtime "dataflowengine.exe" process for the E.G.
    - See all the message trees in their entirety.

Speed up development with the Visual Studio Broker Node Template



New Project

Recent Templates

Installed Templates

- Visual Basic
- Visual C#
  - Windows
  - Web
  - Office
  - Cloud
  - Message Broker
  - Reporting
  - SharePoint
  - Silverlight
  - Test
  - WCF
  - Workflow
- Visual C++
- Visual F#
- Other Project Types
- Database
- Test Projects

Online Templates

.NET Framework 4 | Sort by: Name Ascending | Search Installed Templates

Outlook 2010 Add-in	Visual C#
Project to create a Message Broker message	Visual C#
Project to modify a Message Broker message	Visual C#
Project to route a Message Broker message	Visual C#
Silverlight Application	Visual C#
Silverlight Business Application	Visual C#
Silverlight Class Library	Visual C#
WCF RIA Services Class Library	Visual C#
WCF Service Application	Visual C#
WCF Workflow Service Application	Visual C#
Windows Forms Application	Visual C#

Type: Visual C#

A project that contains a C# class to create a new output message

Name:	BrokerCSharpProject1
Location:	c:\users\mgk\documents\visual studio 10\Projects
Solution name:	BrokerCSharpProject1

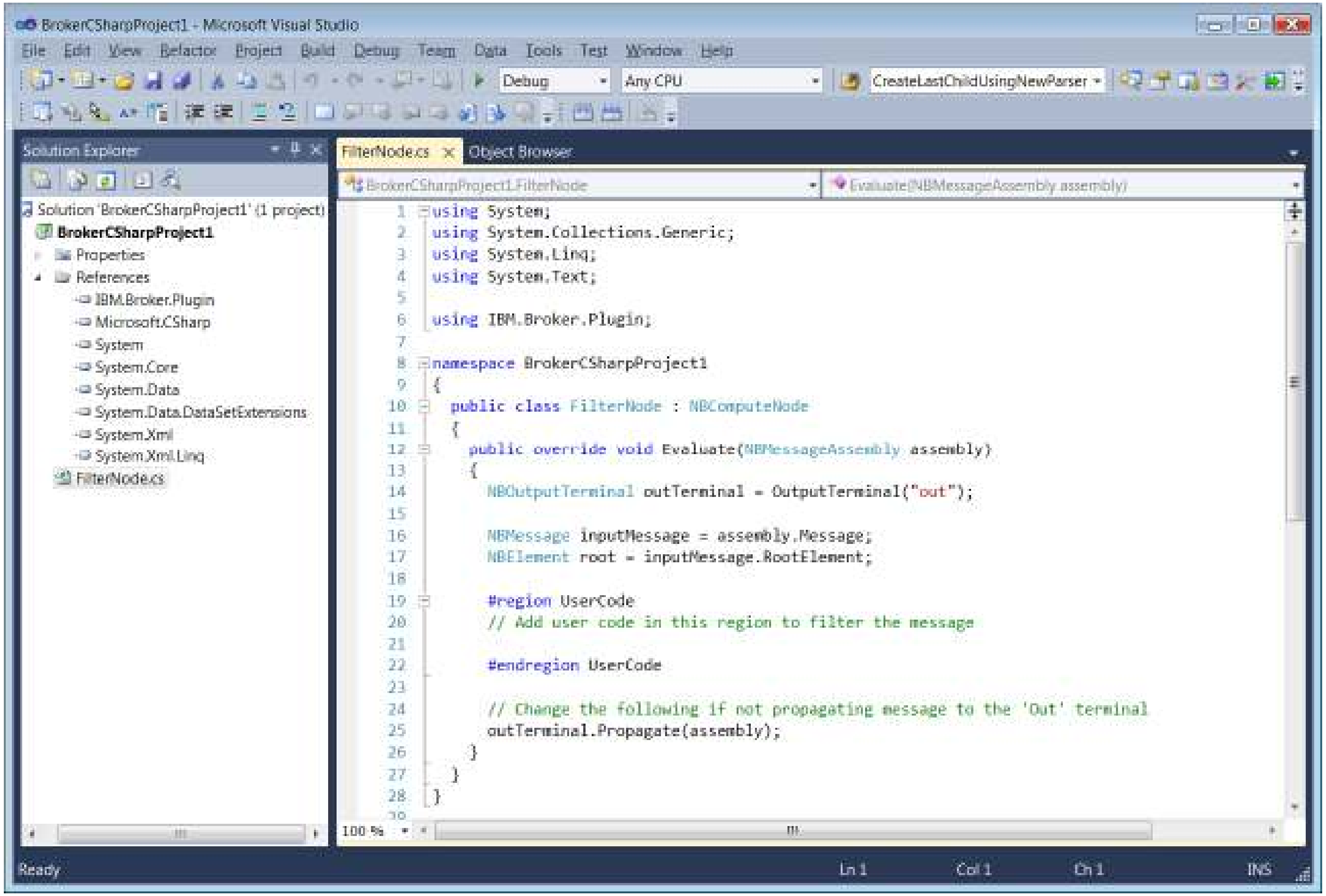
Browse...

[✓] Create directory for solution

[ ] Add to source control

OK | Cancel

Auto generated node templates for Visual Studio


BrokerCSharpProject1 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug
Any CPU
CreateLastChildUsingNewParser
Solution Explorer
Solution 'BrokerCSharpProject1' (1 project)
BrokerCSharpProject1
Properties
References
IBM.Broker.Plugin
Microsoft.CSharp
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Xml
System.Xml.Linq
FilterNode.cs
FilterNode.cs
Object Browser
BrokerCSharpProject1.FilterNode
Evaluate(NBMessageAssembly assembly)
100 %
Ready
Ln 1
Col 1
Ch 1
INS

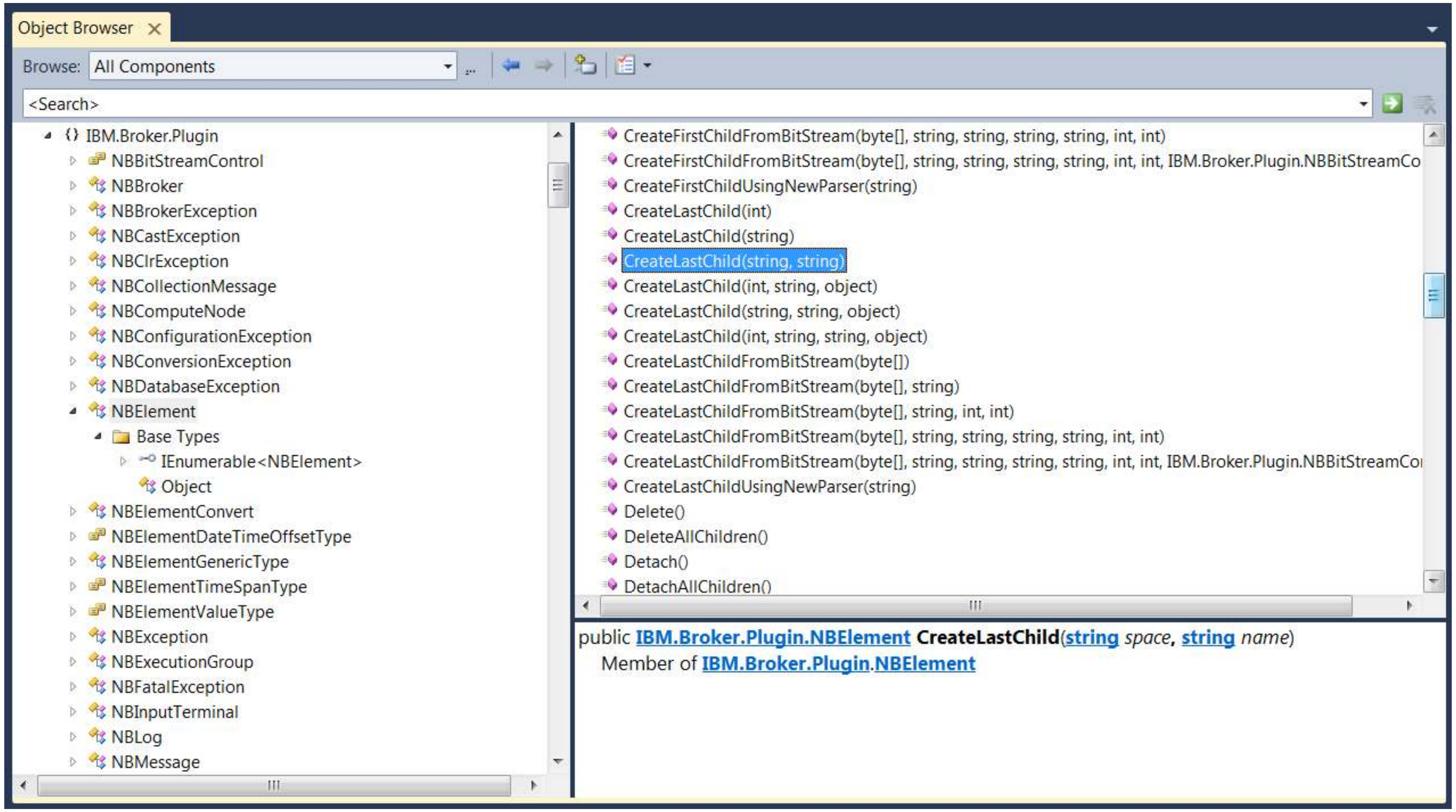
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using IBM.Broker.Plugin;

namespace BrokerCSharpProject1
{
  public class FilterNode : NBComputeNode
  {
    public override void Evaluate(NBMessageAssembly assembly)
    {
      NBOutputTerminal outTerminal = OutputTerminal("out");

      NBMessage inputMessage = assembly.Message;
      NBElement root = inputMessage.RootElement;

      #region UserCode
      // Add user code in this region to filter the message

      #endregion UserCode

      // Change the following if not propagating message to the 'Out' terminal
      outTerminal.Propagate(assembly);
    }
  }
}
```

Complete Template Filter Node in C#

```
using System;
using IBM.Broker.Plugin;

namespace FilterNodes {
  public class SimpleFilterNode : NBComputeNode {
    public override void Evaluate(NBMessageAssembly assembly) {
      NBOutputTerminal outTerminal = OutputTerminal("out");
      NBMessage inputMessage = assembly.Message;
      NBElement root = inputMessage.RootElement;

      #region UserCode
      // Add user code in this region to filter the message
      #endregion UserCode

      outTerminal.Propagate(assembly);
    }
  }
}
```

The Visual Studio Object Browser

- **Use the Object browser to view the Broker .NET APIs**

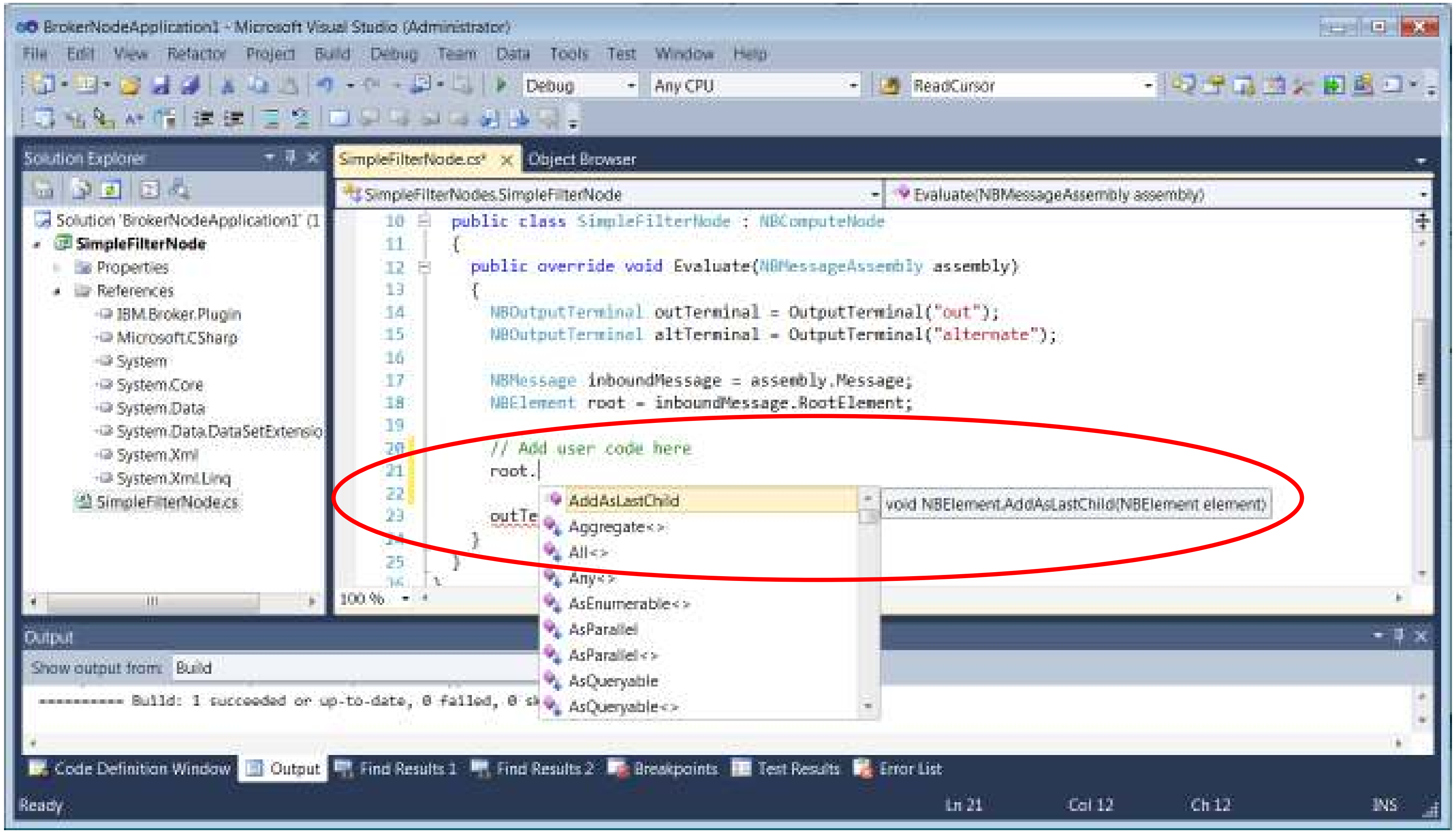


```
IBM.Broker.Plugin
  NBBitStreamControl
  NBBroker
  NBBrokerException
  NBCastException
  NBClrException
  NBCollectionMessage
  NBComputeNode
  NBConfigurationException
  NBConversionException
  NBDatabaseException
  NBElement
    Base Types
      IEnumerable<NBElement>
      Object
  NBElementConvert
  NBElementDateTimeOffsetType
  NBElementGenericType
  NBElementTimeSpanType
  NBElementValueType
  NBException
  NBExecutionGroup
  NBFatalException
  NBInputTerminal
  NBLog
  NBMessage
```

```
CreateFirstChildFromBitStream(byte[], string, string, string, string, int, int)
CreateFirstChildFromBitStream(byte[], string, string, string, string, int, int, IBM.Broker.Plugin.NBBitStreamCo
CreateFirstChildUsingNewParser(string)
CreateLastChild(int)
CreateLastChild(string)
CreateLastChild(string, string)
CreateLastChild(int, string, object)
CreateLastChild(string, string, object)
CreateLastChild(int, string, string, object)
CreateLastChildFromBitStream(byte[])
CreateLastChildFromBitStream(byte[], string)
CreateLastChildFromBitStream(byte[], string, int, int)
CreateLastChildFromBitStream(byte[], string, string, string, string, int, int)
CreateLastChildFromBitStream(byte[], string, string, string, string, int, int, IBM.Broker.Plugin.NBBitStreamCo
CreateLastChildUsingNewParser(string)
Delete()
DeleteAllChildren()
Detach()
DetachAllChildren()
```

public IBM.Broker.Plugin.NBElement **CreateLastChild**(string *space*, string *name*)
Member of IBM.Broker.Plugin.NBElement

Visual Studio Content Assist for the Broker Plugin API



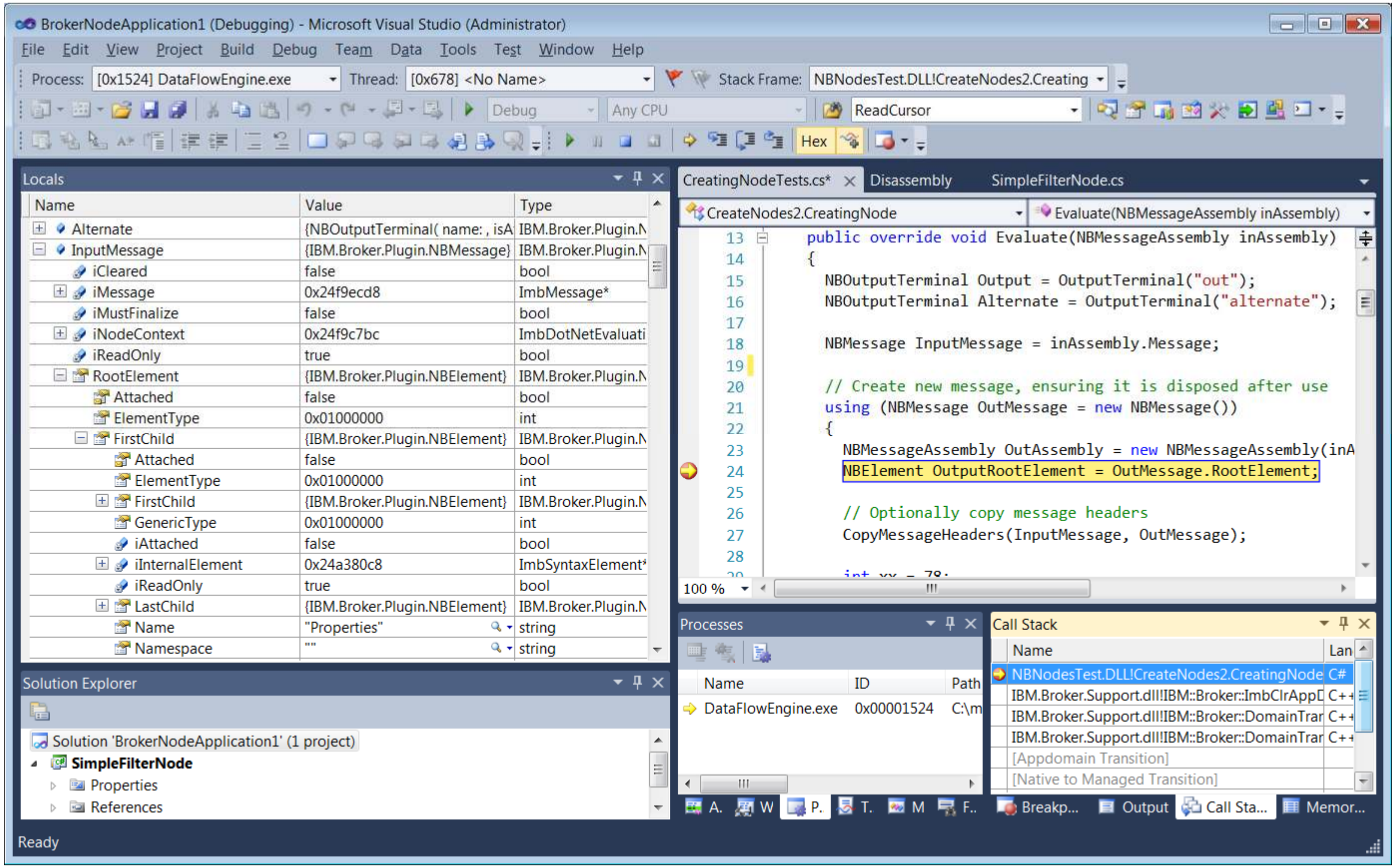
```
public class SimpleFilterNode : NBComputeNode
{
  public override void Evaluate(NBMessageAssembly assembly)
  {
    NBOutputTerminal outTerminal = OutputTerminal("out");
    NBOutputTerminal altTerminal = OutputTerminal("alternate");

    NBMessage inboundMessage = assembly.Message;
    NBElement root = inboundMessage.RootElement;

    // Add user code here
    root.

    outTe
  }
}
```

```
AddAsLastChild
Aggregate<>
All<>
Any<>
AsEnumerable<>
AsParallel
AsParallel<>
AsQueryable
AsQueryable<>
```

```
void NBElement.AddAsLastChild(NBElement element)
```

```
========== Build: 1 succeeded or up-to-date, 0 failed, 0 sk
```

Debug your .NET code with the Visual Studio Debugger


BrokerNodeApplication1 (Debugging) - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Test Window Help
Process: [0x1524] DataFlowEngine.exe
Thread: [0x678] <No Name>
Stack Frame: NBNodesTest.DLL!CreateNodes2.Creating
Debug
Any CPU
ReadCursor
Hex
Locals
CreatingNodeTests.cs*
Disassembly
SimpleFilterNode.cs
CreateNodes2.CreatingNode
Evaluate(NBMessageAssembly inAssembly)
100 %
Processes
Call Stack
Solution Explorer
A. W P. T. M F.
Breakp... Output Call Sta... Memor...
Ready

Locals

Name	Value	Type
Alternate	{NBOutputTerminal(name: , isA	IBM.Broker.Plugin.N
InputMessage	{IBM.Broker.Plugin.NBMessage}	IBM.Broker.Plugin.N
iCleared	false	bool
iMessage	0x24f9ecd8	ImbMessage*
iMustFinalize	false	bool
iNodeContext	0x24f9c7bc	ImbDotNetEvaluati
iReadOnly	true	bool
RootElement	{IBM.Broker.Plugin.NBElement}	IBM.Broker.Plugin.N
Attached	false	bool
ElementType	0x01000000	int
FirstChild	{IBM.Broker.Plugin.NBElement}	IBM.Broker.Plugin.N
Attached	false	bool
ElementType	0x01000000	int
FirstChild	{IBM.Broker.Plugin.NBElement}	IBM.Broker.Plugin.N
GenericType	0x01000000	int
iAttached	false	bool
iInternalElement	0x24a380c8	ImbSyntaxElement*
iReadOnly	true	bool
LastChild	{IBM.Broker.Plugin.NBElement}	IBM.Broker.Plugin.N
Name	"Properties"	string
Namespace	""	string

```
public override void Evaluate(NBMessageAssembly inAssembly)
{
  NBOutputTerminal Output = OutputTerminal("out");
  NBOutputTerminal Alternate = OutputTerminal("alternate");

  NBMessage InputMessage = inAssembly.Message;

  // Create new message, ensuring it is disposed after use
  using (NBMessage OutMessage = new NBMessage())
  {
    NBMessageAssembly OutAssembly = new NBMessageAssembly(inA
    NBElement OutputRootElement = OutMessage.RootElement;

    // Optionally copy message headers
    CopyMessageHeaders(InputMessage, OutMessage);

    int xx = 78;
```

Solution Explorer

- Solution 'BrokerNodeApplication1' (1 project)
  - **SimpleFilterNode**
    - Properties
    - References

Processes

Name	ID	Path
DataFlowEngine.exe	0x00001524	C:\m

Call Stack

Name	Lan
NBNodesTest.DLL!CreateNodes2.CreatingNode	C#
IBM.Broker.Support.dll!IBM::Broker::ImbClrAppD	C++
IBM.Broker.Support.dll!IBM::Broker::DomainTrar	C++
IBM.Broker.Support.dll!IBM::Broker::DomainTrar	C++
[Appdomain Transition]	
[Native to Managed Transition]	

Programming the Broker with the .NET APIs

- **The API is designed to look and feel like a standard .NET API**
  - Follows the Microsoft "Framework Design Guidelines"
  - For example, uses properties where appropriate, follows naming conventions etc
- **Is designed to be usable by as many .NET languages as possible**
  - Plugin assembly is marked as 'CLSCompliant'.
  - CLS guidelines followed
  - Where facilities that are not CLS compliant are used, alternatives are offered
    - E.g. Alternatives for explicit datatype casting
- **Scalar values and Nullable value types supported throughout**
  - All broker types are "Nullable"
  - Conversions to/from Nullable equivalents available
- **Simple but Powerful**
  - Utility methods provided for common tasks, such as throwing user exceptions,
  - creating XMLDecl's etc `[<?xml version="1.0" encoding="UTF-8" standalone="yes"?>]`

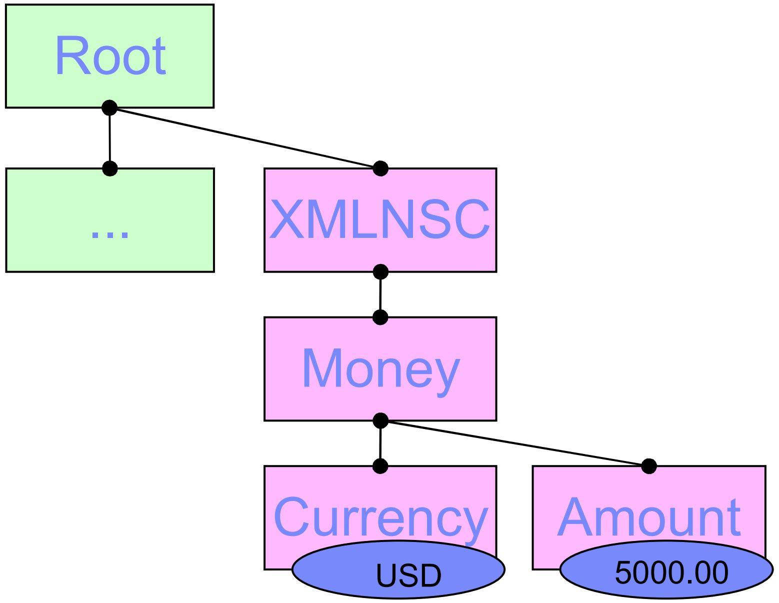
```
NBParsers.XMLNSC.CreateXmlDeclaration( element, "1.0", "utf-8", "yes" );
```

Message Broker Tree : Low Level Access

- **Each element in the tree has several properties**
  - `Parent, LastChild, FirstChild, PreviousSibling, NextSibling`
- **They can be chained to access a specific element**
  - `NBElement amount = root.LastChild.LastChild.LastChild;`
- **These properties return either an Element or NULL**
  - `NullReferenceException` possible if you dereference NULL

XML Message

```
<Money>
  <Currency>USD</Currency>
  <Amount>5000.00</Amount>
</Money>
```



Message Broker Tree : Access by Name

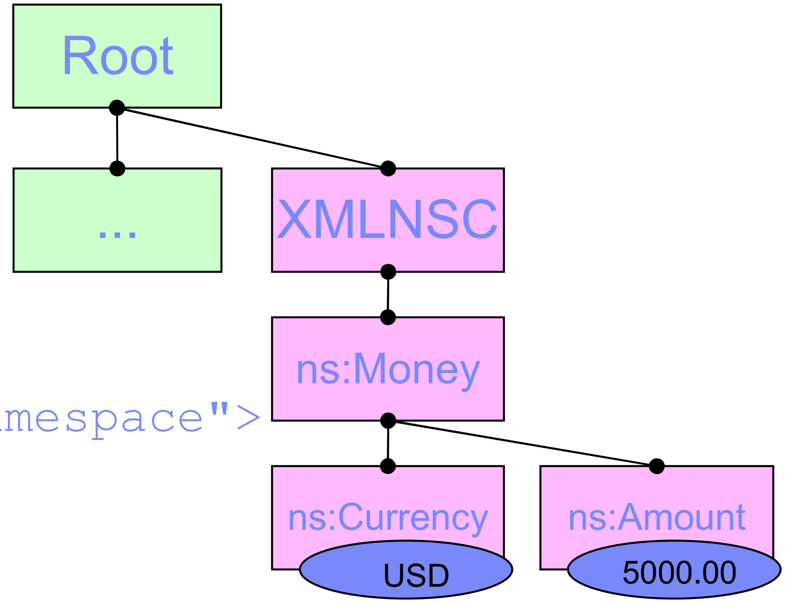
- **Navigation by name is also possible**

```
NBElement amount = root["XMLNSC"]["Money"]["Amount"];
```

- **Navigation by name and namespace as well**

```
string ns = new string("http://my.long.namespace");
NBElement amount = root["XMLNSC"][ns, "Money"][ns, "Amount"];
```

- **Still have to be careful:**
  – `NullReferenceException` still possible



XML Message

```
<ns:Money xmlns:ns="http://my.long.namespace">
  <ns:Currency>USD</ns:Currency>
  <ns:Amount>5000.00</ns:Amount>
</ns:Money>
```

Integration breadth: Call COM and .NET applications within Broker

- **Allows Broker to integrate with COM and .NET applications**
  – Access existing COM and .NET applications that run in a .NET 4 environment
- **.NET makes it easy to call other .NET applications and components**

```
 using SharePointClient = Microsoft.SharePoint.Client;
//Update SharePoint with details from Message
private void UpdateSharepoint(NBElement fileInfo)
{
  string fileSource = (string)fileInfo["Location"];
  string fileName = "/sites/pp/Documents/" + (string)fileInfo["Name"];
  ClientContext context = new ClientContext("http://avoca2008");
  using (FileStream fs = new FileStream(fileSource, FileMode.Open))
  {
    SharePointClient.File.SaveBinaryDirect(context, fileName, fs, true);
  }
}
```

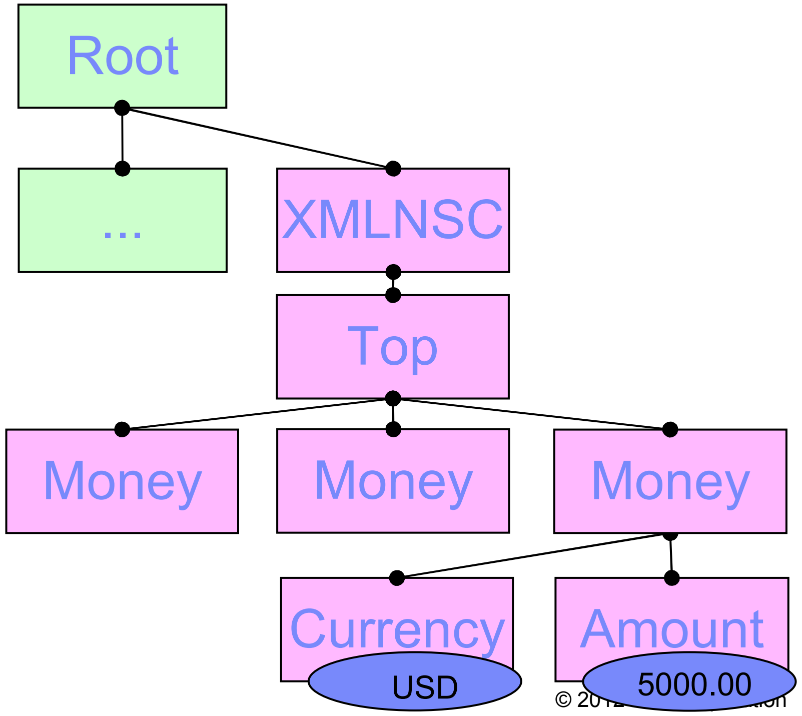
Message Broker Tree : Using LINQ

- **Use LINQ queries to access the Broker Tree**

```
NBElement x = InputMessage.RootElement["XMLNSC"]["Top"];
var list = x.Where(t => t.Name == "Money" && (String)t["Currency"] == "USD");
foreach (NBElement element in list) {
 //Process each element in turn
}
```

XML Message

```
<Top>
  <Money>
    <Currency>GBP</Currency>
    <Amount>1000.00</Amount>
  </Money>
  <Money>
    <Currency>USD</Currency>
    <Rate>1.4</Rate>
    <Amount>5000.00</Amount>
  </Money>
  <Money>
    <Currency>USD</Currency>
    <Amount>2000.00</Amount>
  </Money>
</Top>
```



Handle Exceptions with Ease

- **Broker exceptions are turned into NBExceptions so they can be caught in .NET code**
  - `NBException`
    - `NBRecoverableException`

      `NBUserException`

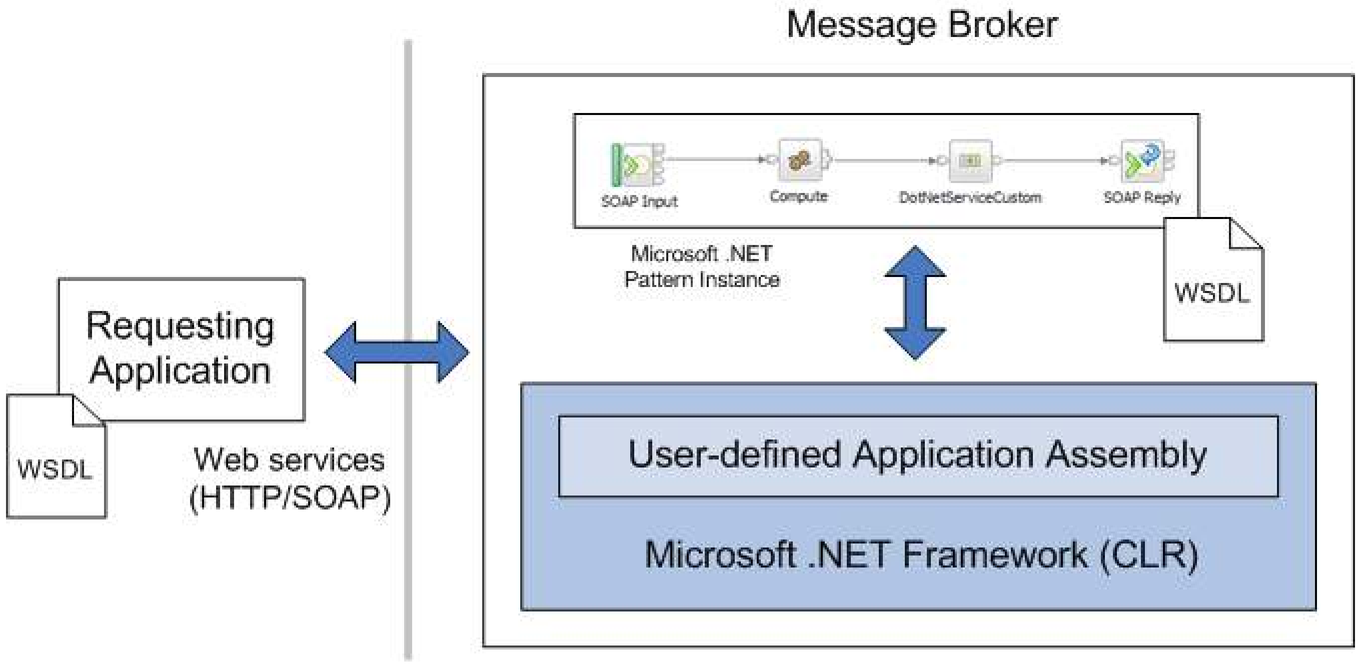
      `NBXxxException`

- **NBExceptions are turned into Broker exceptions if thrown out of the user .NET code**
  - You can “leave” your .NET code with an exception if necessary.
  - You can catch the exception by using in a Try/Catch node or wiring a Catch terminal.
  - NBRecoverable exceptions can be caught in an ESQL handler, with a specified SQLCode and SQLState

- **.NET exceptions are turned into Broker exceptions if thrown out of the user .NET code**
  - You can catch the exception by using in a Try/Catch node or wiring a Catch terminal

Expose .NET Methods as Services – a Pattern Based Approach

- **New Service Facade Pattern**
  - “Microsoft .NET Request-Response”
- **Easily expose .NET Methods as Web Services**
  - “drag-drop” of assembly onto “pattern wizard”
- **Pattern Flow is ready to deploy**
  - Auto Generated WSDL and ESQL



ESQL Calling .NET : Declaring the Method

- **ESQL Function and Procedure syntax extended to allow .NET method calls**

```
CREATE PROCEDURE DotNetMethod (
    IN x INTEGER NOT NULL,
     OUT y INTEGER NOT NULL,
     INOUT z INTEGER NOT NULL)
 RETURNS INTEGER NOT NULL
 LANGUAGE .NET
 EXTERNAL NAME "MyNamespace.MyClass.MyDotNetMethod"
 ASSEMBLY "D:\WMB\Assemblies\MyApplication.dll"
 APPDOMAIN "MyAppDomain";
```

- **Drag Drop an Assembly onto a Compute node to auto generate matching signatures**
  – Choose the app domain (optional)

ESQL Calling .NET : Making the Call

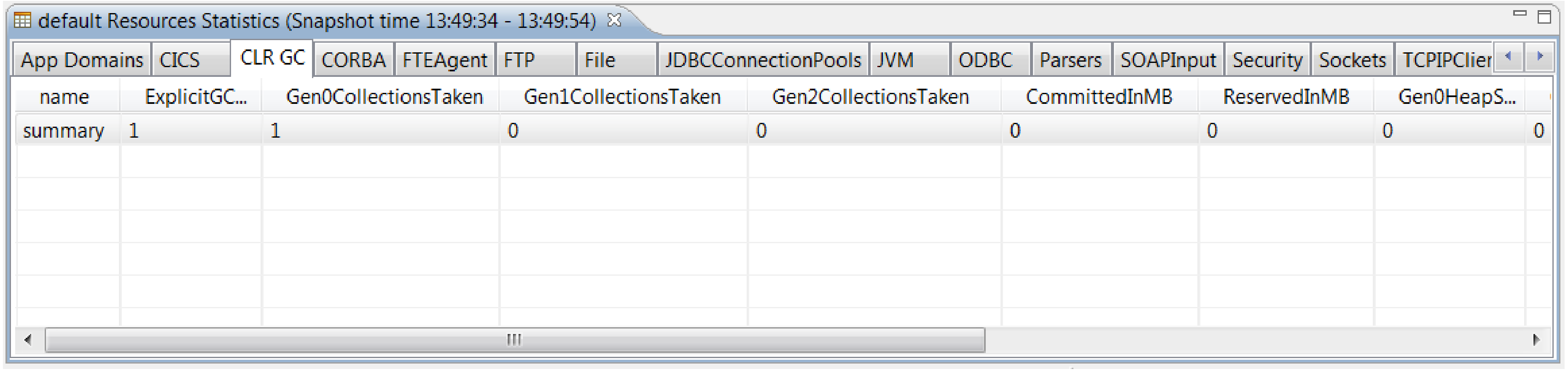
- **There is nothing .NET specific when the routine is invoked.**

```
DECLARE input INTEGER 42;

DECLARE output INTEGER ;

DECLARE inAndOut INTEGER 45;

DECLARE result INTEGER ;

CALL DotNetMethod(input, output, inAndOut) INTO result;
```

- **Call static methods that have compatible types**
  – Comprehensive type mapping table

Hosting the CLR in the Execution Group

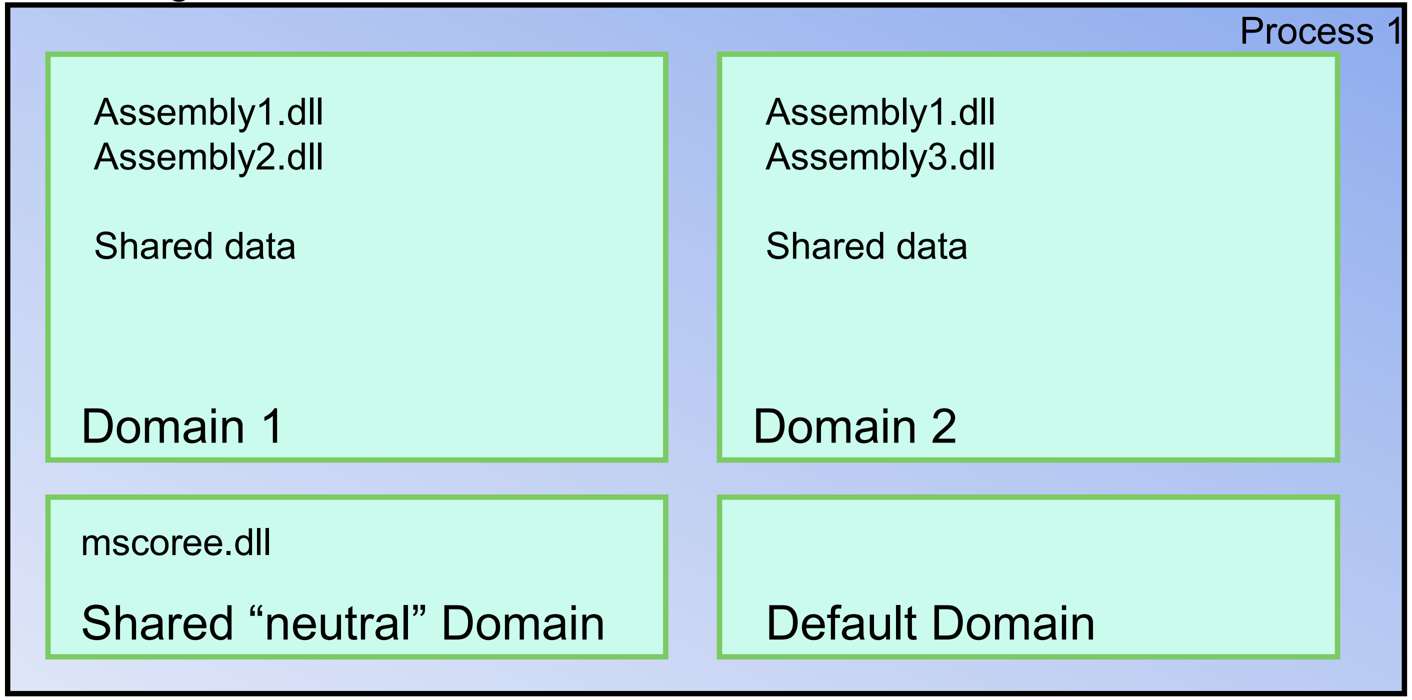
- **A CLR is hosted inside each execution group on Windows**
  - V4.0 CLR. If the .NET code is supported running in the .NET 4 CLR then you can run it in Broker.
- **The CLR is started automatically if found when the E.G. starts**
  - Not an error if it is not found
  - But .NET code cannot run without it.
- **CLR statistics available to show memory usage, Garbage collections etc.**


default Resources Statistics (Snapshot time 13:49:34 - 13:49:54)

name	ExplicitGC...	Gen0CollectionsTaken	Gen1CollectionsTaken	Gen2CollectionsTaken	CommittedInMB	ReservedInMB	Gen0HeapS...	
summary	1	1	0	0	0	0	0	0

Application Domains in .NET : 1

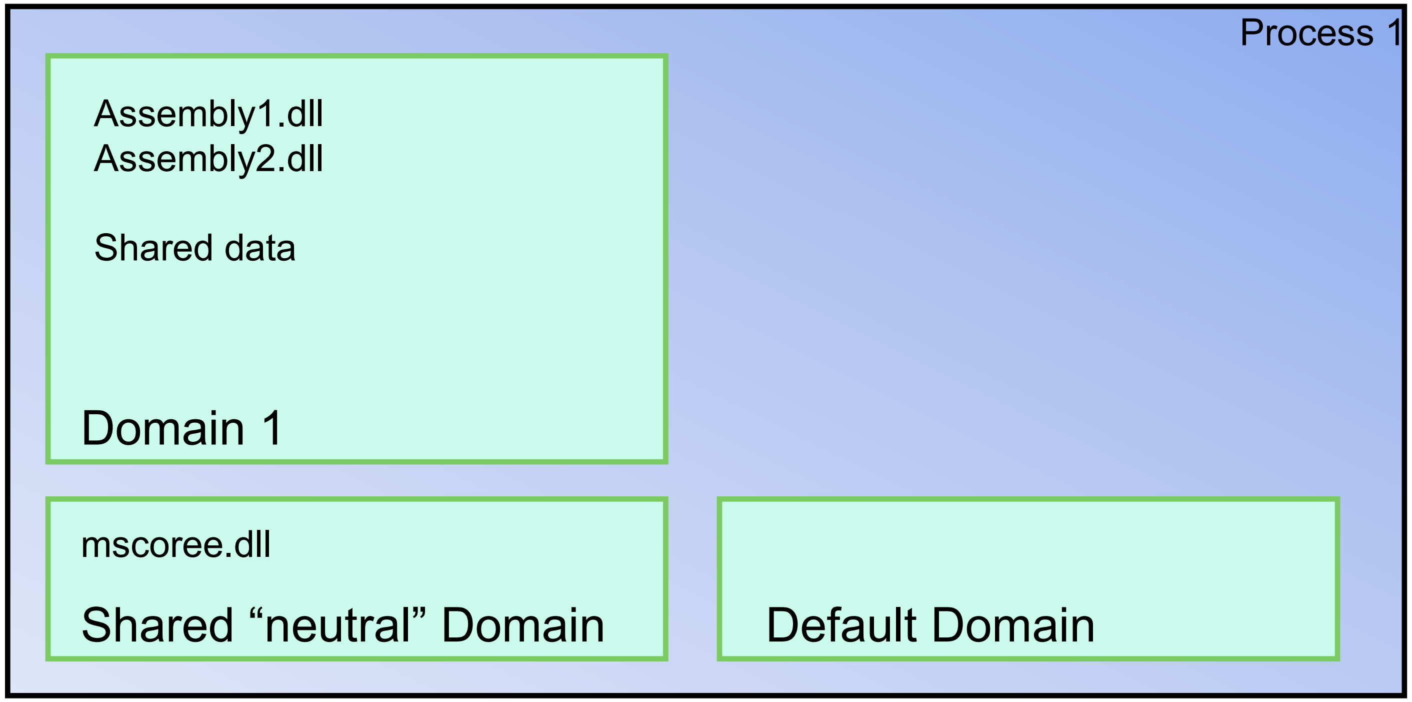
- **All code executed in the CLR runs inside an App Domain**
  - A default App Domain created by the CLR
  - Extra App Domains can be created by user code
  - Code can be shared between App Domains if it is loaded “domain neutral”
- **An App Domain provides a scoping point**
  - A sub-process unit of isolation for managed code
    - For unload / reload of code
    - For sharing of data



Application Domains in .NET : 2

- **Only whole App Domains can be unloaded**
  - Not possible to unload a single Assembly
  - Not possible to unload “domain neutral” assemblies

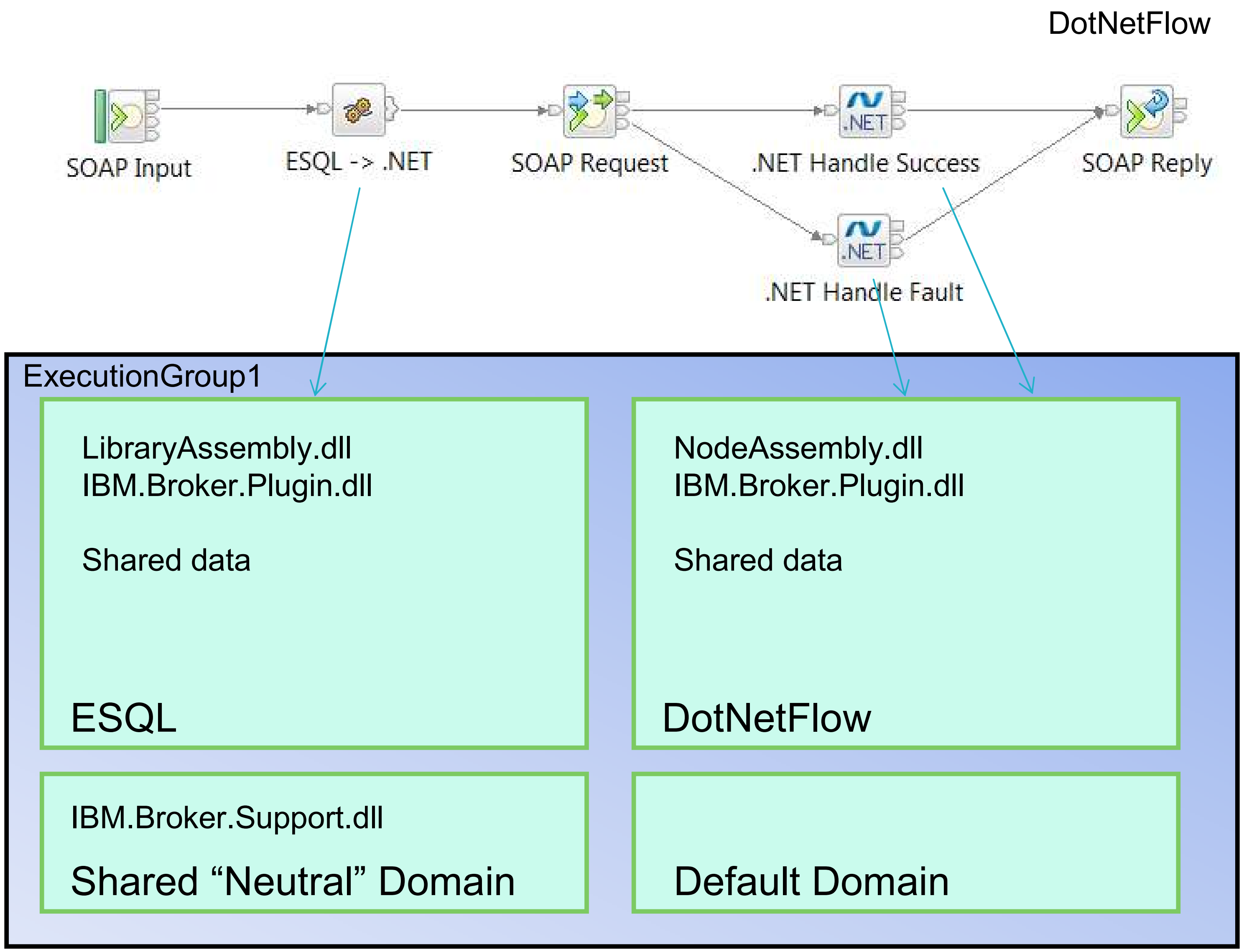
- **Sharing Data between assemblies**
  - Code sharing an app domain can share state
  - Cross App Domain data sharing requires “remoting” of the data to be shared



Application Domains in the Broker : 1

- **App Domains provide the unit of scoping for all .NET code**
- **App Domains have several properties**
  - A name and a “base directory” where the code in that AppDomain lives
  - An optional configuration file
    - Provides extra information to code running in that domain
- **App Domains can be created by name**
  - If unnamed a domain will be named after the broker Application the flow is a part of
- **App Domains can also be created by a Configurable Service**
  - Specify App Domain properties
  - **App Domains allow the “hot swap” of a .NET assembly with “Shadow Copy”**
    - Flow will dynamically reload associated App Domains if the code is changed.
    - Speed up development time
  - **App Domains provide statistics about their memory usage**

Application Domains in the Broker : 2



CLR inside the Execution Group

AppDomain Shadow Copy - “Hot Swap” Your Assemblies

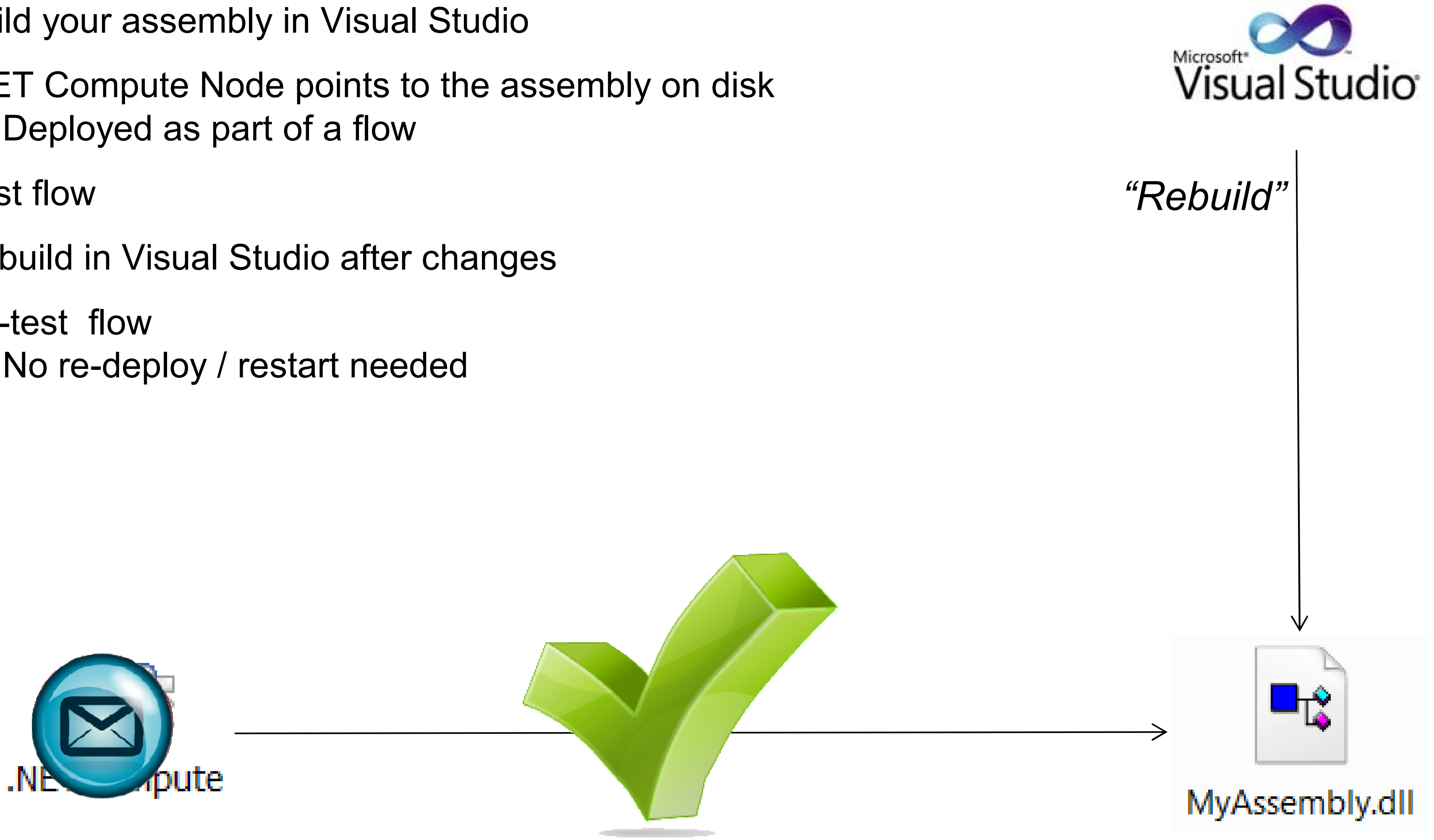
1: Build your assembly in Visual Studio

2: .NET Compute Node points to the assembly on disk
- Deployed as part of a flow

3: Test flow

4: Rebuild in Visual Studio after changes

5: Re-test flow
- No re-deploy / restart needed



CLR Native Datatype Mapping

Broker Type 1	CLR Type 1	Broker Type 2	CLR Type 2
Integer Not Null **Integer**	Int64 Nullable<Int64>	**Date Not Null** **Date**	DateTimeOffset Nullable<DateTimeOffset>
Int Not Null **Int**	Int32 Nullable<Int32>	**Time Not Null** **Time**	TimeSpan Nullable<TimeSpan>
Decimal Not Null **Decimal**	Decimal Nullable<Decimal>	**Timestamp Not Null** **Timestamp**	DateTimeOffset Nullable<DateTimeOffset>
Float Not Null **Float**	Double Nullable<Double>	**Gmttime Not Null** **Gmttime**	TimeSpan Nullable<TimeSpan>
Bit Not Null **Bit**	BitArray “”	**Gmttimestamp Not Null** **Gmttimestamp**	DateTime Nullable<DateTime>
Blob Not Null **Blob**	Byte[] “”	**Interval Not Null *** **Interval ***	TimeSpan Nullable<TimeSpan>
Character Not Null **Character**	String “”	**Interval YEAR – MONTH**	Not Supported
Char Not Null **Char**	Char Nullable<Char>	**Reference Not Null** **Reference**	NBElement “”
Boolean Not Null **Boolean**	Boolean Nullable<Boolean>		

* DAY – HOUR – MINUTE – SECOND

C# Datatype Mapping 1

Broker Type	C# Type (In)	C# Type (Out)	C# Type (Inout)
Integer Not Null **Integer**	long long?	out long out long?	ref long ref long?
Int Not Null **Int**	int int?	out int out int?	ref int ref int?
Decimal Not Null **Decimal**	decimal decimal?	out decimal out decimal?	ref decimal ref decimal?
Float Not Null **Float**	double double?	out double out double?	ref double ref double?
Bit Not Null **Bit**	BitArray “”	out BitArray “”	ref BitArray “”
Blob Not Null **Blob**	Byte[] “”	out Byte[] “”	ref Byte[] “”
Character Not Null **Character**	string “”	out string “”	ref string “”
Char Not Null **Char**	char char?	out char out char?	ref char ref char?
Boolean Not Null **Boolean**	bool bool?	out bool out bool?	ref bool ref bool?

VB Datatype Mapping 1

Broker Type	VB Type (In)	VB Type (Out)	VB Type (Inout)
Integer Not Null **Integer**	ByVal Long ByVal Long?	<Out()> ByRef Long <Out()> ByRef Long?	ByRef Long ByRef Long?
Int Not Null **Int**	ByVal Integer ByVal Integer?	<Out()> ByRef Integer <Out()> ByRef Integer?	ByRef Integer ByRef Integer?
Decimal Not Null **Decimal**	ByVal Decimal ByVal Decimal?	<Out()> ByRef Decimal <Out()> ByRef Decimal?	ByRef Decimal ByRef Decimal?
Float Not Null **Float**	ByVal Double ByVal Double?	<Out()> ByRef Double <Out()> ByRef Double?	ByRef Double ByRef Double?
Bit Not Null **Bit**	ByVal BitArray “”	<Out()> ByRef BitArray “”	ByRef BitArray “”
Blob Not Null **Blob**	ByVal Byte() “”	<Out()> ByRef Byte() “”	ByRef Byte() “”
Character Not Null **Character**	ByVal String “”	<Out()> ByRef String “”	ByRef String “”
Char Not Null **Char**	ByVal Char ByVal Char?	<Out()> ByRef Char <Out()> ByRef Char?	ByRef Char ByRef Char?
Boolean Not Null **Boolean**	ByVal Boolean ByVal Boolean?	<Out()> ByRef Boolean <Out()> ByRef Boolean?	ByRef Boolean ByRef Boolean?

Summary

- Very tight .NET Integration
  - CLR v4 hosted inside the Execution Group
  - .NET code executed natively inside the broker
  - Use any CLR language to create your nodes
  - Integrated App Domain support

- Large API to provide access to message broker facilities
  - Navigation
  - Element creation
  - Exception handling
    - From .NET exception to ExceptionList
    - From ExceptionList to .NET exception
    - Catch exceptions in ESQL

- Visual Studio Integration
  - Launch Visual Studio from Eclipse
  - Plugins to provide fast node creation
  - Content assist for easy access to the API
  - Debug your nodes using Visual Studio

Questions?

धन्यवाद

Hindi

多謝

Traditional Chinese

Grazie

Italian

ขอบคุณ

Thai

Спасибо

Russian

Obrigado

Brazilian Portuguese

Merci

French

Gracias

Spanish

شكراً

Arabic

多谢

Simplified Chinese

Danke

German

நன்றி

Tamil

ありがとうございました

Japanese

감사합니다

Copyright Information