



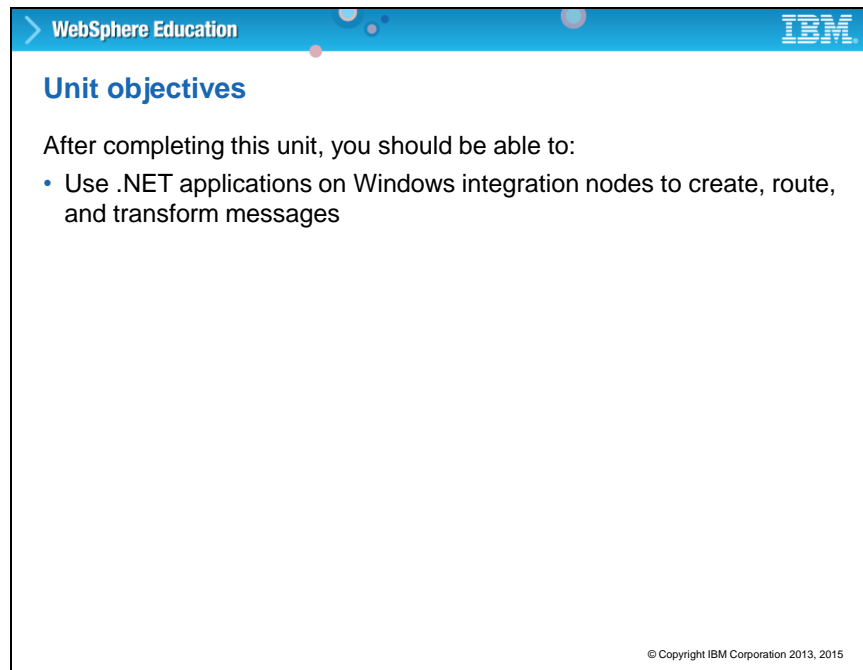
Slide 1

> WebSphere Education 

Transforming data with Microsoft .NET



© Copyright IBM Corporation 2013, 2015
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The slide features a blue header bar with a right-pointing arrow, the text 'WebSphere Education', and the IBM logo. The main content area is white with a black border. It contains the title 'Unit objectives' in blue, followed by a sentence 'After completing this unit, you should be able to:' and a single bullet point. A small copyright notice is at the bottom right.

> WebSphere Education IBM

Unit objectives

After completing this unit, you should be able to:

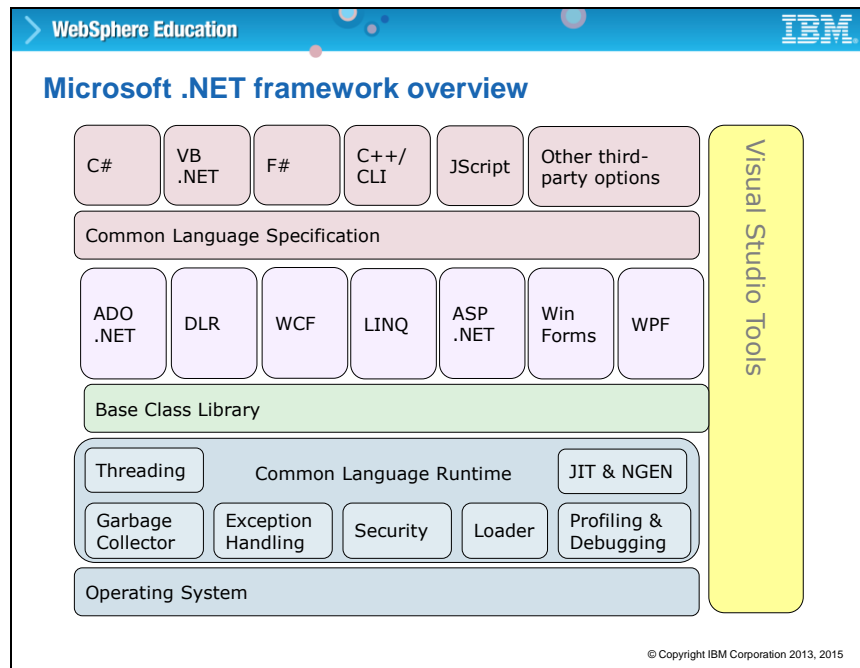
- Use .NET applications on Windows integration nodes to create, route, and transform messages

© Copyright IBM Corporation 2013, 2015

Unit objectives

You can use .NET applications on Windows integration nodes to create, route, and transform data by using the .NETCompute and .NETInput nodes. This unit describes the .NETCompute node and the .NETInput node.

After completing this unit, you should be able to use .NET applications on Windows integration nodes to create, route, and transform messages.

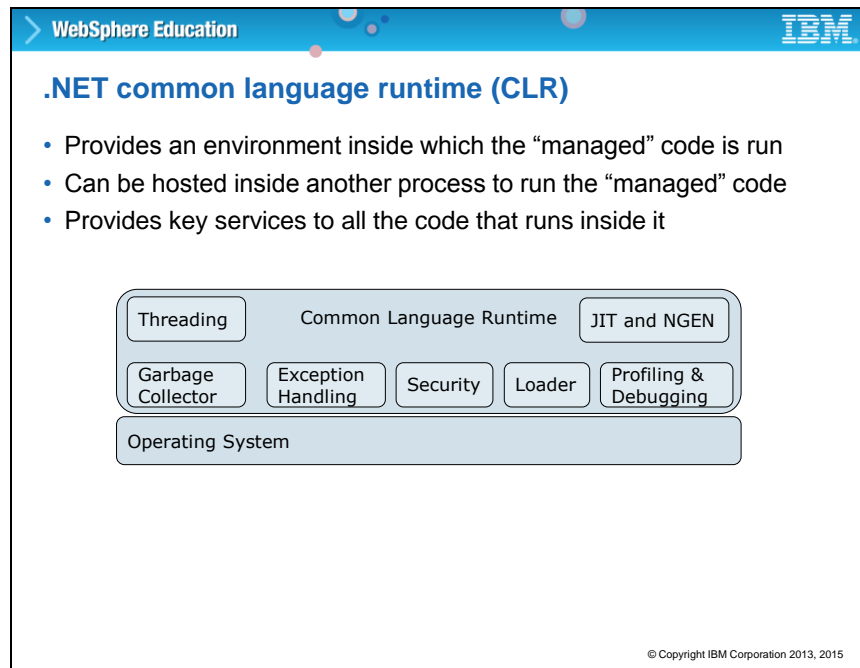


Microsoft .NET framework overview

The Microsoft .NET framework is a software framework that runs primarily on Microsoft Windows. It includes a large library and provides language interoperability across several programming languages. Programs that are written for the .NET framework are run in a software environment that is known as the common language runtime (CLR). CLR is an application virtual machine that provides services such as security, memory management, and exception handling.

The base class library of the .NET framework provides the user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their own source code with the .NET framework and other libraries.

Visual Studio is the Microsoft integrated development environment for .NET.



.NET common language runtime (CLR)

The .NET CLR runs the code and provides services that make the development process easier. Compilers and tools provide the CLR functions so that you can write code that uses this managed runtime environment.

CLR provides key services such as cross-language integration, cross-language exception handling, enhanced security, deployment support, a simplified model for component interaction, and debugging and profiling services.

For more information about the .NET framework, see the Microsoft Developer Network library.

WebSphere Education
IBM

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)
- CIL code is in a `.dll` or `.exe` and is called an *assembly*
 - Assembly is loaded into the CLR to be run
 - Code is created just before it is run
- At run time, the CLR does not care what the source language was

```

graph TD
    C[C#] --> CLS[Common Language Specification]
    VB[VB .NET] --> CLS
    F[F#] --> CLS
    C[C++/CLI] --> CLS
    J[JScript] --> CLS
    O[Other third-party options] --> CLS
  
```

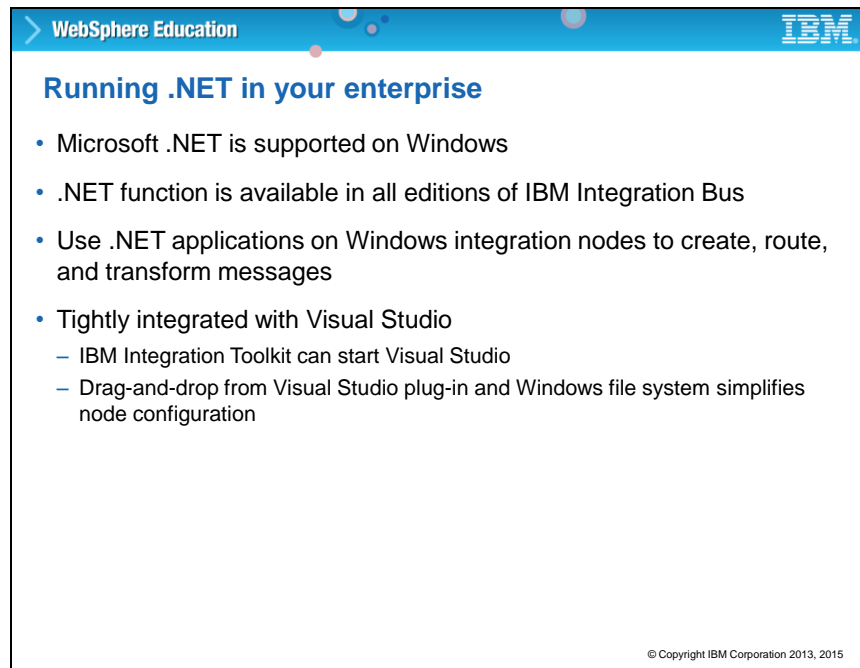
© Copyright IBM Corporation 2013, 2015

From source code to byte code


Common intermediate language (CIL) is the lowest-level human-readable programming language that the common language infrastructure (CLI) specification defines. Languages that target a CLI-compatible runtime environment compile to CIL, which is assembled into an object code that has a bytecode-style format. CIL is an object-oriented assembly language, and is entirely stack-based. Its bytecode is converted into operating system code or run by a virtual machine.

A common language specification (CLS) is a document that defines how computer programs can be turned into bytecode. The .NET Framework uses a CLS so that it can interact with other objects regardless of the language.

The goal is to generate a `.dll` or `.exe` file that can be used in a message flow. In Integration Bus, the .NETCompute message processing node uses any language that complies to create or modify the message, such as C#, Visual Basic (VB), F#, and C++.



The slide is titled "Running .NET in your enterprise" and is part of a WebSphere Education presentation. It features a blue header with the "WebSphere Education" text and the IBM logo. The main content is a bulleted list of features related to .NET support in IBM Integration Bus. The footer contains a copyright notice for IBM Corporation from 2013 to 2015.

WebSphere Education 

Running .NET in your enterprise

- Microsoft .NET is supported on Windows
- .NET function is available in all editions of IBM Integration Bus
- Use .NET applications on Windows integration nodes to create, route, and transform messages
- Tightly integrated with Visual Studio
 - IBM Integration Toolkit can start Visual Studio
 - Drag-and-drop from Visual Studio plug-in and Windows file system simplifies node configuration

© Copyright IBM Corporation 2013, 2015

Running .NET in your enterprise


The Microsoft .NET Framework runs on Windows only.

You can use .NET applications on Windows integration nodes to create, route, and transform messages by using the .NETCompute and .NETInput nodes.

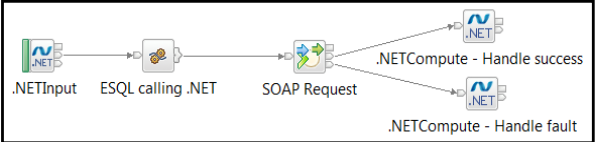
- With the .NETInput node, you can position .NET connectivity at the beginning of the message flow.
- With the .NETCompute node, you can position .NET connectivity in the middle of the message flow as a part of your core infrastructure.

Your message flows can call .NET assemblies. You write and maintain your .NET code in Microsoft Visual Studio. You develop your message flows in the Integration Toolkit. It is suggested that you keep the assemblies in the Visual Studio framework; do not import them into the Integration Toolkit.

You can also drag an existing .NET assembly or .NETCompute node onto the canvas, and then follow the prompts to create this association. The assembly must contain an implementation of a .NETCompute node.

WebSphere Education 

Integrating .NET with IBM Integration Bus



The diagram illustrates a message flow for integrating .NET with IBM Integration Bus. It starts with a .NETInput node, which connects to an ESQL calling .NET node. This node then connects to a SOAP Request node. The SOAP Request node has two outgoing paths: one to a .NETCompute - Handle success node and another to a .NETCompute - Handle fault node. Both .NETCompute nodes are represented by icons with the .NET logo.

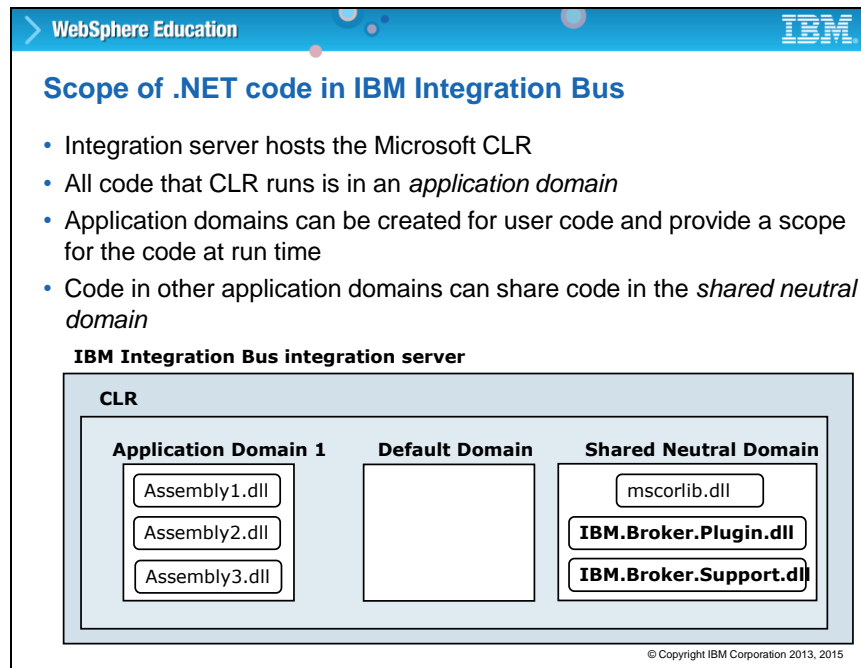
- Use .NETInput node to get data from disparate sources such as MSMQ, IBM MQ, file system, or database
- Use .NETCompute node to construct output messages and interact with Microsoft .NET Framework or Component Object Model (COM) applications
- Use the CREATE FUNCTION or CREATE PROCEDURE statements in a node that supports ESQL to call .NET code

© Copyright IBM Corporation 2013, 2015

Integrating .NET with IBM Integration Bus

In Integration Bus, you can call .NET assemblies in your message flows from the .NETCompute node, .NETInput node, or from an ESQL procedure.

You can also include .NET assemblies as dependencies of a library. These assemblies run in a .NET application domain and can be packaged in a BAR file. Application domains are shown in the Toolkit Application Development view as peers of applications and libraries.



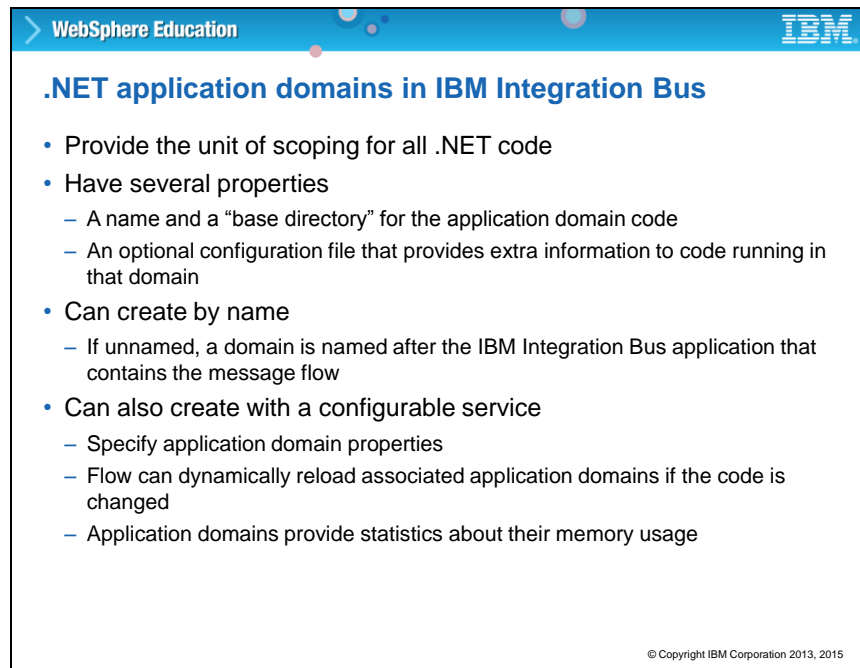
Scope of .NET code in IBM Integration Bus

At run time, an integration server hosts the Microsoft CLR.

You configure a .NET node with a .NET assembly that contains the code of the node. The code consists of a class that is derived from the abstract class that is provided in the IBM.Broker.Plugin.dll assembly.

The .NET assembly is run inside a .NET application domain. A .NET application domain is a runtime container for .NET assemblies and the associated resources that the .NET code in the message flows uses.

If you do not explicitly create a .NET application domain, and do not set a value for the **AppDomain** property on the .NET node, .NET assemblies can also be contained in an implicit .NET application domain.



The slide is titled ".NET application domains in IBM Integration Bus" and is part of a WebSphere Education presentation. It contains a bulleted list of features and properties of .NET application domains. The list includes: providing a unit of scoping for all .NET code, having several properties (name, base directory, optional configuration file), creating by name (named after the application if unnamed), and creating with a configurable service (specifying properties, dynamic reloading, and memory usage statistics). A copyright notice for IBM Corporation 2013, 2015 is at the bottom right.

- Provide the unit of scoping for all .NET code
- Have several properties
 - A name and a “base directory” for the application domain code
 - An optional configuration file that provides extra information to code running in that domain
- Can create by name
 - If unnamed, a domain is named after the IBM Integration Bus application that contains the message flow
- Can also create with a configurable service
 - Specify application domain properties
 - Flow can dynamically reload associated application domains if the code is changed
 - Application domains provide statistics about their memory usage

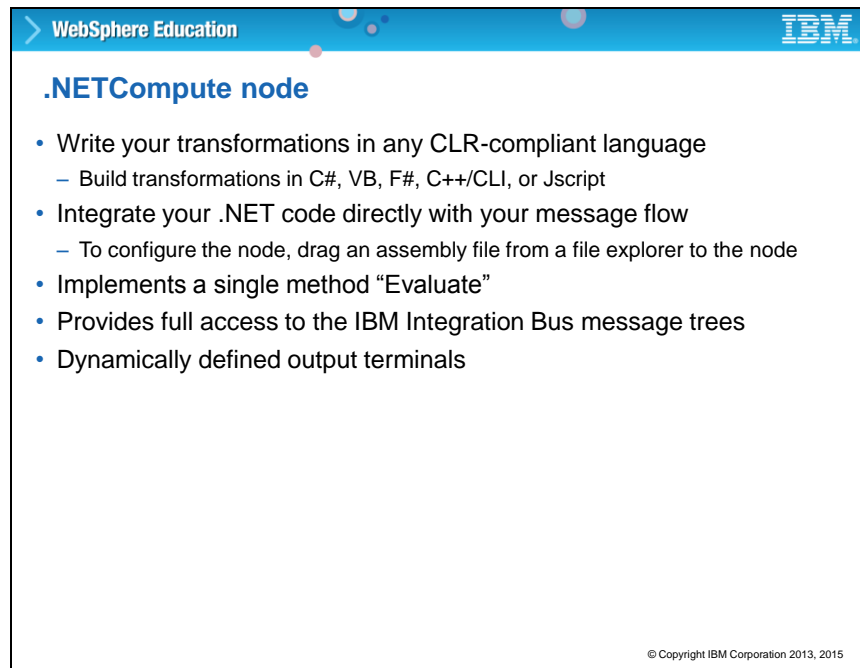
© Copyright IBM Corporation 2013, 2015

.NET application domains in IBM Integration Bus

The .NET assembly is run inside a .NET application domain. A named .NET application domain can be associated with Integration Bus applications. Assemblies, or associated resources, contained in the .NET application domain are deployed with the referencing message flow or application.

Integration Bus has an XML configuration file that is used to configure the domain. It is used to set container and application-specific properties and is loaded when the application domain is initialized. The value can specify a base name or a fully qualified path. The integration node looks up a base name in the directory that is specified in the **Application Base** property.

The DotNetAppDomain configurable service can be used to specify application domain properties.



The slide is titled ".NETCompute node" and is part of a WebSphere Education presentation. It features a blue header with the text "WebSphere Education" and the IBM logo. The main content is a bulleted list of features for the .NETCompute node. At the bottom right, there is a small copyright notice: "© Copyright IBM Corporation 2013, 2015".

- Write your transformations in any CLR-compliant language
 - Build transformations in C#, VB, F#, C++/CLI, or Jscript
- Integrate your .NET code directly with your message flow
 - To configure the node, drag an assembly file from a file explorer to the node
- Implements a single method "Evaluate"
- Provides full access to the IBM Integration Bus message trees
- Dynamically defined output terminals

© Copyright IBM Corporation 2013, 2015

.NETCompute node

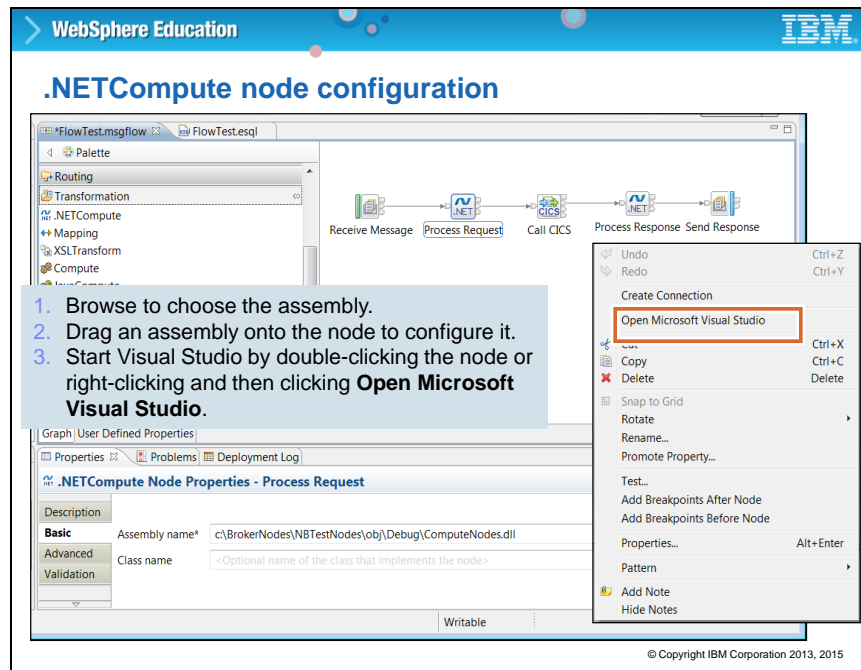
The .NETCompute node is similar to the JavaCompute node, in that you can use it to create, modify, or filter a message.

The .NETCompute node uses any CLR-compliant language to create or modify the message, such as C#, Visual Basic, F#, and C++/CLI.

The .NETCompute node class contains a method that must be overridden, and two optional methods that you can choose to override if necessary. You must always override the Evaluate method. If the template is used to generate the skeleton code, the Integration Bus template automatically implements the method.

You can customize the node to create an output message, or a number of messages, by using an input message or data from an external source.

The .NETCompute node has one output terminal but you can define more dynamic output terminals as required.



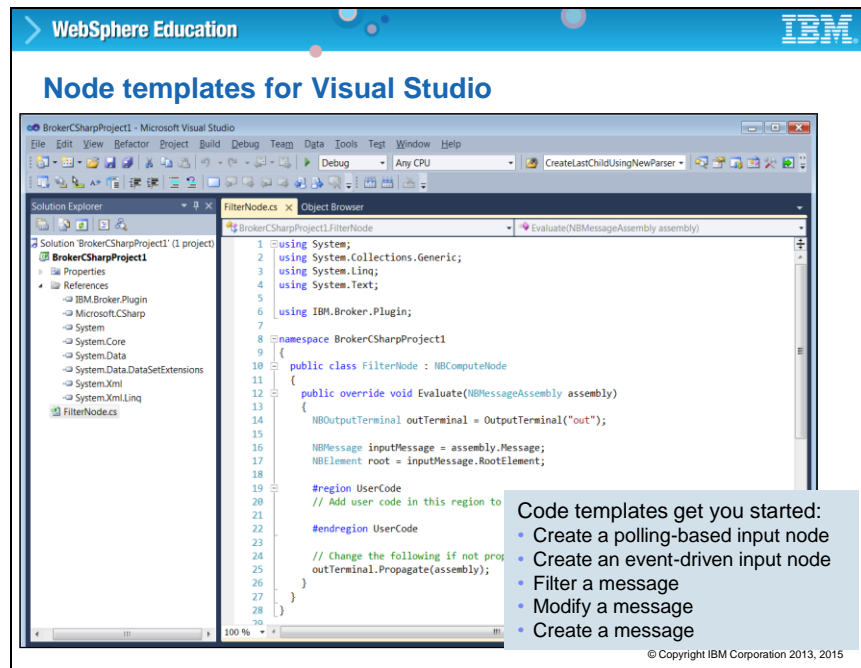
.NETCompute node configuration

You create a .NETCompute node by using both the Integration Toolkit and Microsoft Visual Studio. You can use any CLR-compliant language to code the .NETCompute node.

If you have a vendor-supplied application .dll file, you must set the .NETCompute node **Assembly name** property and other node properties. If you do not have a .dll, you can create one by using Microsoft Visual Studio. Visual Studio provides development templates for common message flow operations, such as creating a message, modifying a message, or filtering a message that is based on its content.

After you create the .NET assembly, add a .NETCompute node to the message flow in the Integration Toolkit. Drag the .NET assembly file from Windows Explorer onto the .NETCompute node, and the Integration Toolkit automatically sets several of the properties for you.

Microsoft Visual Studio must be installed on the same workstation where the Integration Toolkit is installed to start Visual Studio from the .NETCompute node.



Node templates for Visual Studio

Integration Bus can integrate with Microsoft Visual Studio 2010 or 2012 to create a .NET assembly that represents the node.

If the Integration Toolkit is installed after Microsoft Visual Studio, the Project templates with skeleton code are automatically installed ready for you to use.

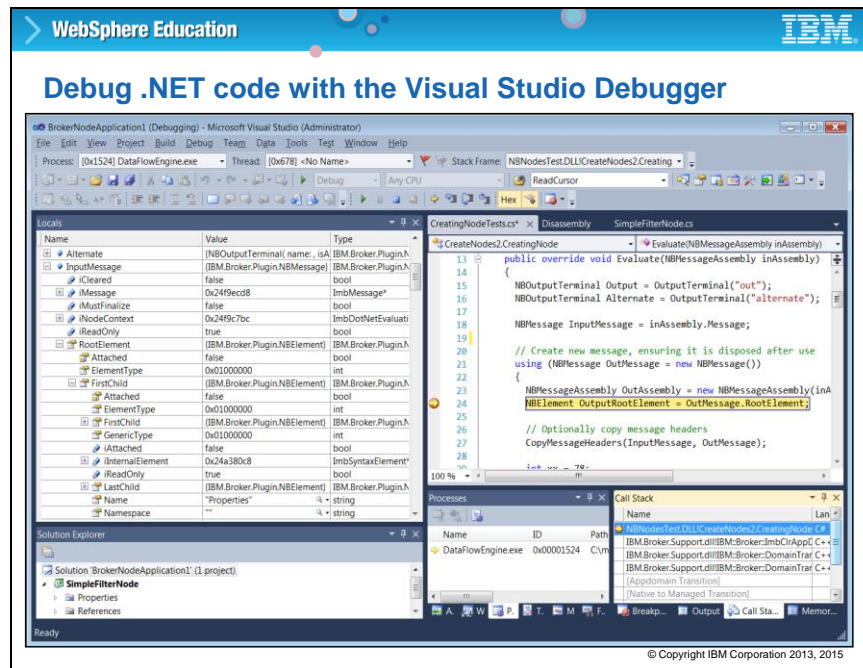
If the Integration Toolkit is installed first, you must manually install the Microsoft Visual Studio templates. This installation can be achieved by running the file IBM.Broker.DotNet.vsix and stepping through the wizard, accepting the license file as part of the process.

For the .NETCompute node, you can choose to:

- Filter a message.
- Modify a message.
- Create a message.

For the .NETInput node, you can choose to create a polling-based .NET input node or an event-driven .NET input node.

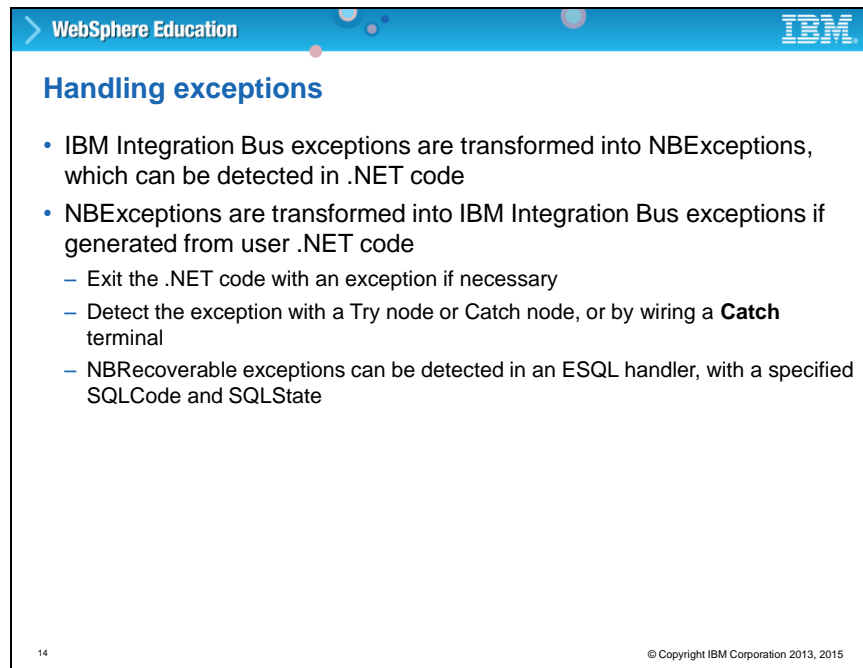
Slide 13




Debug .NET code with Visual Studio Debugger

You can use the Visual Studio Debugger to debug .NET code that is running inside the integration server.

First, put the .pdb file for the assembly that implements the node in the same directory as the assembly for the node. Then, attach Microsoft Visual Studio to the DataFlowEngine.exe process. Finally, set breakpoints in your code and examine variables when the breakpoint is reached.



The slide is titled "Handling exceptions" and is part of a WebSphere Education presentation. It contains a bulleted list of information regarding IBM Integration Bus exceptions and NBExceptions. The slide also includes a footer with the number 14 and a copyright notice for IBM Corporation.

WebSphere Education 

Handling exceptions

- IBM Integration Bus exceptions are transformed into NBExceptions, which can be detected in .NET code
- NBExceptions are transformed into IBM Integration Bus exceptions if generated from user .NET code
 - Exit the .NET code with an exception if necessary
 - Detect the exception with a Try node or Catch node, or by wiring a **Catch** terminal
 - NBRecoverable exceptions can be detected in an ESQL handler, with a specified SQLCode and SQLState

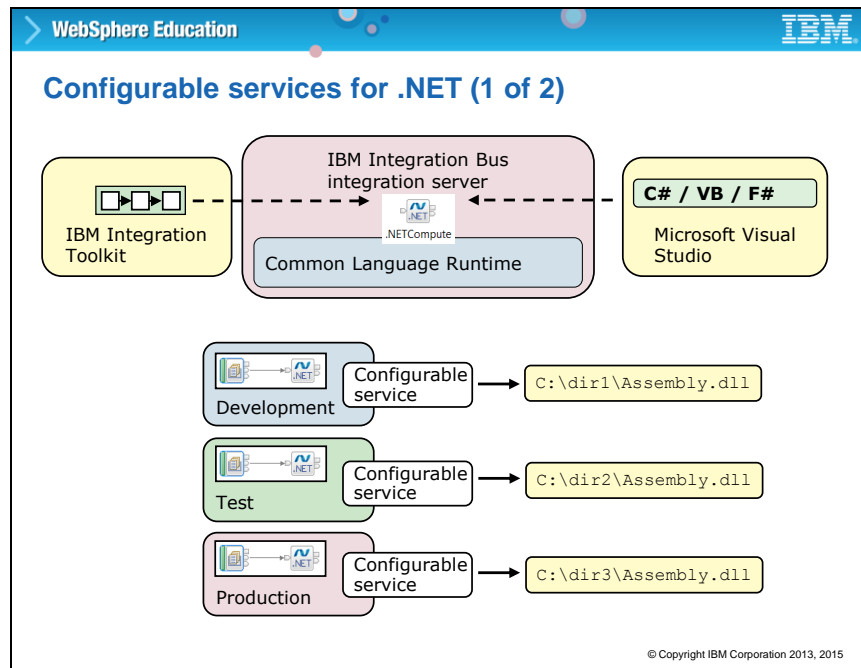
14 © Copyright IBM Corporation 2013, 2015

Handling exceptions

Any Integration Bus exceptions that occur at run time generate NBExceptions.

NBException represents the base class of the integration node exception hierarchy from which all integration node exceptions are derived.

In the .NET code, any NBExceptions are transformed into Integration Bus exceptions. As with any other message processing node, you can optionally wire the **Failure** terminal and the **Catch** terminal of the input node to handle the message.



Configurable services for .NET (1 of 2)

The properties on the .NET node or on the ESQL Procedure signature control the .NET configuration for an integration node. Optionally, a DotNetAppDomain configurable service can also be defined for further configuration options.

You can create a DotNetAppDomain configurable service to connect to .NET. At run time, you can reference a different assembly for the development, test, and production environments, as shown in the figure.

You must stop and start the integration server for a change in a property value to take effect in the DotNetAppDomain configurable services.

WebSphere Education
IBM

Configurable services for .NET (2 of 2)

IBM Integration

Filter Options...

TESTNODE_ibadmin
Servers
Operational Policy
Configurable Services
ActivityLog
Aggregation
CDServer
CICSConnection
CORBA
Collector
ConnectorProviders
DataCaptureSource
DataCaptureStore
DataDestination
DotNetAppDomain
AppDomainTemplate

AppDomainTemplate - DotNetAppDomain Configurable Service

Overview

Create and manage configurable services by using:

- IBM Integration web interface
- IBM Integration Bus commands

Properties

DisallowCodeDownload	true
ApplicationBase	
AllowHotSwapDeploy	true
UseBrokerWorkpathForShadowCopyCache	false
ShadowCopyFiles	true
ConfigurationFile	
PrivateBinPath	
PrivateBinPathProbe	

© Copyright IBM Corporation 2013, 2015

Configurable services for .NET (2 of 2)

You can create and manage the .NET configurable services in the Integration web user interface or by using the `mqsicreateconfigurablesevice` command.














The DotNetAppDomain properties are summarized in your student notes.

- IBM Integration Bus
- IBM.Broker.Plugin Namespace
- IBM.Broker.Plugin Namespace
- Data/Validation Enumeration
 - NByteArrayInputEvent Class
 - NByteArrayInputRecord Class
 - NByteArrayPollingResult Class
 - NBConnector Class
 - NBConnectorFactory Class
 - NBElementInputRecord Class
 - NBEventInputConnector Class
 - NBInputConnector Class
 - NBInputRecord Class
 - NBPollingInputConnector Class
 - NBPollingResult Class
 - NBTimeoutPollingResult Class

IBM.Broker.Plugin.Connector Namespace

This namespace provides the classes for creating a .NET Connectors for .NET Input nodes.



Classes

Class	Description
 NByteArrayInputEvent	NByteArrayInputEvent provides a default class to handle byte array based events.
 NByteArrayInputRecord	NByteArrayInputRecord provides a default class to handle input records for polling results and events.
 NByteArrayPollingResult	NByteArrayPollingResult provides a default class to handle byte array based polling results.
 NBConnector	An abstract base class that represents a connector.
 NBConnectorFactory	NBConnectorFactory is an abstract base class for connector factories.
 NBElementInputRecord	NBElementInputRecord provides a default class to handle input records for polling results and events.
 NBEvent	NBEvent is a base class which must be extended to be able to receive an event from the system.
 NBEventInputConnector	NBEventInputConnector is the base class which must be extended to implement a connector which can receive events from the system.
 NBInputConnector	NBInputConnector is an abstract base class which should not be extended by the user, but is extended by the system.
 NBInputRecord	NBInputRecord is an abstract base class for records returned from NBEvent.BuildInputRecord or NBPollingResult.BuildInputRecord.
 NBPollingInputConnector	NBPollingInputConnector is the base class which must be extended to implement a connector which can receive data from the system.
 NBPollingResult	NBPollingResult is the base class for a polling result from the system.
 NBTimeoutPollingResult	NBTimeoutPollingResult is a class which is used to indicate the ReadData method has no data to return.

© Copyright IBM Corporation 2013, 2015

The IBM Knowledge Center for Integration Bus contains documentation for the .NET API, which includes:

- For more information and examples of .NET integration with Integration Bus, see the Integration Toolkit tutorials and the Integration Bus patterns and samples on GitHub.



Deploying a .NET assembly

- .NET assembly contains the runtime code that the .NETCompute node calls and must be accessible to the integration node at run time
- .NET assembly can be included in the BAR file
- If the .NET assembly is not included in the BAR file, and it is loaded directly from the integration node file system, the assembly is found in one of the following ways:
 - Absolute directory path that is specified on a .NET node **Assembly name** property, or specified on an ESQL function or procedure
 - An override to the absolute directory path in a BAR file for the .NET node
 - A **DotNetAppDomain** configurable service that overrides the absolute directory path
 - Using advanced properties to search the global assembly cache

© Copyright IBM Corporation 2013, 2015

Deploying a .NET assembly

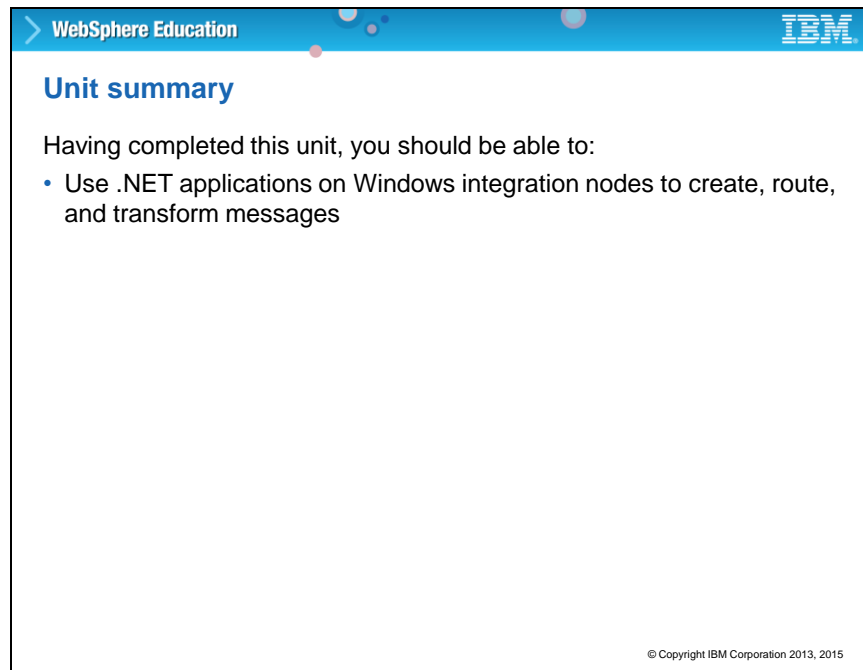
.NET assemblies can be included in a BAR file, or loaded directly from the integration node file system.

Include the .NET assembly in the BAR file by packaging it in a .NET application domain. The application domain is packaged as a .appdomainzip file at the root level of the BAR file. The .NET assembly is deployed with any other resources in the application domain.

If the .NET application domain is associated with an application, you can deploy the referencing application. The .NET assemblies in the application domain are deployed with the application. Alternatively, you can deploy the .NET application domain separately.

If the .NET assembly is not included in the BAR file, and is loaded directly from the integration node file system, the assembly is found in one of the following ways:

- The absolute directory path that is specified on a .NET node **Assembly name** property, or specified on an ESQL function or procedure.
- An override to the absolute directory path in a BAR file for the .NET node.
- A DotNetAppDomain configurable service that overrides the absolute directory path.



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. Below the header, the title 'Unit summary' is displayed in blue. The main content area contains a paragraph stating 'Having completed this unit, you should be able to:' followed by a bulleted list with one item: '• Use .NET applications on Windows integration nodes to create, route, and transform messages'. At the bottom right of the slide, there is a small copyright notice: '© Copyright IBM Corporation 2013, 2015'.

> WebSphere Education IBM

Unit summary

Having completed this unit, you should be able to:

- Use .NET applications on Windows integration nodes to create, route, and transform messages

© Copyright IBM Corporation 2013, 2015

Unit summary

You can use .NET applications on Windows integration nodes to create, route, and transform data by using the .NETCompute and .NETInput nodes. This unit described the .NETCompute node and the .NETInput node.

Having completed this unit, you should be able to use .NET applications on Windows integration nodes to create, route, and transform messages.