

iWay

iWay .NET Technology Adapter User's Guide

Version 7.0.x and Higher

Active Technologies, EDA, EDA/SQL, FIDEL, FOCUS, Information Builders, the Information Builders logo, iWay, iWay Software, Parlay, PC/FOCUS, RStat, Table Talk, Web390, WebFOCUS, WebFOCUS Active Technologies, and WebFOCUS Magnify are registered trademarks, and DataMigrator and Hyperstage are trademarks of Information Builders, Inc.

Adobe, the Adobe logo, Acrobat, Adobe Reader, Flash, Adobe Flash Builder, Flex, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Due to the nature of this material, this document refers to numerous hardware and software products by their trademarks. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2018, by Information Builders, Inc. and iWay Software. All rights reserved. Patent Pending. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Contents

Preface	7
Documentation Conventions	8
Related Publications	8
Customer Support	8
Help Us to Serve You Better	9
User Feedback	12
Information Builders Consulting and Training	12
1. Introducing the iWay .NET Technology Adapter	13
iWay .NET Technology Adapter Overview	13
Defining and Understanding User Proficiency Levels.	14
Installing the iWay .NET Technology Adapter	14
Required Installation Files.	15
Required User Provided Files.	16
Sample Files.	16
Understanding the Technology of the iWay .NET Adapter	17
Session Support.	19
Constructor.	19
Loading Assemblies Overview.	21
Parameters and Data Types.	22
Serialization.	22
Method States.	22
Calling COM Objects.	22
Component Information for the iWay .NET Technology Adapter	23
iWay Explorer.	23
iWay Service Manager.	23
iWay Business Services Provider.	23
2. Supported Platforms Matrix for iWay .NET Technology Adapter	25
Support Overview for iWay .NET Technology Adapter	25
Supported Versions for Microsoft .NET	26
Operating Systems for iWay .NET Technology Adapter	26
Database Drivers for iWay .NET Technology Adapter	27

Java Development Kit (JDK) for iWay .NET Technology Adapter	27
Communication Modes for iWay .NET Technology Adapter	27
Object Types and Interfaces for iWay .NET Technology Adapter	27
Communication Types for iWay .NET Technology Adapter	28
Supported Operations for iWay .NET Technology Adapter	28
Data Types Support	28
Other Functions for iWay .NET Technology Adapter	29
Known Limitations for iWay .NET Technology Adapter	29
Restrictions for .NET Framework	30
Common Language Run-time Scope for .NET	34
Related Information for Specific iWay Releases	34
3. Design Time Concepts and Configuration Tasks	35
Exploring Assemblies	35
Starting iWay Explorer	36
Adding the .NET Adapter to iWay Explorer	40
Working With a Target	41
Application Domain Boundary	45
Accessing .NET Classes in iWay Explorer	45
Calling Distributed Assemblies on Different Machines	45
Determining if a Method can be Called	46
Understanding the Assembly Viewer in iWay Explorer	47
Controlling the Adapter Behavior Using a Properties File	49
Understanding the Format of the Properties File	50
Usage Considerations and Examples	52
Display Assemblies, Hiding Classes and Methods, and Disabling Cache	57
Sample Properties File	62
4. Run Time Concepts and Configuration Tasks	65
Run Time Implementation	65
Understanding Design Time and Run Time Targets	66
Using the Persist Connection Parameter	66
Understanding the Relationship Between Java and .NET	67
Invoking the iWay .NET Technology Adapter	67

Understanding Run Time Types.....	68
Invoking the iWay .NET Technology Adapter Through an iWay Business Service.....	68
Invoking the iWay .NET Technology Adapter Through a Process Flow.....	73
5. Configuring the Adapter in an iWay Environment	93
Configuring the Adapter in iWay Service Manager	93
A. Samples and Reference Guide	97
Using COM Interop in the iWay .NET Technology Adapter	97
Generating XML Schema in Legacy Mode	98
ComposeDocMethod Function.....	99
ComposeRpcMethod Function.....	99
Generating XML Schema in the Current Adapter Version	100
Complex Custom Type Representation Rules.....	102
Namespace and Target Namespace.....	103
Formats of XML Request and Response Schemas.....	107
Modifying XML Schemas	108
Constructor Schemas.....	109
Method Schemas.....	110
Schema Properties Reference.....	111
Expanding Nodes Based on Different Complex Types.....	113
System.Collections.ArrayList Type.....	113
System.Collections.Generic.Dictionary Type.....	114
Array or System.Collections.Generic.List Type.....	114
Pointer Type.....	115
Constructor Type.....	115
Complete XML Request Schema.....	116
Additional Sample XML Documents	117
Constructor Example.....	117
Array Example.....	117
Pointer Example.....	118
Dictionary Example.....	119
ArrayList Example.....	120
XMLElement Example.....	120

List Example.....	121
Getting SESSIONID Example.....	121
Using SESSIONID Example.....	122
B. Known Issues and Limitations	123
Supported and Unsupported Areas	123
.Net Application Essentials.....	123
.Net Data and Modeling.....	123
.Net Framework 4.5 and 4 Scope List.....	124
Configuration.....	125
Common Language Runtime Scope.....	128
Tested Application Scope	128

Preface

This document is written for system integrators who develop client interfaces between .NET and other applications. It assumes that readers have a general understanding of Microsoft Windows and UNIX systems as well as some experience using Enterprise Information System (EIS) and integration products and an understanding of the products with which this software integrates, general knowledge of .NET applications and the .NET framework, and knowledge of integration processes and data models for the required application area.

Note: This Release 7.0.x content is currently being updated to support iWay Release 8.0.x software. In the meantime, it can serve as a reference for your use of iWay Release 8. If you have any questions, please contact Customer_Success@ibi.com.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		Contents
1	Introducing the iWay .NET Technology Adapter	Provides an overview of the iWay .NET Technology Adapter and how it works.
2	Supported Platforms Matrix for iWay .NET Technology Adapter	Specifies version, platform, and database support information for iWay .NET Technology Adapter.
3	Design Time Concepts and Configuration Tasks	Describes design time concepts and configuration tasks for the iWay .NET Technology Adapter.
4	Run Time Concepts and Configuration Tasks	Describes run time concepts and configuration tasks for the iWay Technology Adapter for .NET.
5	Configuring the Adapter in an iWay Environment	Describes how to configure the adapter in the iWay Service Manager Administration Console.
A	Samples and Reference Guide	Provides samples and reference information for the iWay .NET Technology Adapter.
B	Known Issues and Limitations	Describes known issues and limitations for iWay .NET Technology Adapter.

Documentation Conventions

The following table describes the documentation conventions that are used in this manual.

Convention	Description
<code>THIS TYPEFACE</code> or <code>this typeface</code>	Denotes syntax that you must enter exactly as shown.
<i>this typeface</i>	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option that you can click or select.
<u>underscore</u>	Indicates a default setting.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices. Type one of them, not the braces.
	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis (...).
.	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our Technical Content Library at <http://documentation.informationbuilders.com>. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about this product?

Join the Focal Point community. Focal Point is our online developer center and more than a message board. It is an interactive network of more than 3,000 developers from almost every profession and industry, collaborating on solutions and sharing tips and techniques. Access Focal Point at <http://forums.informationbuilders.com/eve/forums>.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our website, <http://www.informationbuilders.com>. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of <http://www.informationbuilders.com> also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

Call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your questions. Information Builders consultants can also give you general guidance regarding product capabilities. Please be ready to provide your six-digit site code number (xxxx.xx) when you call.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Help Us to Serve You Better

To help our consultants answer your questions effectively, be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the environment information our consultants require.

Platform	
Operating System	
OS Version	
JVM Vendor	
JVM Version	

The following table lists the deployment information our consultants require.

Adapter Deployment	For example, iWay Business Services Provider, iWay Service Manager
Container	For example, WebSphere
Version	
Enterprise Information System (EIS) - if any	
EIS Release Level	
EIS Service Pack	
EIS Platform	

The following table lists iWay-related information needed by our consultants.

iWay Adapter	
iWay Release Level	
iWay Patch	

The following table lists additional questions to help us serve you better.

Request/Question	Error/Problem Details or Information
Did the problem arise through a service or event?	
Provide usage scenarios or summarize the application that produces the problem.	
When did the problem start?	
Can you reproduce this problem consistently?	
Describe the problem.	

Request/Question	Error/Problem Details or Information
Describe the steps to reproduce the problem.	
Specify the error message(s).	
Any change in the application environment: software configuration, EIS/database configuration, application, and so forth?	
Under what circumstance does the problem <i>not</i> occur?	

The following is a list of error or problem files that might be applicable.

- ☐ Input documents (XML instance, XML schema, non-XML documents)
- ☐ Transformation files
- ☐ Error screen shots
- ☐ Error output files
- ☐ Trace files
- ☐ Service Manager package or archive to reproduce problem
- ☐ Custom functions and agents in use
- ☐ Diagnostic Zip
- ☐ Transaction log
- ☐ Archive File
- ☐ IIA

For information on tracing, see the *iWay Service Manager User's Guide*.

User Feedback

In an effort to produce effective documentation, the Technical Content Management staff welcomes your opinions regarding this document. You can contact us through our website, <http://documentation.informationbuilders.com/connections.asp>.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our website (<http://education.informationbuilders.com>) or call (800) 969-INFO to speak to an Education Representative.

Introducing the iWay .NET Technology Adapter

The following topics provide an overview of the iWay .NET Technology Adapter and how it works, including descriptions of key features and functionality.

In this chapter:

- ☐ [iWay .NET Technology Adapter Overview](#)
- ☐ [Installing the iWay .NET Technology Adapter](#)
- ☐ [Understanding the Technology of the iWay .NET Adapter](#)
- ☐ [Component Information for the iWay .NET Technology Adapter](#)

iWay .NET Technology Adapter Overview

The iWay .NET Technology Adapter enables .NET developers to create integration endpoints using their familiar design language or to reuse existing endpoints. Code written in a source language, such as Visual C# or Visual Basic .NET, that compiles into Common Language Infrastructure (CLI) code can be interpreted and the methods invoked from Java executables. The CLI code runs in an instance of the Common Language Runtime (CLR) and data can be passed through the adapter in either direction (inbound or outbound).

This version of the adapter introduces support for constructors, sessions, and database connections. In addition, enhanced metadata capabilities are provided.

iWay .NET Technology Adapter is designed to work with method calls contained in .NET assemblies. The source code input is not used only the Common Intermediate Language (CIL).

The adapter has been tested with .NET framework runtime 3.5, 4.0 and 4.5 on Windows client and server machines. No other platforms are currently supported.

While the new enhancements open new areas of the .NET framework to use with the adapter, some parts of the framework remain out of scope and are explicitly not supported:

- ☐ Screen or presentation elements or Windows Presentation Foundation
- ☐ .NET remoting, .NET web services, or Windows Communication Foundation
- ☐ Windows services or Windows Workflow
- ☐ ASP .NET

❑ Executable files

Do not use any class types that attempt to use Application environment variables. They will return the *iway_home* directory and not the expected assembly. All Assembly environment variables function as documented.

Known Issues and Limitations

For more information on supported functionality, usage considerations, known issues and limitations, see [Known Issues and Limitations](#) on page 123.

Defining and Understanding User Proficiency Levels

There are three different user levels that should be considered when using the iWay .NET Technology Adapter:

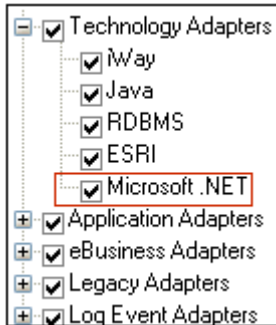
- ❑ **Basic User.** Performs beginner-level tasks and runs (explores) only specific Assemblies. Basic user tasks include locating Assemblies through iWay Explorer, creating XML schemas, retrieving data, creating XML instances, and executing XML request documents. The types of tasks that can be expected from a basic user are described in [Design Time Concepts and Configuration Tasks](#) on page 35.
- ❑ **Development User.** Creates integration based on existing Assemblies. Explores multiple Assemblies, understands parameters, return types, connections, and is comfortable using these objects without a guide. This user has a more detailed understanding of parameter types and how they appear in iWay Explorer, what objects can be exposed through XML, and what can be exposed only by a program (writing an agent). The types of tasks that can be expected from a basic user are described in [Design Time Concepts and Configuration Tasks](#) on page 35 and [Run Time Concepts and Configuration Tasks](#) on page 65.
- ❑ **Professional Developer.** Creates new code and develops integration. Develops .NET programs or Java programs and understands more about how to design for the adapter. Usage and knowledge of terms such as *protected*, *private*, *public*, *declarations*, and other terms are mandatory for this type of user. All of the topics and tasks in this documentation would be applicable to a professional developer.

Understanding these types of users and their specific roles can help define the expected usage scope for the iWay .NET Technology Adapter and its implementation in your organization.

Installing the iWay .NET Technology Adapter

Microsoft .NET Framework Version 3.5 and higher is the minimum requirement for running this release of the iWay .NET Technology Adapter.

In iWay Service Manager (iSM) Version 7.0, the iWay .NET Technology Adapter supports 32- and 64-bit JVM environments. The required components for both environments are automatically installed. During the iSM installation process, ensure that *Microsoft .NET* is selected under the Technology Adapters category in the Adapter Selection pane, as shown in the following image.



Required Installation Files

The iSM installation process installs the following adapter components in the `<ism_home>\lib` directory:

☐ **iwdotnet.jar**. Exposes design time and runtime interfaces for the iWay .NET Technology Adapter.

☐ **iwdotnet32.dll** or **iwdotnet64.dll**. Export the JNI methods that are required by the Java classes, which implement the adapter and act as a common language runtime host.

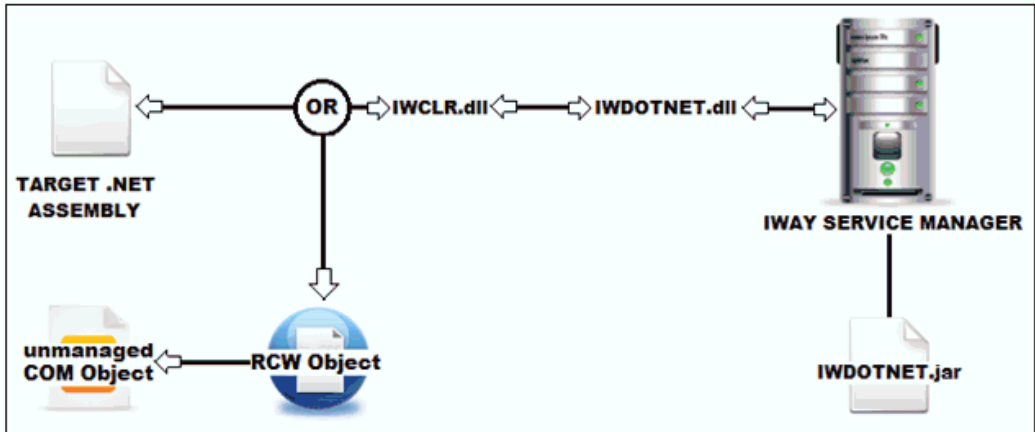
Note: Both versions of the iwdotnet DLL file are shipped with the adapter. Depending on the JVM, the adapter will select the appropriate DLL file to load. A 64-bit JVM can load 32 - and 64-bit Assemblies. A 32-bit JVM can only load 32-bit Assemblies no matter the platform type (32 - or 64-bit).

☐ **iwclr.dll**. Contains the functionality to explore Assemblies during design time, load and invoke classes and methods at runtime, and implement the optional custom attributes used for Assembly annotation.

Note: The installation will also install in the Windows Global Assembly Cache an iwclr module with a strongly named Assembly with key and version. This GAC resident copy of iwclr enables the adapter to reload Assemblies and to introspect GAC resident Assemblies when requested.

The following diagram illustrates the runtime architecture of the iWay .NET Technology Adapter when performing services that interact directly with a .NET application.

The iWay .NET Technology Adapter allows you to expose Microsoft .NET Assemblies in a J2EE environment. Unmanaged COM components may be invoked through a .NET Run Time Callable Wrapper (RCW).



Required User Provided Files

An instance of J2EE.jar, which is dated from 2009 or later is required.

The iWay .NET Technology Adapter has been tested with Microsoft .NET Framework Version 4.5 and 4.0, on both client and server machines. The CLI depends on class libraries that cannot be checked at runtime. Ensure the features of the framework that the deployed code will execute are present and installed on the current system before proceeding.

Sample Files

The iWay .NET Technology Adapter installs the following sample .NET Assemblies in the `<ism_home>\etc\samples\dotnet\bin` directory:

- ☐ **Complex.dll.** Nested classes sample.
- ☐ **Math.dll.** Using mathematical functions.
- ☐ **Misc.dll.** Parameter decoration sample.
- ☐ **scriptor.dll.** Sample text functions.
- ☐ **SubComplex.dll.** A subclass of the Complex sample.

These precompiled Microsoft .NET Assemblies can be used for adapter testing and verification purposes.

Understanding the Technology of the iWay .NET Adapter

The iWay .NET Technology Adapter uses the classes in the Reflection namespace of the .NET class libraries. These classes enable loading an Assembly, and obtaining information from the loaded Assembly about the classes, methods, interfaces, and data types of the Assembly. This information from Reflection will be used to create XML schemas. At runtime, an instance of the .NET Common Language Runtime (CLR) will be instantiated to execute the selected method or methods in the Assembly.

Unlike static compiled code, dynamic code does not have access to all the field values stored in the Assembly, only the types and methods obtained from the Reflection API.

The Reflection API requires that the classes or methods be *Serializable* and that the member to be introspected be public. Private methods are always skipped and returned as *private* with no body.

Terminology Used With the iWay .NET Technology Adapter

The terms in this section are used with reference to the ECMA international and Microsoft documentation on the C# and Visual Basic languages, and to the Common language infrastructure. For more information on the languages and runtime of .NET, refer to these sources.

- ❑ **Accessibility Domain.** A defined text placed before each class, member or variable, defining the visibility and usage. *Public* methods can be inspected, *Private* are hidden, and so on. Refer to the language specific documentation for the proper syntax to use for the method to be invoked.
- ❑ **Application Domain.** A container for the types and class libraries used in a particular application. For the use of the .NET adapter, libraries can be loaded from different application domains but the adapter itself does not use an application domain.
- ❑ **Assembly.** A configuration of code and resources created by a compiler to implement functionality.
- ❑ **Class.** A class is a container object that logically organizes data, functions, or combinations. A class physically is a location in memory, it is a pass by reference object, containing fields, methods, properties, and so on. A class can be instantiated by a constructor.
- ❑ **Class Library.** An Assembly that contains code that can be used by other Assemblies, often contained in an Assembly.
- ❑ **Common Type System.** The underlying type system that unifies the representation and storage of objects and data across languages within .NET.

- ❑ **Constant.** Text values embedded in a class or member.
- ❑ **Constructor.** A class member that can be used to create a new instance of the class.
- ❑ **Field.** A data holding member of a class or structure.
- ❑ **Method.** A member of a class that performs an operation on an object type.
- ❑ **Object.** A location in memory that has been created by a template, either class or structure or other.
- ❑ **Program.** Defined here a stand alone, portable execution format executable for the .NET framework.
- ❑ **Property.** A data holding member of a class or structure that can queried or set.

For a complete list of development sections of .NET, see the *Microsoft .NET Framework 4.5 Development Guide* and the *Framework Design Guides* for best practices.

For the adapter, it is a recommended practice that multiple application domains referencing the same types should not be loaded at the same time. The application domain boundary is at the Assembly level.

Do not use any class types that attempt to use *Application* environment variables. They will return the *ipay_home* directory and not the expected Assembly. All *Assembly* environment variables function as documented.

The following objects are introspected dynamically:

- ❑ **Assembly.** Retrieves the information on the Assembly and the classes in the Assembly. The Assembly scope is the highest level of adapter application support.
- ❑ **Classes.** Retrieves the public methods of each public class.
- ❑ **Method.** Retrieves the name, parameters and the return type.
- ❑ **Field.** Retrieves the information on fields defined in methods and classes.
- ❑ **Parameter.** Retrieves the information on parameters such as *type*, *in*, *out*, or *required*.

The iWay .NET Technology Adapter supports only .NET Assemblies or .NET class libraries as target destination types. PE format executables are not acceptable as input and will return an error message.

Only classes and methods with accessibility domain of *public* will be available through the adapter. The adapter will accept default and overloaded constructors, provided the .NET runtime libraries are available.

Constant or other text defined in classes is usually not returned after introspection, so explicit variables that accept input must be defined. The .NET *using* clause, combined with a type variable, is often not returned, declare an explicit type instead.

Session Support

Without Session behavior, the adapter operates only in *stateless mode*, where all parameters and variable values persist only for the duration for a XML document open tag and close tag on the document level.

During a cached session, the adapter will attempt to generate a state for a given Assembly, and the class instance properties will be persisted, so the ability to perform routines with counters, saved variables, and other work items is enabled.

Cache and session behavior can be controlled on the Assembly or method level. Hiding Assemblies or methods is available locally or globally as well.

For more information on how caching works, see [Constructor](#) on page 19. Only classes with constructors are eligible to run within sessions. The constructor can be *non-stated* (default) or *explicit* (overloaded).

Constructor

A constructor is a special section of a non static class that creates an instance of the class. There are two types of constructors that can be used with the adapter: default and overloaded.

A default constructor has no parameters, and overloaded constructor has one or more parameters. Default:

```
class foo
{
  foo(){
  }
}
Overloaded
class foo
{
  foo(string s){
    s = new string(' ',5);
  }
}
```

A static class must be declared explicitly as static:

```
static class sfoo
{
    string getfooName() {
        string s = "thefoo";
        return s;
    }
}
```

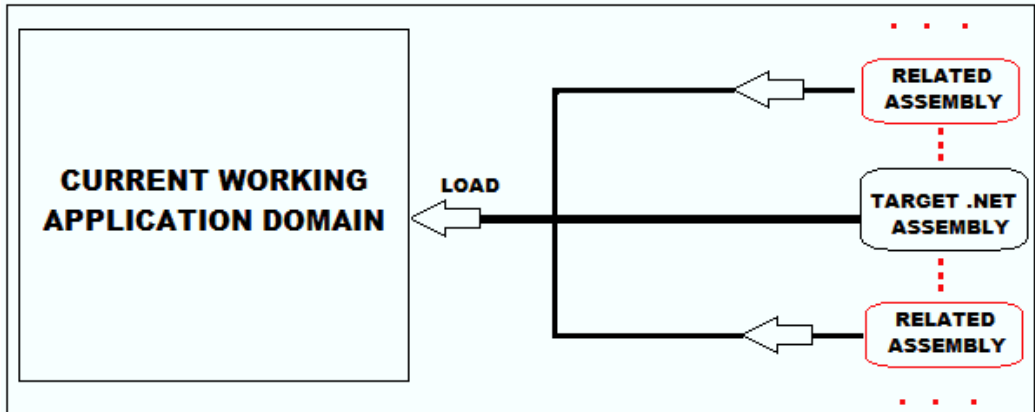
If a class has a constructor, then when calling a constructor from an XML request document (and for an overloaded constructor with the required variables), the adapter will attempt to construct and cache the object. If the constructor operation on the class is successful, then an Instance of the class is created and the adapter will return a Session ID in the XML response document.

When using the Assembly object with a method operation, using the Session ID reuses the same Assembly instance object. Using the Assembly object to call a method without a Session ID or with an invalid Session ID, causes the adapter to call the constructor on the object as *new*, without a session. The Session ID is the only reference to the cached Assembly instance. A session will end when the adapter is removed from the iWay runtime environment.

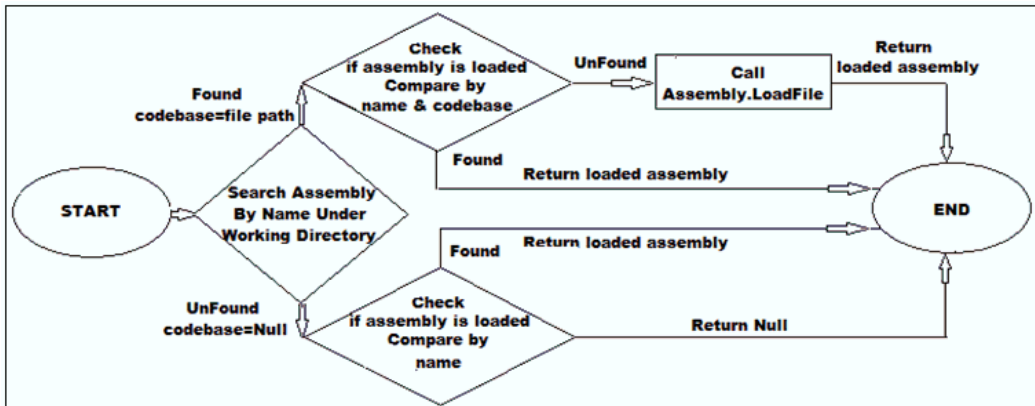
Ensure that the *Persist Connection* check box is selected during the configuration of adapter target properties in iWay Integration Tools (iIT) or the iWay Web Registry Console to use the session capability.

Loading Assemblies Overview

In order to generate request and response schemas for a method of a target .NET assembly, you have to know what methods the .NET assembly can support and what kind of input and output data types are used for a method. Moreover, you also need to know related data types for custom data types. To implement the purpose, the working application domain has to load the target .NET assembly and its related assemblies (for example, base class assembly, derived class assemblies, and so on).



The following logic process is used to handle the AssemblyResolve event.



A target assembly may define multiple classes, a class may have multiple public methods, and two schemas are used to one method. In order to hold all schemas for a target assembly, you use an XML document.

Parameters and Data Types

The iWay .NET Technology Adapter will return compatible data types for integration processing as best as possible. Some methods cannot be refactored for Java usage. If a parameter has a type of *System.Object*, then there is no good way of passing a .NET object from Java to .NET for invoking this method.

The adapter supports all Common Language Runtime (CLR) built in types and custom types, but not all types can be serialized.

When building a response schema, the adapter determines the response type from the .NET data type. If the .NET return type is a complex type that cannot be inferred, then the result will be *anyType* in the schema.

Serialization

It is not possible to use the .NET XML serializers in an integration context. If a *Datatable* or *DataSet* object is used in a method, then it is the responsibility of the caller to serialize the results into a returnable type for Java (either as a structure, an array of strings, or an array). Microsoft has published an article describing a technique for writing a data table to a multi-dimensional array list that can serve as an example. For more information, see the Microsoft Developer Network (MSDN) topic *Binary Serialization of ADO.NET Objects*.

<http://msdn.microsoft.com>

Method States

The state of all methods through the iWay .NET Technology Adapter is non-persistent. The scope is method scope. It is possible to use a constructor and call a method in a repeated persistent state. If multiple methods are needed to be called in sequence, with each result passed to subsequent method calls, then consider using an iWay process flow or writing a .NET wrapper object, which will make all the method calls in one .NET method body.

If you are using a session or one or more objects to enable object persistence, ensure that the Session ID is present and valid on all documents, and that the methods that are invoked do not cause the Assembly to be unloaded or garbage collected, as the result of invalid inputs could be unpredictable.

Calling COM Objects

The iWay .NET Technology Adapter works with .NET managed code only. If COM objects must be invoked from the adapter, then the technique of a Runtime Callable Wrapper (RCW) must be used to invoke a COM object and manage the lifecycle of the object. For more information on how to use it for a .NET to COM bridge, see the Microsoft Developer Network (MSDN) topic *Runtime Callable Wrapper*.

<http://msdn.microsoft.com>

Component Information for the iWay .NET Technology Adapter

The iWay .NET Technology Adapter works in conjunction with one of the following components:

- ☐ iWay Explorer
- ☐ iWay Service Manager
- ☐ iWay Business Services Provider (iBSP)

When hosted in an iWay environment, the adapter is configured through iWay Service Manager and iWay Explorer. iWay Explorer is used to configure .NET connections, create web services, and configure event capabilities.

When the adapter is hosted in a third-party application server environment, iWay Explorer can be configured to work in a web services environment in conjunction with iBSP.

iWay Explorer

iWay Explorer uses a tree metaphor to introspect the .NET system metadata. The explorer enables you to create XML schemas and web services for the associated object. In addition, you can create ports and channels to listen for events in .NET. External applications that access .NET through the iWay .NET Technology Adapter use either XML schemas or web services to pass data between the external application and the adapter.

iWay Service Manager

iWay Service Manager is the heart of the Universal Adapter Framework and is an open transport service bus. Service Manager uses graphical tools to create sophisticated integration services without writing custom integration code by:

- ☐ Creating metadata from target applications.
- ☐ Transforming and mapping interfaces.
- ☐ Managing stateless processes.

Its capability to manage complex adapter interactions makes it ideally suited to be the foundation of a service-oriented architecture.

iWay Business Services Provider

iWay Business Services Provider (iBSP) exposes (as web services) enterprise assets that are accessible from adapters regardless of the programming language or the particular operating system.

iBSP simplifies the creation and execution of web services when running:

- ☐ Custom and legacy applications.
- ☐ Database queries and stored procedures.
- ☐ Packaged applications.
- ☐ Terminal emulation and screen-based systems.
- ☐ Transactional systems.

Coupled with a platform and language independent messaging protocol called SOAP (Simple Object Access Protocol), XML enables application development and integration by assembling previously built components from multiple web services.

Supported Platforms Matrix for iWay .NET Technology Adapter

iWay Software is committed to support the diverse environments and varied systems of our users through support for leading enterprise applications, platforms, and databases.

This section specifies version, platform, and database support information for iWay .NET Technology Adapter. It is designed to provide a consolidated view of Microsoft .NET releases and the various operating systems and databases, on which they are supported.

In this chapter:

- ☐ [Support Overview for iWay .NET Technology Adapter](#)
- ☐ [Supported Versions for Microsoft .NET](#)
- ☐ [Operating Systems for iWay .NET Technology Adapter](#)
- ☐ [Database Drivers for iWay .NET Technology Adapter](#)
- ☐ [Java Development Kit \(JDK\) for iWay .NET Technology Adapter](#)
- ☐ [Communication Modes for iWay .NET Technology Adapter](#)
- ☐ [Object Types and Interfaces for iWay .NET Technology Adapter](#)
- ☐ [Communication Types for iWay .NET Technology Adapter](#)
- ☐ [Supported Operations for iWay .NET Technology Adapter](#)
- ☐ [Data Types Support](#)
- ☐ [Other Functions for iWay .NET Technology Adapter](#)
- ☐ [Known Limitations for iWay .NET Technology Adapter](#)
- ☐ [Restrictions for .NET Framework](#)
- ☐ [Common Language Run-time Scope for .NET](#)
- ☐ [Related Information for Specific iWay Releases](#)

Support Overview for iWay .NET Technology Adapter

iWay .NET Technology Adapter enables .NET developers to create integration endpoints or to reuse existing endpoints using their familiar design language. Code written in a source language, such as Visual C# or Visual Basic .NET, that compiles into Common Language Infrastructure (CLI) code can be interpreted and the methods invoked from Java executables.

The CLI code runs in an instance of the Common Language Runtime (CLR) and data can be passed through the adapter in either direction (inbound or outbound).

The adapter includes a .NET and a Java component version.

Supported Versions for Microsoft .NET

iWay .NET Technology Adapter supports the following Microsoft .NET versions:

- ☐ 3.0
- ☐ 3.5
- ☐ 4.0
- ☐ 4.5
- ☐ 4.5.1
- ☐ 4.5.2
- ☐ **Common Language Runtime Version:**
 - ☐ 2.0
 - ☐ 4.0

Operating Systems for iWay .NET Technology Adapter

iWay .NET Technology Adapter supports only the following operating systems:

- ☐ **Windows Client:**
 - ☐ Windows Vista
 - ☐ Windows 7
 - ☐ Windows 8
 - ☐ Windows 8.1
- ☐ **Windows Server:**
 - ☐ 2008
 - ☐ 2008R2

- ❑ 2012
- ❑ 2012R2

Database Drivers for iWay .NET Technology Adapter

iWay .NET Technology Adapter supports the following database drivers:

- ❑ Microsoft SQL
- ❑ IBM DB2
- ❑ Oracle
- ❑ Standard Microsoft-distributed database drivers

Java Development Kit (JDK) for iWay .NET Technology Adapter

iWay .NET Technology Adapter supports the Java Development Kit (JDK) versions that are listed in the *iWay Installation and Configuration Guide* under *Java Requirements*.

Communication Modes for iWay .NET Technology Adapter

iWay .NET Technology Adapter supports the following communication modes:

- ❑ **Services (Outbound).** iWay .NET Technology Adapter can send messages to Microsoft .NET.
- ❑ **Events (Inbound).** iWay .NET Technology Adapter can receive messages from Microsoft .NET.

Object Types and Interfaces for iWay .NET Technology Adapter

iWay .NET Technology Adapter supports the following object types and Interfaces:

- ❑ **Assembly.** Retrieves the information on the Assembly and the classes in the Assembly. The Assembly scope is the highest level of adapter application support.
- ❑ **Classes.** Retrieves the public methods of each public class.
- ❑ **Method.** Retrieves the name, parameters and the return type.

- ☐ **Field.** Retrieves the information on fields defined in methods and classes.
- ☐ **Parameter.** Retrieves the information on parameters such as *type*, *in*, *out*, or *required*.

Communication Types for iWay .NET Technology Adapter

The adapter itself uses the COM interface of .NET to start an instance of the .NET run time. All communication between the Java and C component interfaces of the adapter are proprietary JNDI components. Techniques such as .NET remoting are not supported because they work only on object references. The adapter works only on object instances.

Supported Operations for iWay .NET Technology Adapter

The iWay .NET Technology Adapter depends on the .NET code being called for the operation type. The security level of the user and the code written, determines the permitted operations.

Using iWay Explorer in iWay Integration Tools (iIT), the iWay .NET Technology Adapter supports the following operations:

- ☐ Search Recursive/Exclusive/Assembly name
- ☐ Infer complex relationships when creating schemas
- ☐ Hide inherited methods from System.Object
- ☐ Display classes and methods
- ☐ Runtime operations
- ☐ Call Constructors (optionally, establish a session)
- ☐ Invoke methods of classes or COM objects

For more information, see [Known Limitations for iWay .NET Technology Adapter](#) on page 29.

Data Types Support

The following simple and value Data Types are supported by the iWay .NET Technology Adapter:

- ☐ bool
- ☐ byte
- ☐ char

- ☐ decimal
- ☐ double
- ☐ enum
- ☐ float
- ☐ int
- ☐ long
- ☐ sbyte
- ☐ short
- ☐ struct
- ☐ uint
- ☐ ulong
- ☐ ushort

All calls through the adapter between Java and .NET must be *pass by value* and not *pass by reference* since a .NET reference has no meaning in Java. Explicit object passing is not available between Java and .NET. The user must create an object serializer or marshaller if an instance of a class is to be passed between Java and .NET. Support for reference types and delegates are available between .NET classes.

Other Functions for iWay .NET Technology Adapter

There is no known list of other functions for the iWay .NET Technology Adapter.

Known Limitations for iWay .NET Technology Adapter

No explicit or implicit guarantee of compatibility or performance with user application components is given with the iWay .NET Technology Adapter. The scope of the framework is too wide and deep to do so. If you have specific questions about supported components of .NET or application compatibility, contact your customer support representative.

The following areas of .NET are not supported:

- ☐ Windows Store, Windows Forms, XAML, and any display methods

- ☐ Dynamic assemblies
- ☐ Network I/O
- ☐ SOAP Serialization
- ☐ Write to console

Using .NET serialization classes are not supported for integration.

The following list is arranged by features that are not supported, not supported for integration, and not supported Entity Framework restrictions (.NET areas).

Not supported:

- ☐ WCF

Not supported for integration:

- ☐ Writing to a file or XML file.

Not supported Entity Framework restrictions (.NET areas):

- ☐ Windows Presentation Foundation and Windows forms
- ☐ Common Client technologies
- ☐ Windows Service applications
- ☐ Parallel and asynchronous processing
- ☐ Windows Communication Foundation
- ☐ Windows Identity Foundation
- ☐ Windows Workflow Foundation
- ☐ Platforms other than Windows client or server

Restrictions for .NET Framework

The following technology support list shows the .NET Framework that is arranged by features that are supported, not supported, limited support, limited and legacy support, and partially supported.

Supported

- ☐ 64-bit application development

☐ Application domains

☐ Assemblies

☐ Collections

☐ Common language runtime (CLR)

Not Supported

☐ .NET for Windows Store applications

☐ Accessibility

☐ Add-ins

☐ ASP.NET

☐ Assembly binding redirection

☐ Asynchronous programming

☐ Code DOM

Limited Support

☐ .NET Framework Class Library

☐ Common type system

Limited and Legacy Support

☐ Attributes

Partial Support

☐ ADO.NET

Supported

☐ Exceptions

☐ Generics

☐ Files and streams

☐ Interoperability

☐ Side-by-Side Execution in the .NET Framework

Not Supported

- ☐ Configuring Applications
- ☐ Data Service
- ☐ Debugging, tracing, and profiling
- ☐ Deploying applications
- ☐ Designers and the design environment
- ☐ Directory services
- ☐ Dynamic Language Runtime (DLR)
- ☐ GDI+
- ☐ Compressing files
- ☐ Image file handling
- ☐ Working with Images, Bitmaps, Icons, and Metafiles
- ☐ Images
- ☐ Lazy initialization
- ☐ Managed Extensibility Framework (MEF)
- ☐ Media and multimedia:
 - ☐ Graphics and Multimedia in Window Presentation Foundation
 - ☐ Graphics and Multimedia Portal
- ☐ Memory-mapped files
- ☐ Moving user interface elements
- ☐ MSBuild
- ☐ Network programming
- ☐ Out-of-band (NuGet) releases
- ☐ Parallel programming
- ☐ Portable Class Library

- ☐ Silverlight
- ☐ Transaction processing
- ☐ UI Automation
- ☐ WCF Data Services
- ☐ Windows Communication Foundation
- ☐ Windows Forms
- ☐ Windows Forms controls
- ☐ Windows Identity Foundation
- ☐ Windows Presentation Foundation (WPF)
- ☐ Windows services
- ☐ Windows Store applications
- ☐ Windows Workflow Foundation (WF)
- ☐ Zip files and archives

Limited Support

- ☐ Data access
- ☐ Globalization and localization
- ☐ I/O
- ☐ LINQ (Language-Integrated Query)
- ☐ Reflection
- ☐ Security in the .NET Framework
- ☐ Serialization
- ☐ Threading
- ☐ Windows Runtime
- ☐ XML documents and data

Partial Support

- ☐ Events

Implicit Only

- ☐ Garbage Collection

Classes in .NET are passed by reference, and cannot be used for integration as a result. Structures are passed by value, and the results can be passed between Java and .NET. Native types in the common type library are supported by default. Defined types must have a constructor if instance-based or declared as static if not.

Serialization of data between Java and .NET is possible, but direct serialization or synchronization of objects is not currently supported. The underlying APIs of Java and .NET do not support the level of object graph serialization required.

The Entity Framework uses a conceptual model to access an underlying data object. Because the model may have any type of design, some model types are not useful with the adapter. Some models are designed for update only, or access only by a data control object in a form. The ideal framework model for the adapter returns an object or contains an object query. Then object query can be serialized through a structure and array list for integration.

Common Language Run-time Scope for .NET

The adapter uses the .NET unmanaged COM API to instantiate the .NET Common Language Runtime (CLR). The .NET CLR is not an emulated environment, but communication with the instance is governed by the adapter. There are several languages that can generate Microsoft Intermediate Language (MSIL) code. The adapter has been extensively tested with C# and on a more limited basis with source from Visual Basic and F#. For adapter usage requirements with any other language, contact your customer support representative.

Related Information for Specific iWay Releases

For more information, see the *iWay New Features Bulletin and Release Notes* documentation for a specific release (for example, iWay Version 7.0.3).

Design Time Concepts and Configuration Tasks

This section describes design time concepts and configuration tasks for the iWay .NET Technology Adapter. For example, how iWay Explorer is used to create schemas and Business Services to provide integration between the adapter and a .NET application for services.

Note: Before continuing with design-time configuration tasks, refer to the application scope before using the iWay .NET Technology Adapter. For more information, see [Tested Application Scope](#) on page 128.

In this chapter:

- ☐ [Exploring Assemblies](#)
 - ☐ [Starting iWay Explorer](#)
 - ☐ [Adding the .NET Adapter to iWay Explorer](#)
 - ☐ [Working With a Target](#)
 - ☐ [Understanding the Assembly Viewer in iWay Explorer](#)
 - ☐ [Controlling the Adapter Behavior Using a Properties File](#)
-

Exploring Assemblies

In a development environment a *project* is the high level container.

A project contains source files and references to other projects.

In the source files are descriptions to classes and methods that are *compiled* to make a run time assembly.

In the iWay .NET Technology Adapter, a *target* is the high level container. A target is a reference to a location where one or more Assemblies are located directly on the machine running iWay Integration Tools (iIT) with iWay Explorer.

Note: The .NET framework does not support prefixed device names, such as:

`\\sharedrive\\sharedirectory`

This is the case even if the shared drive shows as a letter in the current Windows Explorer.

Upon opening a target, one or more Assemblies are displayed. This display behavior can be customized with individual folder profiles or one global profile. For more information, see [Controlling the Adapter Behavior Using a Properties File](#) on page 49.

An Assembly contains classes, which are grouping containers of similar objects that allow actions or methods to be performed on the objects. The usual term is the class exposes methods. Each method in the iWay explorer is exposed concatenated to the class that exposes the method. This is a different view of objects than that of a development environment, but very useful when the method must be used for performing an action. Depending on how the Assembly was programmed, the navigation tree of the assembly will be simple or complex.

When exploring a method, the inbound and outbound parameters that the method exposes will be displayed by name in the tree and by type in the detail pane of iWay Explorer. The .NET framework has a series of types called the *Common Type System* that defines a set of types that are used throughout the .NET framework. It is possible to define custom types made up of the common types or build structures from the common types.

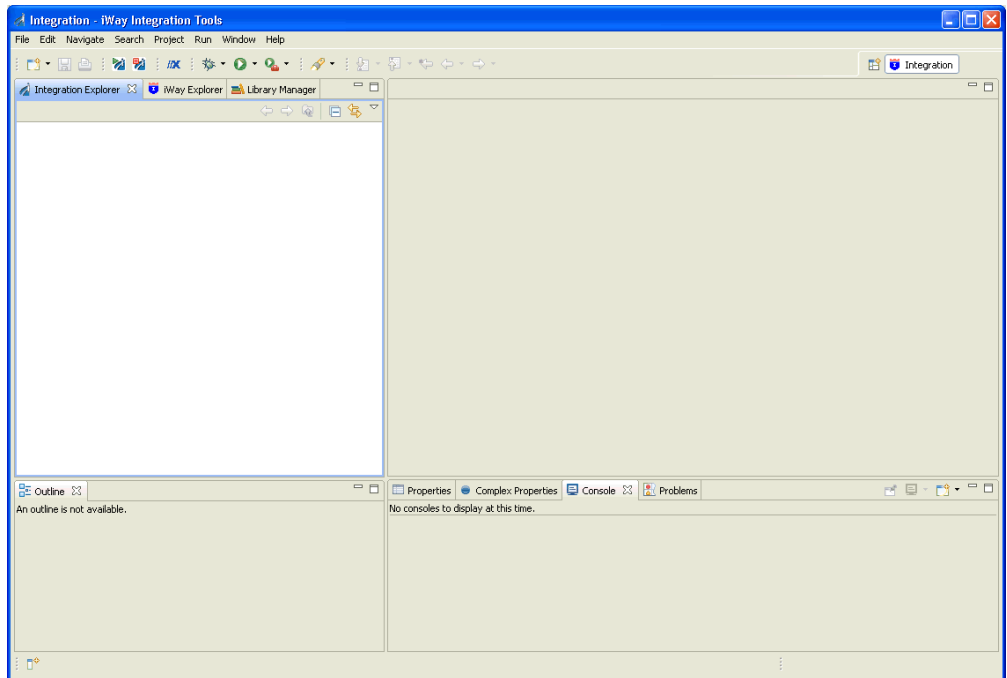
Starting iWay Explorer

This section describes how to start iWay Explorer.

Procedure: How to Open iWay Integration Tools

1. Navigate to your local drive where you have iIT installed, and open the *eclipse* folder.
2. Double-click *iit.exe*.

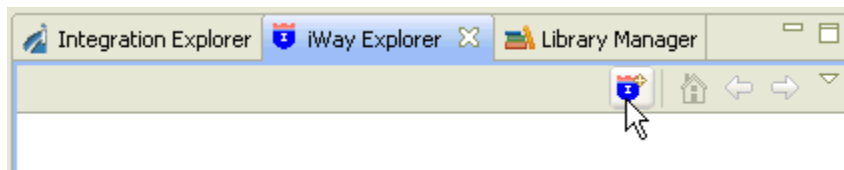
iWay Integration Tools suite opens.



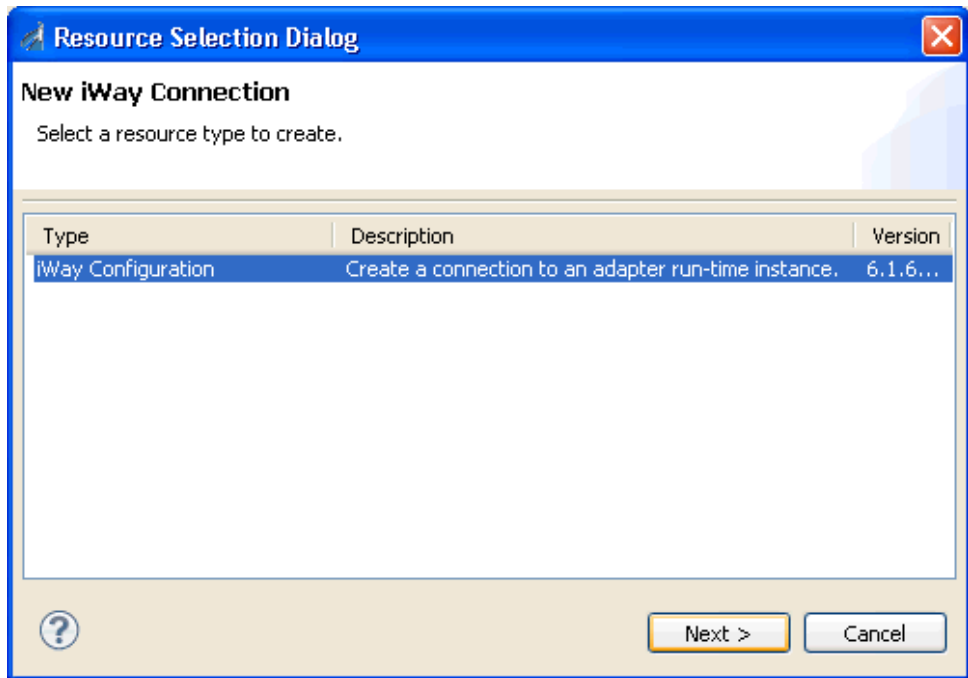
Procedure: How to Create an iWay Explorer Connection to an iSM Server

This procedure assumes that you have opened iWay Integration Tools (iIT) and are in the Workbench.

1. Click the *iWay Explorer* tab to make it active.
2. Click the *Launch iWay Resource Creator Wizard* button on the tool bar.
In the following image, the iWay Explorer tab is active, and the cursor is pointing to the Launch iWay Resource Creator Wizard button.



When you click the button, the Resource Selection Dialog opens and displays the New iWay Connection pane, as shown in the following image.



3. Under the Type heading, click *iWay Configuration*, which is the type of resource that you are going to create.
4. Click *Next*.

The Add iWay Configuration dialog box opens and displays the Select Connection Types pane.

5. In the Configuration Alias field, type a name for the new configuration (for example, *SampleConfig*).

Note: The name that you supply is used only for display purposes in the tree. It is not a server connection property.

6. For Connection Type, ensure that *HTTP Connection* is selected.
7. Optionally, select the *Connect to Host upon Wizard Completion* check box if you want iWay Explorer to automatically connect to this instance of iSM after you have created it. If you select this option, all the explorer environments under the new iSM connection are automatically connected to iSM when this procedure is finished.

If you do not select this option, the explorer environments are not automatically connected to iSM. You can connect to an individual explorer environment when you want to access it.

8. Click *Next* to continue the procedure.
9. If you selected an HTTP Connection, the Enter Connection Information pane opens, as shown in the following image.

Add iWay Configuration

Enter Connection Information

Provide the server's connection string, SOAP port and the iSM console port.

Connection String:

User Name:

Password:

SOAP Port/Endpoint:

Console Port/Endpoint:

Presets

- ☐ Verify the values in the three fields, or type the valid value or values.
 - ☐ The Connection String field contains the URL that connects to the iSM.
 - ☐ The SOAP Port/Endpoint field contains the SOAP port number.
 - ☐ The Console Port/Endpoint field contains the port number that the iSM Administration Console is listening on.
- ☐ Optionally, under Presets, click *Local Connection* to insert values for a local default iSM connection in the fields, or click *Servlet* to insert values for a sample servlet connection.

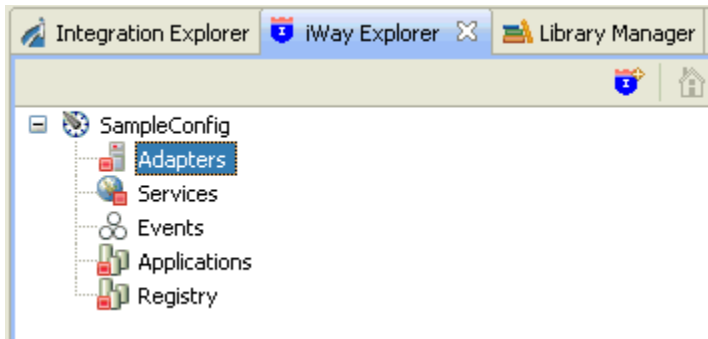
- ☐ Click *Finish*.
- ☐ In the File Path field, browse to the full path for your iWay installation directory and insert the path in the field. This path is used to locate the iWay adapters and store the XML schemas. For example:

`C:\Program Files\iWay7`

- ☐ In the Configuration Name field, verify the name of your iWay server configuration. The base configuration is specified by default. Alternatively, you can type the name of the server configuration.
- ☐ Click *Finish*.

The new iSM connection is added to the tree on the iWay Explorer tab.

In the following image, an iSM connection named SampleConfig was added to iWay Explorer. The tree is expanded to show the five explorer environments that are available.



Adding the .NET Adapter to iWay Explorer

iWay Explorer supports access to many different application systems. When you connect to and expand the Adapters node, the iWay adapters for the supported application systems are displayed. They are the iWay adapters that you have installed and are licensed to use.

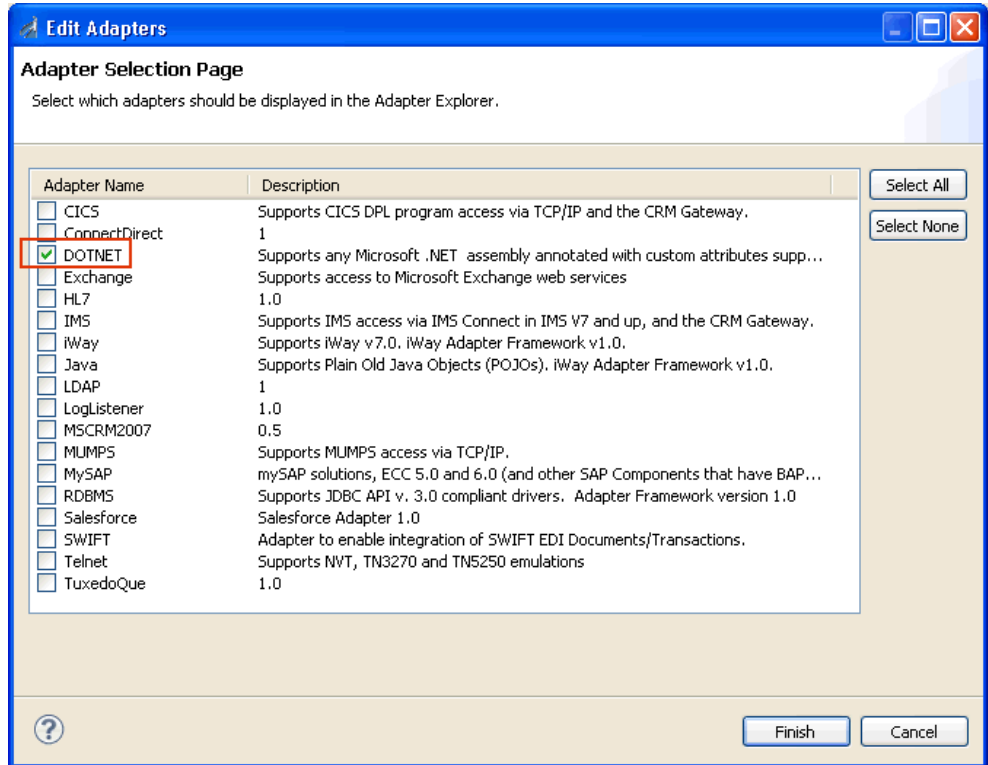
Procedure: How to Add the .NET Adapter to iWay Explorer

In this procedure, you are going to add the iWay .NET Technology Adapter to the list of adapters displayed in the Adapters node.

1. Right-click the *Adapters* node, and click *Edit* from the menu.

The Edit Adapters dialog opens, prompting you to select the iWay adapter or adapters to add to iWay Explorer.

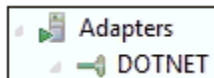
2. Select the check box for *DOTNET*, as shown in the following image.



3. Click *Finish*.

The tree is automatically refreshed and displays the new adapter.

In the following image, the DOTNET node is displayed in the Adapters node of iWay Explorer, as shown in the following image.



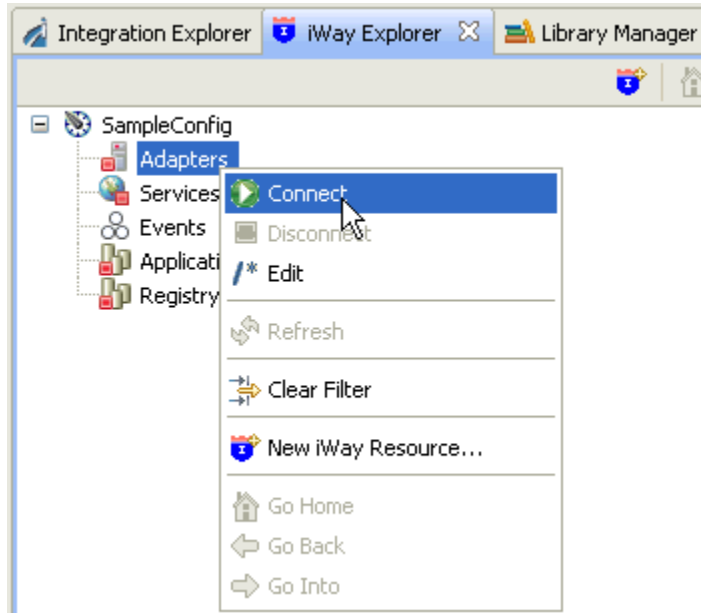
Working With a Target

To browse the metadata and objects of an application system, you must create a target for that system. The target is the means by which you connect to the system. It contains the logon properties used to access the system.

Using the target, you must establish a connection to an application system every time you want to browse the system in iWay Explorer.

Procedure: How to Create a Target

1. Right-click the *Adapters* node, and click *Connect* from the menu, as shown in the following image.



2. Once you are connected, expand the *Adapters* node.
3. Right-click *DOTNET*, and click *Add Target* from the menu.
The Add Target dialog opens and displays the Generic Target Properties pane.
4. Supply the values for the fields on the dialog box as follows.
 - a. In the Name field, type a descriptive name for the target (for example, *DOTNET_Target*).
 - b. In the Description field, optionally type a brief description of the target.
5. Select the *Connect to target upon wizard completion* check box if you want iWay Explorer to automatically connect to this target after it has been created.

If you deselect this option, iWay Explorer will not automatically connect to the target. From the tree, you can connect to an individual target when you want to access the associated application system.

6. Click *Next*.

The Add Target dialog opens and displays the Target Properties pane, as shown in the following image.

Add Target

Target Properties

Please enter the properties associated with the new target.

Assemblies' Directory
c:\assembliesdirectory

Search Recursively
true

Look for
All Native Assemblies

Assembly List

Infer Complex Schemas
false

Hide System.Object methods
true

? < Back Next > Finish Cancel

7. Supply the connection information for the .NET application to which you are connecting.

The following table lists and describes the .NET connection parameters.

Parameter	Description
Assemblies Directory	<p>Enter the starting folder to look for the .NET assembly to introspect.</p> <p>Note: The drive for the Assembly must reside on the same machine where iWay Service Manager (iSM) is installed. Mapped and network drives do not qualify for inspection. This is a limitation of .NET, not the adapter.</p>
Search Recursively	<p>Select <i>true</i> or <i>false</i> from the drop-down list.</p> <p>Search only the current folder entered in the directory box, or explore down the hierarchy. The adapter will generate an explorer type interface for all of the folders from the entered folder down. If a drive letter is entered, then all top level folders are displayed, and you can navigate down to locate the assembly.</p>
Look for	<p>Select one of the following options from the drop-down list:</p> <p><input type="checkbox"/> iWay Decorated Assemblies. Enables legacy mode.</p> <p><input type="checkbox"/> All Native Assemblies. Enables browse mode if Search Recursively is set to <i>true</i>.</p>
Assembly List	Enter a comma-delimited list of .NET assemblies to introspect.
Infer Complex Schemas	<p>Select one of the following options from the drop-down list:</p> <p><input type="checkbox"/> true. Infer complex relationships between assembly elements (recommended).</p> <p><input type="checkbox"/> false. Present only assembly elements.</p>

Parameter	Description
Hide System.Object Methods	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <input type="checkbox"/> true. Hide all methods inherited from object (ToString, GetHashCode, and so on) for each object. <input type="checkbox"/> false. Display all methods including inherited method, for each object. <p>Note: Hiding inherited methods is useful when you know the method(s) you wish to access and want to retrieve them quickly without viewing a long list.</p>

8. Click *Finish* when you are done.

The new .NET target is added to the Adapters node of iWay Explorer.

Application Domain Boundary

Only native drives on the current machine where iWay Service Manager is running are eligible for introspection. Mapped drives will generate errors. The adapter does not support the Microsoft deprecated technique called *.NET remoting*. For more information, see the Microsoft Developer Network (MSDN) topic on this subject for .NET Framework Version 4 or 4.5:

<http://msdn.microsoft.com>

Accessing .NET Classes in iWay Explorer

The default accessibility of a .NET class is not public. Classes and methods that are to be exposed using the iWay .NET Technology Adapter must use the *public* keyword in the declaration to be visible in iWay Explorer.

Calling Distributed Assemblies on Different Machines

The iWay .NET Technology Adapter does not support the .NET remoting protocol as either a client or server. In addition, the adapter does not support the Windows Communication Foundation as either client or server. If the user application requires calling distributed Assemblies on different machines, then instances of iWay Service Manager (iSM) and the iWay .NET Technology Adapter must be installed on the machines and use supported iSM methods of integration (TCP, FTP, web services, and so on).

Determining if a Method can be Called

Using the Parameters tab in iWay Integration Tools, click on the parameter in the method and find the type of the parameter. Verify that the type is an XML serializable type. If the type is not an XML serializable type, then special actions will be required to invoke the method, or it may not be possible to invoke the method through the adapter.

For example, the *getItemNumber* method has type parameter *value* of type *System.Int32*, *integer*. This can be used by the adapter.

The *DynamicInvoke* method has a parameter *args* represented as an array of *System.Object*. This type is not directly XML serializable, as *object* is an abstract root without a distinctly defined type.

Procedure: How to Connect to a Target

1. Expand the *DOTNET* node to locate the name of the target that you want to connect to, for example, *DOTNET_Target*.
2. Right-click the target, and click *Connect* from the menu.

The *DOTNET_Target* node icon changes to green, which indicates a successful connection. You can click a folder and then expand it to display its contents.

Procedure: How to Disconnect From a Target

Although you can maintain multiple open connections to different application systems, it is a good practice to close a connection when you are not using it.

1. In the tree, expand the *DOTNET* node to locate the name of the target from which you want to disconnect, for example, *DOTNET_Target*.
2. Right-click the target, and click *Disconnect from Target* from the menu.

The connection to the .NET application is closed.

Procedure: How to Edit a Target

After you create a target, you can edit the information that you provided during the creation procedure.

1. In the tree, expand the *DOTNET* node to locate the name of the target that you want to edit, for example, *DOTNET_Target*.
2. Right-click the target, and click *Edit Target* from the menu.

The Edit Target dialog opens and displays the DOTNET adapter target properties.

3. Modify the connection properties as required.

4. Optionally select the *Reconnect to target upon wizard completion* check box if you want iWay Explorer to automatically connect to this target after it has been edited. iWay Explorer will use the modified properties to connect.
5. Click *Finish* when you have made your edits.

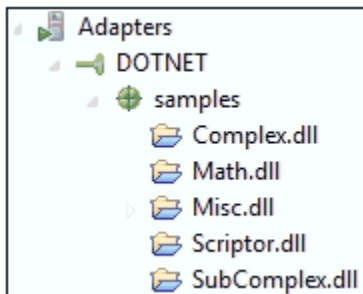
Procedure: How to Delete a Target

You can delete a target that is no longer needed. You can delete it whether or not it is closed. If open, the target automatically closes before it is deleted.

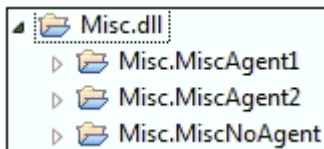
1. In the tree, expand the *DOTNET* node to locate the name of the target that you want to delete, for example, *DOTNET_Target*.
2. Right-click the target, and click *Delete Target* from the menu.
iWay Explorer displays a prompt, asking you to confirm the deletion of the selected target.
3. Click *OK* to proceed with the deletion.

Understanding the Assembly Viewer in iWay Explorer

iWay Explorer will first display the .NET assemblies found in a particular directory, in this example, the *samples* directory.



Expand an assembly, and the list of classes for the assembly are displayed.



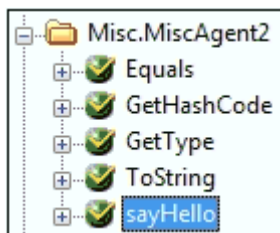
In the Detail pane, the Assembly Information window will display relevant information on the selected assembly, as shown in the following image.

Name	Misc.dll
Type	Folder
Container	true
Searchable	false
iwaf.description	
Node Type	Assembly
Assembly Path	G:\wx39883\etc\samples\dotnet\bin\Misc.dll
Assembly Full Name	Misc, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Number of Classes	3

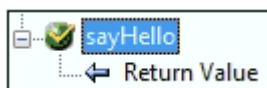
The Detail pane will display information on the class, as shown in the following image.

Name	Value
Name	Misc.MiscAgent2
Type	Folder
Container	true
Searchable	false
waf.description	MiscAgent2 Agent
Number of Methods	5

Expand a class and the methods of the class appear, as shown in the following image.



Expand a method, and the method information displays, as shown in the following image.



The Detail pane displays information on the selected method, as shown in the following image.

Detail	
Name	Value
Name	sayHello
Type	Operation
Container	true
Searchable	false
iwaf.description	Returns hello message
Input Parameters	None
Return Type	System.String

To create XML schema for a method, right-click the method and select *Open Schemas* from the context menu.

To create an iWay Business Service (web service) for a method, right-click the method and select *Create iWay Business Service* from the context menu.

Controlling the Adapter Behavior Using a Properties File

To hide or show specific classes, methods, or folders, and enable or disable the cache function, create a property in XML format called *iwaf.xml*. This properties file must be copied to the path that is specified in the Assemblies' Directory field during the creation of the target.

However, if you want to use default system settings, then you are not required to create the *iwaf.xml* properties file. By default, the cache is disabled and all assemblies or folder and public subclasses, methods can be displayed.

The caching behavior of iWay .NET Technology Adapter is configured by iWay Service Manager (iSM) in cooperation with the adapter during iSM initialization. When the caching settings are changed in the *iwaf.xml* properties file configuration files are changed, iSM must be restarted to read the updated properties file.

Note: A separate *iwaf.xml* properties file can be created for each adapter target. As a result, specific cache behaviors can be applied to different Assemblies.

To review a sample *iwaf.xml* properties file, see [Sample Properties File](#) on page 62.

Understanding the Format of the Properties File

The following image illustrates the settings and format of the iway.xml properties file.

```
<iwaydotnetadapter cache="[BOOLEAN]">
  <nodes>
    <node isassembly="[BOOLEAN]" path="[AF_PATH]" display="[BOOLEAN]" cache="[BOOLEAN]">
      <class name="[CLASS_NAME]" display="[BOOLEAN]">
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
        . . .
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
      </class>
      . . .
      <class name="[CLASS_NAME]" display="[BOOLEAN]">
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
        . . .
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
      </class>
    </node>
    . . .
    <node isassembly="[BOOLEAN]" path="[AF_PATH]" display="[BOOLEAN]" cache="[BOOLEAN]">
      <class name="[CLASS_NAME]" display="[BOOLEAN]">
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
        . . .
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
      </class>
      . . .
      <class name="[CLASS_NAME]" display="[BOOLEAN]">
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
        . . .
        <method name="[METHOD_SIGNATURE]" display="[BOOLEAN]" />
      </class>
    </node>
  </nodes>
</iwaydotnetadapter>
```

where:

[BOOLEAN]

Is a boolean value (true or false).

[AF_PATH]

Is the full path of an assembly or folder.

[METHOD_SIGNATURE]

Is the signature of a method, which is the combination of the name of the method along with the number and types of the parameters (and their order).

[CLASS_NAME]

Is a full class name which is equal to:

```
[namespace] + '.' + [classname]
```

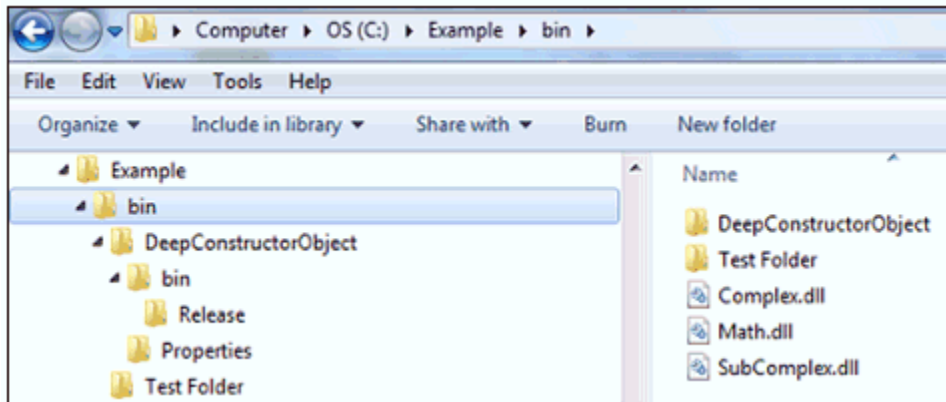
In this case `[namespace]` is the name of a namespace and `[classname]` is the name of a class.

The following is a list of rules and guidelines for the creation and usage of the `iway.xml` properties file:

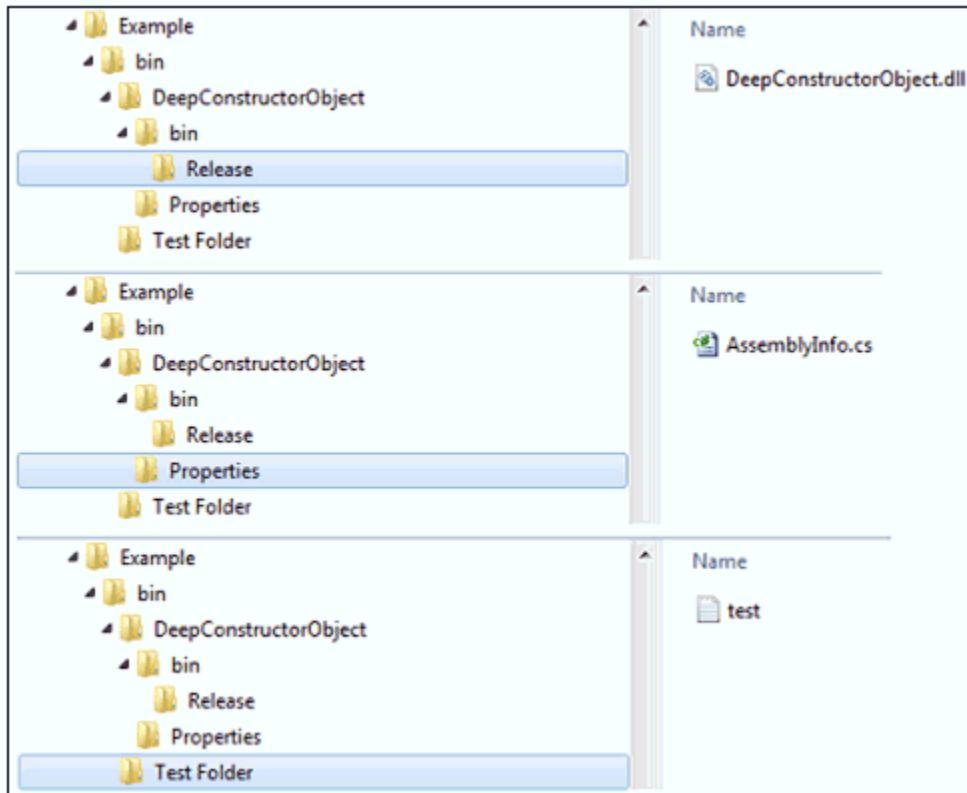
1. You can set the `cache` attribute value to `true` to enable caching or `false` to disable caching. The default cache value is the value of the `cache` attribute of the root element `iwaydotnetadapter`. This is used for all assemblies under the path that is specified in the Assemblies' Directory field during the creation of the target. The only exception is that subassemblies have their own cache values. If a subassembly has a cache value, then the iWay .NET Technology Adapter will base the cache value to enable or disable cache for the assembly. Otherwise, the iWay .NET Technology Adapter will use the default cache value setting.
2. To hide or show a folder, assembly, class, method (for example, item), you can set the value of the `display` attribute value of the corresponding item to `true` or `false`. By default, all assemblies, folders, public subclasses, and methods can be displayed. Therefore, if you do not want to hide an item and its sub-items, do not list this item in the `iway.xml` properties file.
3. If a method is inherited from the `System.Object`, then the value of the `display` attribute of the method can overwrite the value of the `Hide System.Object methods` property for the method. Therefore, you can show or hide inherited `System.Object` methods under a particular class based on their own status.
4. If you enable cache for an assembly, then the constructors of the classes of the assembly will be displayed in iWay Explorer.

Usage Considerations and Examples

For demonstration purposes, this example has a subfolder called *bin* located in the C:\Example directory. In addition, there is no *iway.xml* file saved in the \bin subfolder, as shown in the following image.



The following compiled image shows the contents of the `\Release`, `\Properties`, and `\Test Folder` subdirectories.

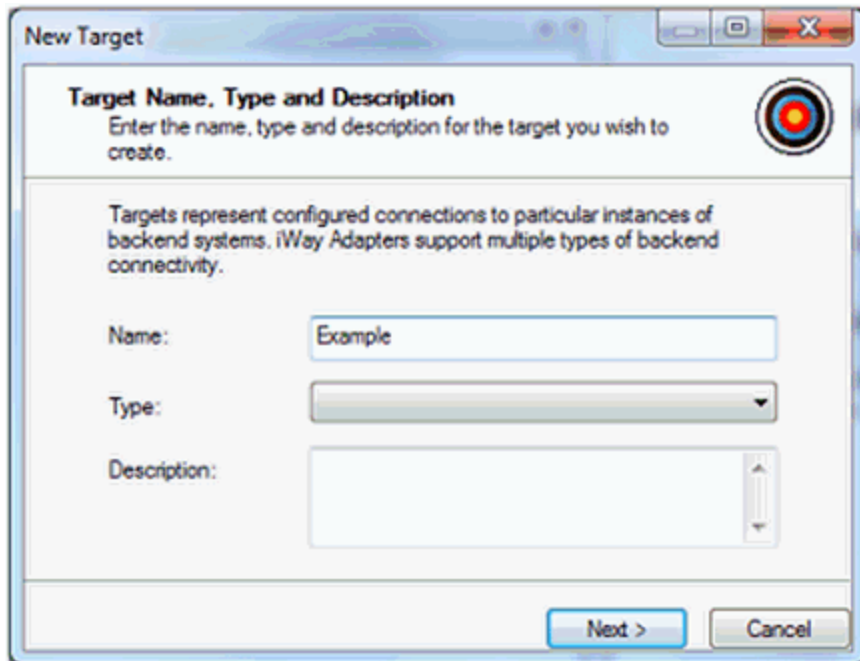


This example will use the following directory path as the Assemblies' Directory:

`C:\Example\bin`

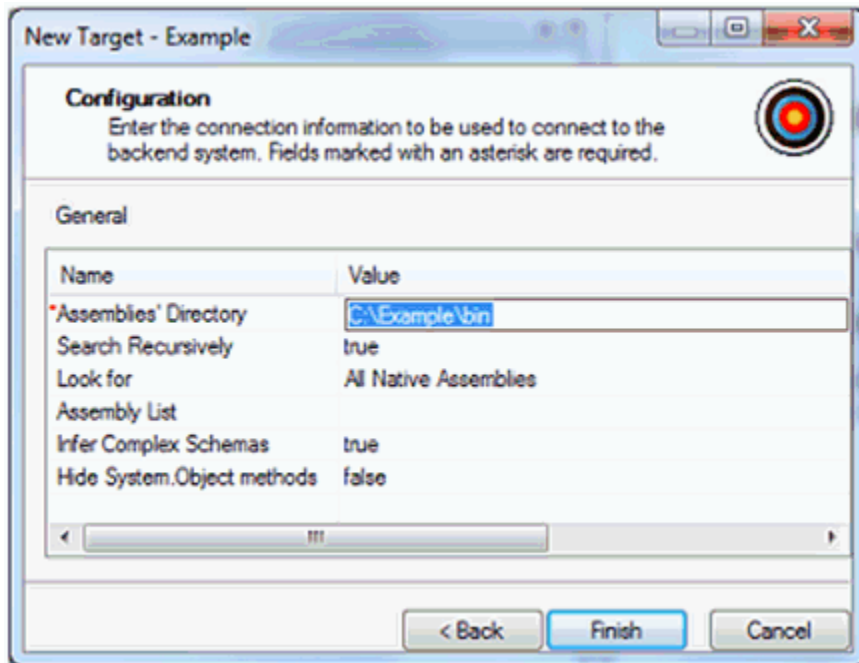
This is the path that must be specified in the Assemblies' Directory field during the creation of the adapter target using iWay Explorer.

The following image shows the New Target dialog box that opens when you create a target for the iWay .NET Technology Adapter.



Specify Example as the name of this new adapter target and click Next.

The Configuration pane of the New Target dialog box opens, as shown in the following image.

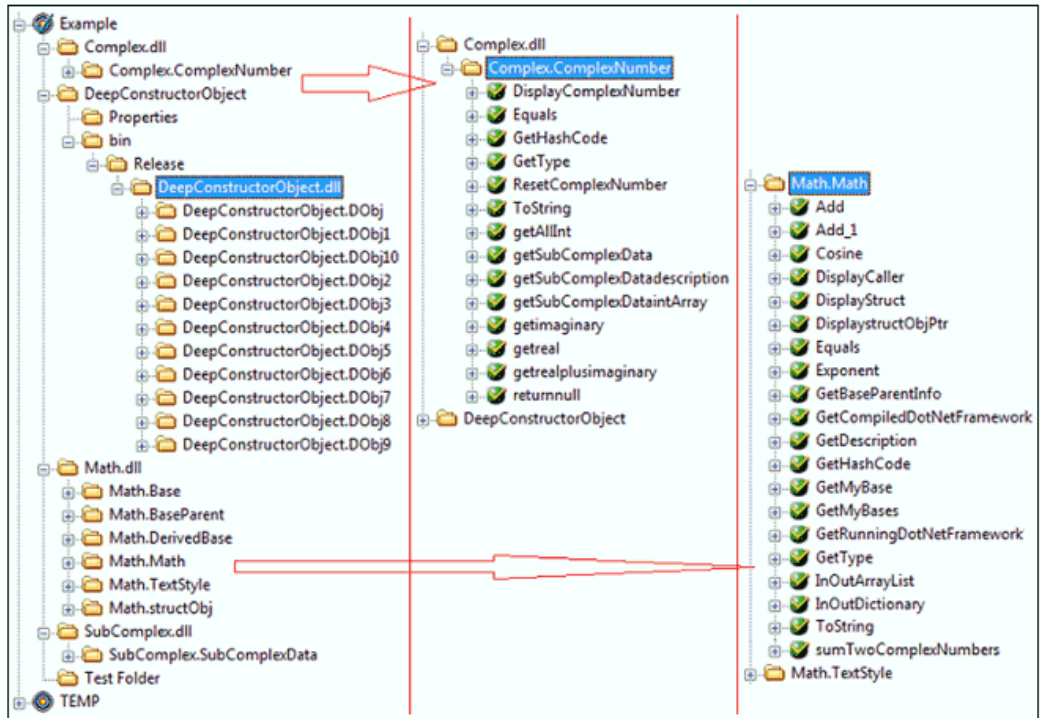


Specify values for the new adapter target as listed in the following table.

Parameter	Value
Assemblies' Directory	C:\Example\bin
Search Recursively	true
Look for	All Native Assemblies
Assembly List	
Infer Complex Schemas	true
Hide System.Object methods	false

Click *Finish*.

The new adapter target node called *Example* is added under the Adapters node of iWay Explorer, as shown in the following image.



In the above compiled image, the Example adapter target contains the following Assemblies:

- ☐ Complex.dll
- ☐ DeepConstructorObject.dll
- ☐ Math.dll
- ☐ SubComplex.dll

The Complex.dll Assembly supports one public class called *Complex.ComplexNumber*. The *Complex.ComplexNumber* class has fourteen methods. Since the value of the *Hide System.Object methods* parameter was set to *false* during the adapter target creation process, the inherited *System.Object* methods are also displayed.

The Math.dll Assembly supports the following public classes:

- ☐ Math.Base

- ☐ Math.BaseParent
- ☐ Math.DerivedBase
- ☐ Math.Math
- ☐ Math.TextStyle
- ☐ Math.structObj

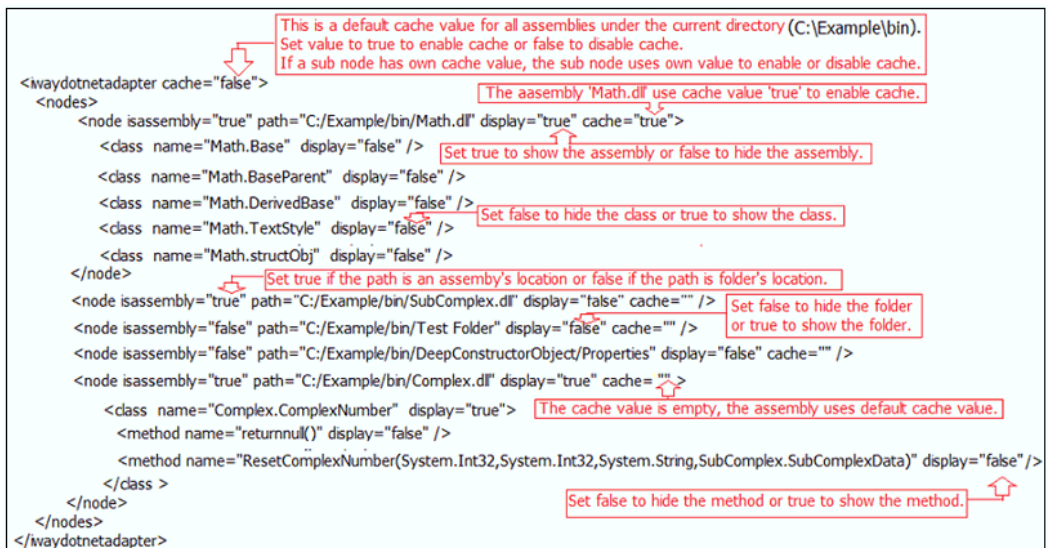
The *Math.Math* class contains 21 methods (including the *Equals*, *GetHashCode*, *GetType*, and *ToString* methods).

Display Assemblies, Hiding Classes and Methods, and Disabling Cache

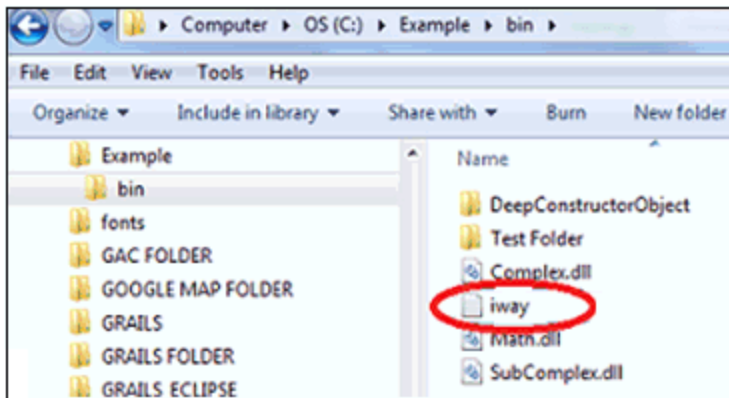
Using the *iway.xml* properties file, this example demonstrates how to:

- ☐ Display the *Math.dll*, *DeepConstructorObject.dll*, and *Complex.dll* Assemblies.
- ☐ Hide the *Math.BaseParent*, *Math.TextStyle*, *Math.structObj*, *Math.DerivedBase*, and *Math.Base* classes.
- ☐ Hide the *returnnull* and *ResetComplexNumber* methods of the *Complex.ComplexNumber* public class.
- ☐ Disable the cache for all Assemblies except the *Math.dll* Assembly.

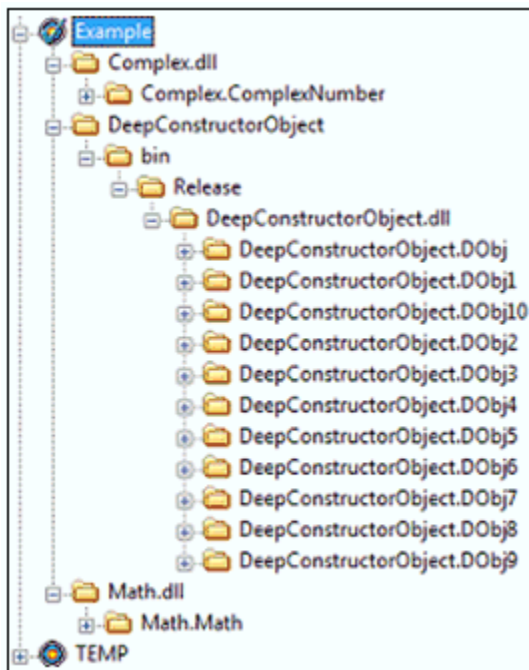
The following image shows a configured *iway.xml* properties file that will implement the conditions listed.



Create an iway.xml properties file based on the above example and save the file to the C:\Example\bin directory, as shown in the following image.

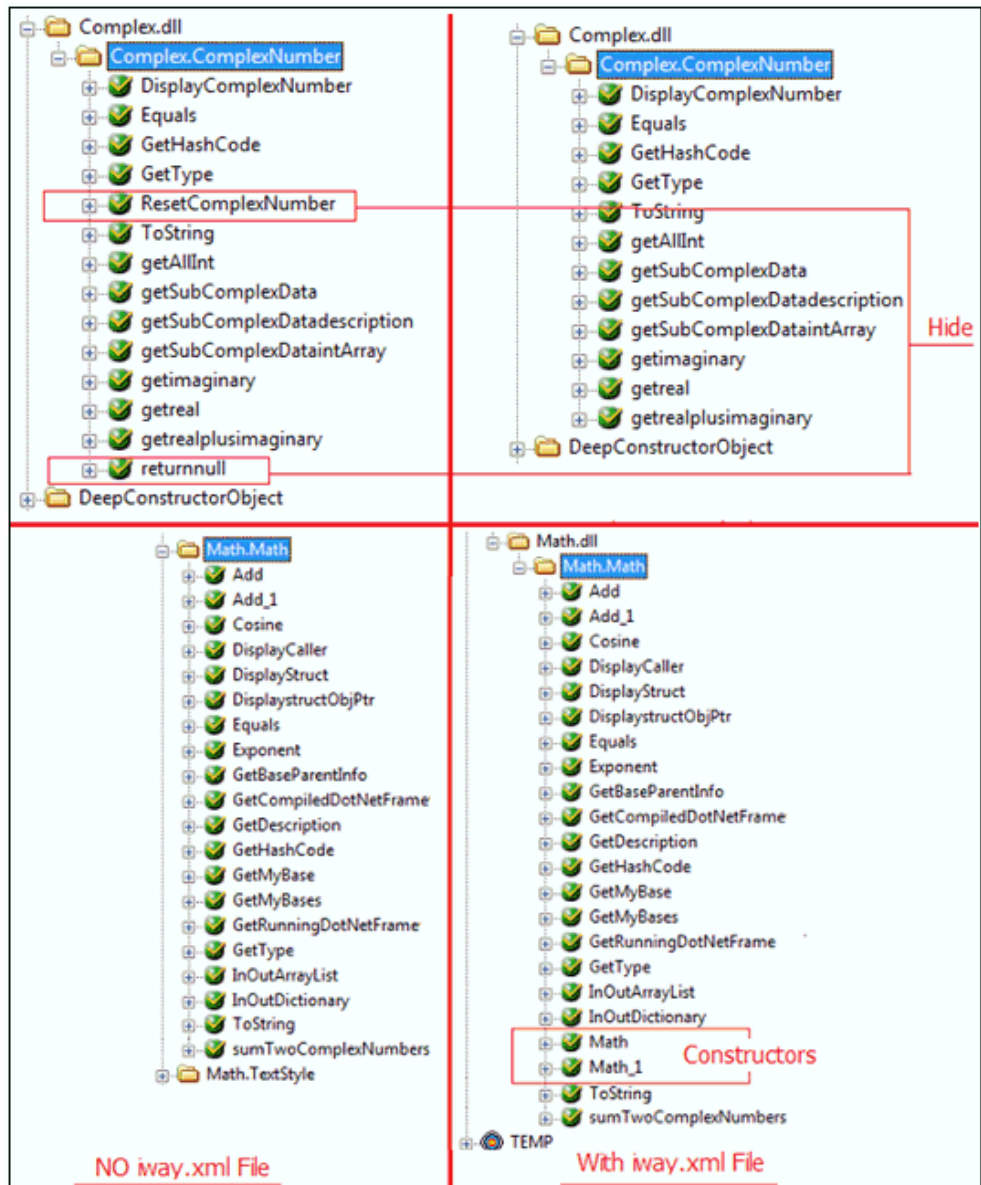


Restart iWay Service Manager (iSM), reconnect to the Example adapter target node, and then expand this node, as shown in the following image.



Notice that only three Assemblies (*Math.dll*, *DeepConstructorObject.dll*, and *Complex.dll*) are displayed. All other folder nodes and Assemblies. In addition, the *Math.Base*, *Math.BaseParent*, *Math.DerivedBase*, *Math.TextStyle*, and *Math.structObj* classes are also hidden.

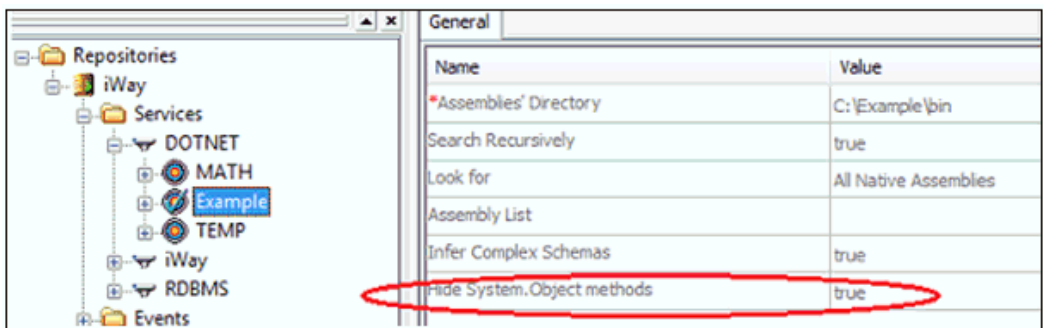
Expand the *Complex.dll* and *Math.dll* Assemblies, as shown in the following image.



The left side of the image shows the hierarchy that is displayed in iWay Explorer when no `iway.xml` properties file is included in the `C:\Example\bin` directory. The right side of the image shows the hierarchy that is displayed in iWay Explorer when the configured `iway.xml` properties file is included in the `C:\Example\bin` directory.

Notice that the `returnnull` and `ResetComplexNumber` methods of the `Complex.ComplexNumber` public class are hidden. Since the `Math.dll` Assembly has its cache value set to `true` (enabling cache), all public constructors of the `Math` class are now displayed.

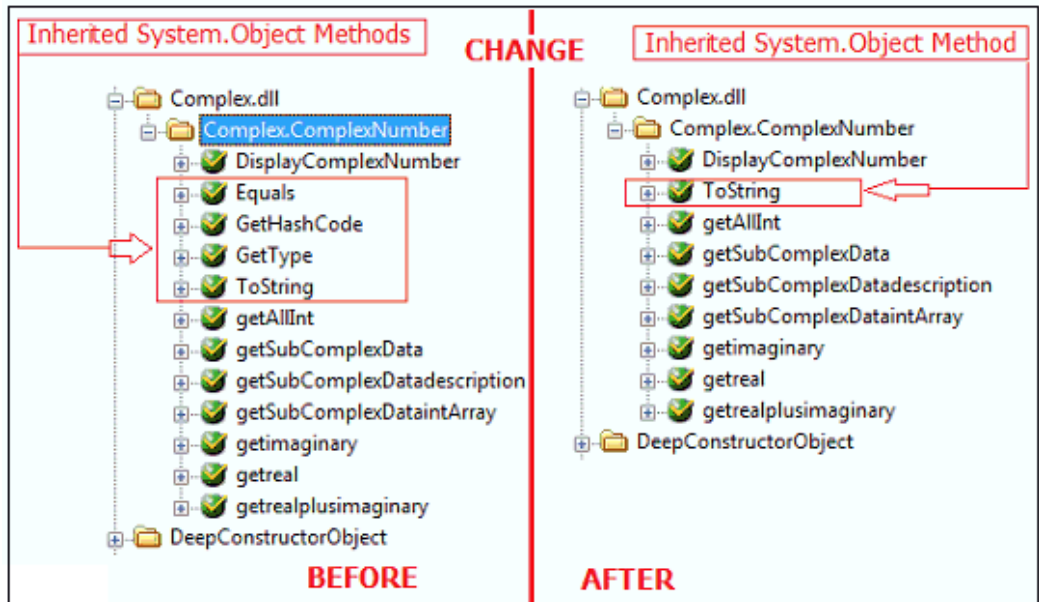
Edit the Example adapter target in iWay Explorer and set the value of the `Hide System.Object methods` parameter to `true`, as shown in the following image.



Modify the existing `iway.xml` properties file, as shown in the following image.

```
<iwaydotnetadapter cache="false">
  <nodes>
    <node isassembly="true" path="C:/Example/bin/Math.dll" display="true" cache="true">
      <class name="Math.Base" display="false" />
      <class name="Math.BaseParent" display="false" />
      <class name="Math.DerivedBase" display="false" />
      <class name="Math.TextStyle" display="false" />
      <class name="Math.structObj" display="false" />
    </node>
    <node isassembly="true" path="C:/Example/bin/SubComplex.dll" display="false" cache="" />
    <node isassembly="false" path="C:/Example/bin/Test Folder" display="false" cache="" />
    <node isassembly="false" path="C:/Example/bin/DeepConstructorObject/Properties" display="false" cache="" />
    <node isassembly="true" path="C:/Example/bin/Complex.dll" display="true" cache="">
      <class name="Complex.ComplexNumber" display="true">
        <method name="returnnull()" display="false" />
        <method name="ToString()" display="true" />
        <method name="ResetComplexNumber(System.Int32,System.Int32,System.String,SubComplex.SubComplexData)" display="false" />
      </class>
    </node>
  </nodes>
</iwaydotnetadapter>
```

Restart iSM, reconnect to the Example adapter target node, and then expand the node for the Complex.dll Assembly, as shown in the following image.



The left side of the image shows the hierarchy that is displayed in iWay Explorer when no modifications are made to the existing iway.xml properties file. The right side of the image shows the hierarchy that is displayed in iWay Explorer when the modified iway.xml properties file is used.

Setting the value of the *Hide System.Object methods* parameter to *true* results in the three inherited System.Object methods (*Equals*, *GetHashCode*, and *GetType*) under the *Complex.ComplexNumber* public class to be hidden. Only the inherited System.Object *ToString* is displayed, since the value of the *display* attribute of the *ToString* method forces the system to have it displayed.

Sample Properties File

For reference purposes, the following is a sample iway.xml properties file that can be customized as required.

```

<iwaydotnetadapter cache="true">
  <nodes>
    <node isassembly="true" path="C:/Program Files (x86)/iway7/etc/samples/
dotnet/bin/Math.dll" display="true" cache="false">
      <class name="Math.Math" display="true">
        <method name="ToString()" display="false"/>
        <method name="Equals(System.Object)" display="true"/>
      </class>
      <class name="Math.Base" display="false">
      </class>
    </node>
    <node isassembly="true" path="C:/Program Files (x86)/iway7/etc/samples/
dotnet/bin/Complex.dll" display="true" cache="false">
      <class name="Complex.ComplexNumber" display="true">
        <method name="ToString()" display="false"/>
        <method name="Equals(System.Object)" display="false"/>
      </class>
    </node>
    <node isassembly="true" path="C:/Program Files (x86)/iway7/etc/samples/
dotnet/bin/SubComplex.dll" display="true" cache="false">
      <class name="SubComplex.SubComplexData" display="true">
        <method name="ToString()" display="false"/>
        <method name="Equals(System.Object)" display="false"/>
      </class>
    </node>
  </nodes>
</iwaydotnetadapter>

```


Run Time Concepts and Configuration Tasks

This section describes run time concepts and configuration tasks for the iWay Technology Adapter for .NET.

Note: Before continuing with run time configuration tasks, refer to the application scope before using the iWay .NET Technology Adapter. For more information, see [Tested Application Scope](#) on page 128.

In this chapter:

- ❑ [Run Time Implementation](#)
 - ❑ [Understanding Design Time and Run Time Targets](#)
 - ❑ [Using the Persist Connection Parameter](#)
 - ❑ [Understanding the Relationship Between Java and .NET](#)
 - ❑ [Invoking the iWay .NET Technology Adapter](#)
-

Run Time Implementation

In the run time, the iWay Technology Adapter for .NET will execute a method of an Assembly by passing XML data based on a request schema.

In order to implement the purpose of running a method, the adapter performs the following steps:

1. Retrieves an instance of the working class, which supports the method by calling *Assembly.CreateInstance*.
2. Discovers parameter information for all parameters of the method by calling *MethodInfo.GetParameters*.
3. Loads and parses the input XML data to retrieve element nodes, which represent values of the parameters in XML format, and deserialize the XML string based on each element node to construct objects for the parameters.
4. Calls *MethodInfo.Invoke* to execute the method and return an object.
5. Serializes the return object, converts the public fields and read/write properties of an object into XML.
6. Generates XML data based on the result XML and the response schema of the method.
7. Returns the final XML data to the caller.

Understanding Design Time and Run Time Targets

Each run time target must point to the Assembly that contains the method being invoked.

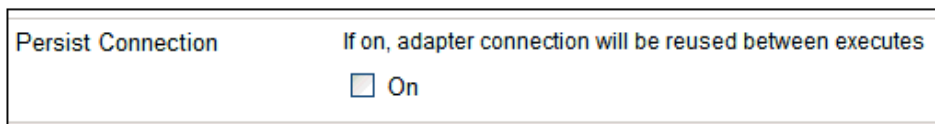
- ☐ process flow adapter object
- ☐ registry adapter object
- ☐ web services adapter object

Persistence is required for cached objects (adapter passivated cache will be released).

While *location* appears in the schema and the instance, it cannot be used at runtime to physically locate the object if the target is pointing to another path.

Using the Persist Connection Parameter

The *Persist Connection* parameter is available for your defined adapter component in the iSM Administration Console, as shown in the following image.



For more information on how to access this parameter, see [Configuring the Adapter in an iWay Environment](#) on page 93.

In a .NET developer project, an external *datasource* object is usually created and embedded inside the Assembly, with an automatic initializer. This is not possible through the iWay Technology Adapter for .NET as the sequence of calls is different. If your Assembly contains the static method *getConnectionstring*, then it can be used to retrieve the connection string and pass the string to a callable object.

If the method initializes a connection object, and the *Persist Connection* parameter is enabled, calling the method again may generate an error indicating that the connection already exists. If the object cannot be changed, then disable the *Persist Connection* parameter to resolve this issue.

If you are using a connection pool, consider using a session with the *Persist Connection* parameter enabled and also using a constructor and cache. The connection will stay initialized throughout the session. If the *Persist Connection* parameter is disabled, then the connection will have to be initialized for each call.

The receiver of a method call is responsible for processing the method call. If the object has not or cannot be initialized, then calling a method without a valid receiver will result in an exception indicating the following:

object not set to an instance of an object

A web service object is not a persisted object. This is the reason cache is not supported on these objects. When an adapter is passivated and retired, all connections are released, so the cache is retired. Only valid instance targets, such as process flow adapter objects can be cached.

Calling the Windows registry when executing from iWay in *Service* mode is governed by the rules of .NET registry retrieval and Windows security settings. This usually means that only results from *HKEY_LOCAL_MACHINE* can be used and the rights from the *Local Administrator* role. For more information, see the topic on C# and the registry in the Microsoft Developer Network (MSDN):

<http://msdn.microsoft.com>

Using .NET configuration files from .NET Assemblies is not supported by .NET. Calling the *Configuration Manager* classes retrieves results from executables only. If there is a requirement to store properties externally, then read and write the properties to an external file from the Assembly method or pass the parameters to the method at invocation time.

Understanding the Relationship Between Java and .NET

The adapter run time has a Java Native Interface (JNI) implementation that uses the C++ language to call the Common Language Runtime (CLR) unmanaged host API, which initializes the runtime host. The runtime host loads the .NET Common Language Runtime instance into a process, creates an application domain within the process, and loads and executes user code within the application domain. The Java Virtual Machine (JVM) through the adapter is the initiator of the process but does not own it. However, when the JVM is shut down, the CLR instance will also shutdown. The iWay Java Native Interface (JNI) layer marshals data between Java and .NET during the life cycle of the adapter. The adapter life cycle may be shorter than the life cycle of the JVM. If the adapter is stopped or disconnected, communication with the hosted CLR instance is terminated. The CLR process will stay active until the JVM is terminated, even if iWay Service Manger is restarted.

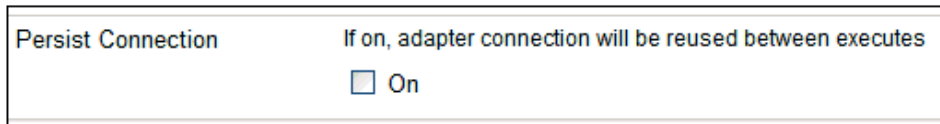
Invoking the iWay .NET Technology Adapter

This section describes how to invoke the iWay .NET Technology Adapter.

Understanding Run Time Types

There are three application run time types:

- ☐ **Static web service target.** Automatic run time target, which has no cached behavior, and is static only. This behavior is the same as an external web service or internal process flow web service.
- ☐ **Defined run time target process flow.** Can be static or cached. This behavior is also controlled by the setting of the *Persist Connection* parameter for your defined adapter component in the iSM Administration Console, as shown in the following image.



For more information on how to access this parameter, see [Configuring the Adapter in an iWay Environment](#) on page 93.

- ☐ **Defined repository channel.** Can be static or cached. This behavior is also controlled by the setting of the *Persist Connection* parameter for your defined adapter component in the iSM Administration Console.

Invoking the iWay .NET Technology Adapter Through an iWay Business Service

This section describes how to invoke the iWay .NET Technology Adapter through an iWay Business Service, using the Hello World sample in the HelloWorld1.dll file.

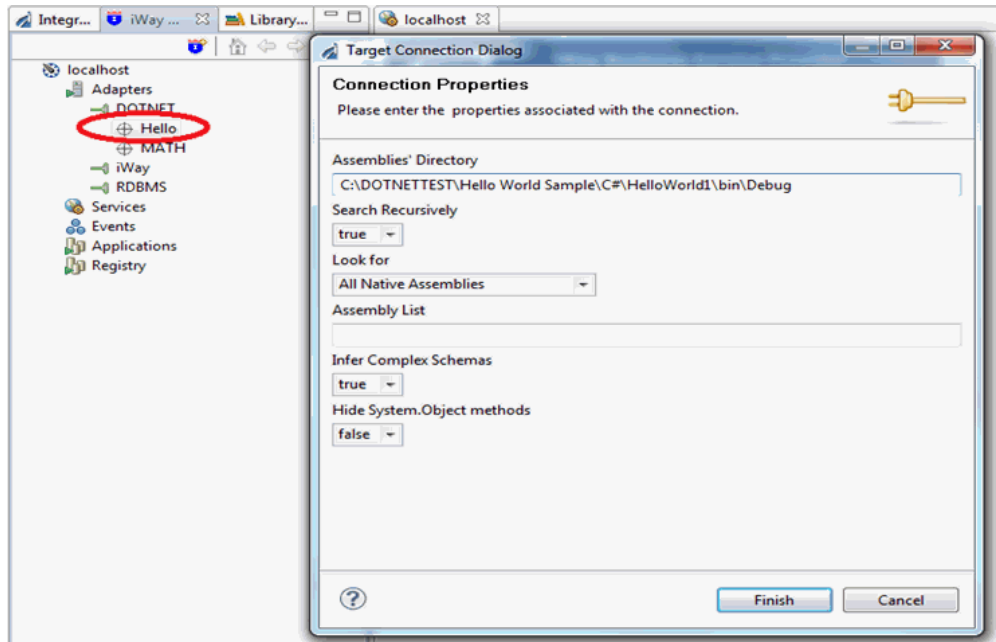
Note: Before exploring a target and creating an integration scenario, check that the scenario is listed in the application scope. For more information, see [Tested Application Scope](#) on page 128. Creating integration scenarios outside of the adapter scope does not qualify for customer support and could destabilize the general iWay working environment.

Procedure: How to Invoke the iWay .NET Technology Adapter Through an iWay Business Service

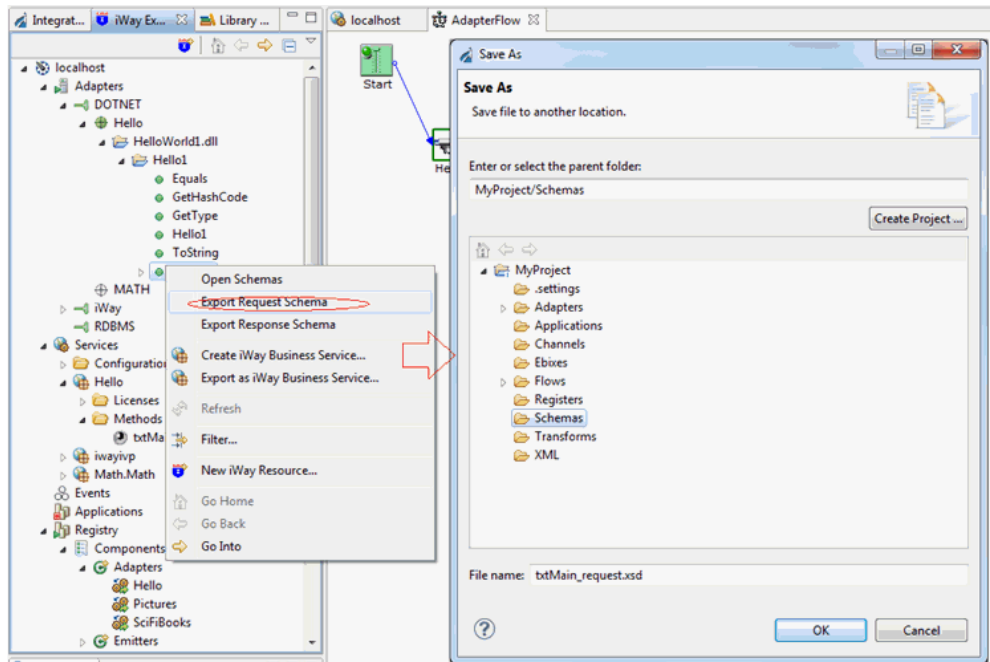
To invoke the iWay .NET Technology Adapter:

1. Using iWay Explorer, add a new target for the adapter (for example, *Hello*).

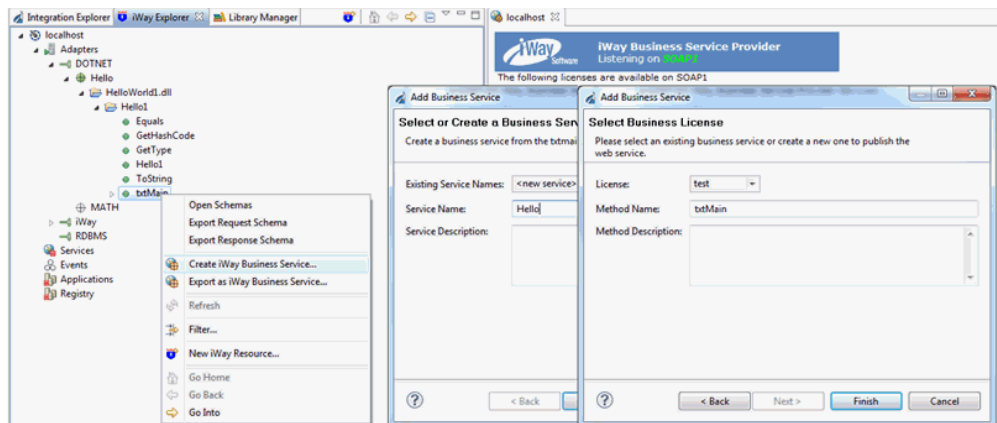
The Target Connection Dialog appears, as shown in the following image.



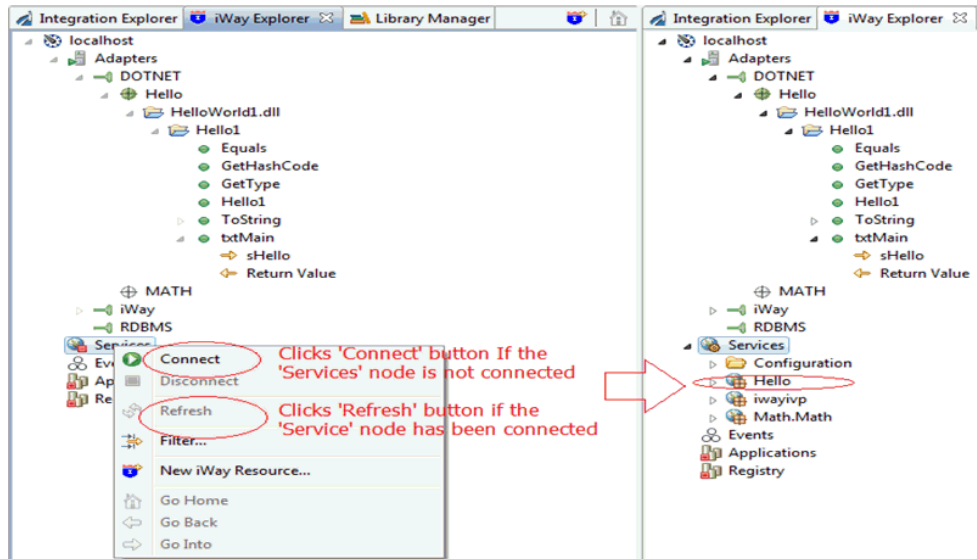
2. Export a request schema with a method (for example, *txtMain*) to an integration project, as shown in the following image.



3. Create an iWay business service (for example, *Hello*) for the method created in the previous step (for example, *txtMain*), as shown in the following image.

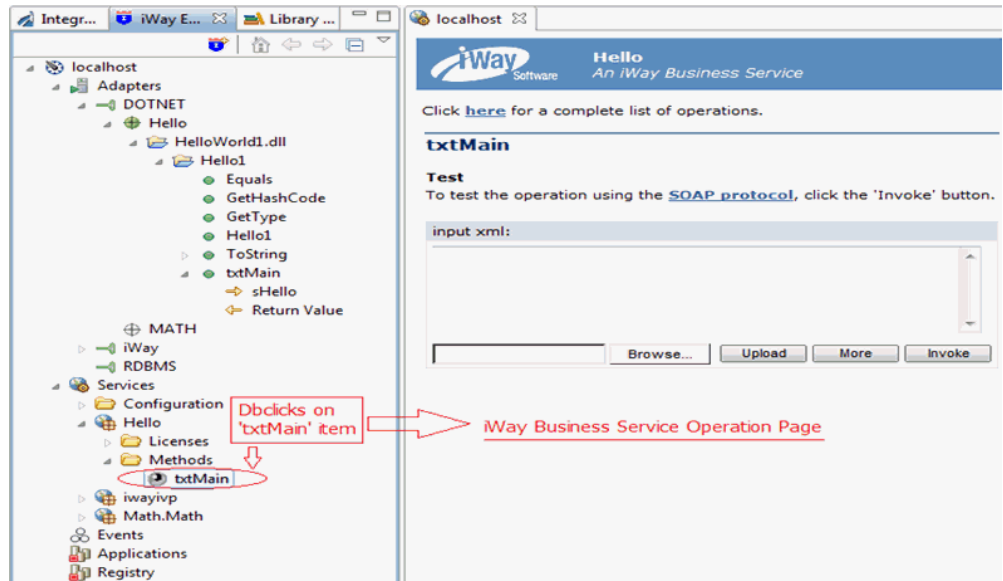


- Display the newly created service (for example, *Hello*), by right-clicking the Services node and selecting *Connect* from the menu if the Services node is not connected, or *Refresh* if the Services node has already been connected.

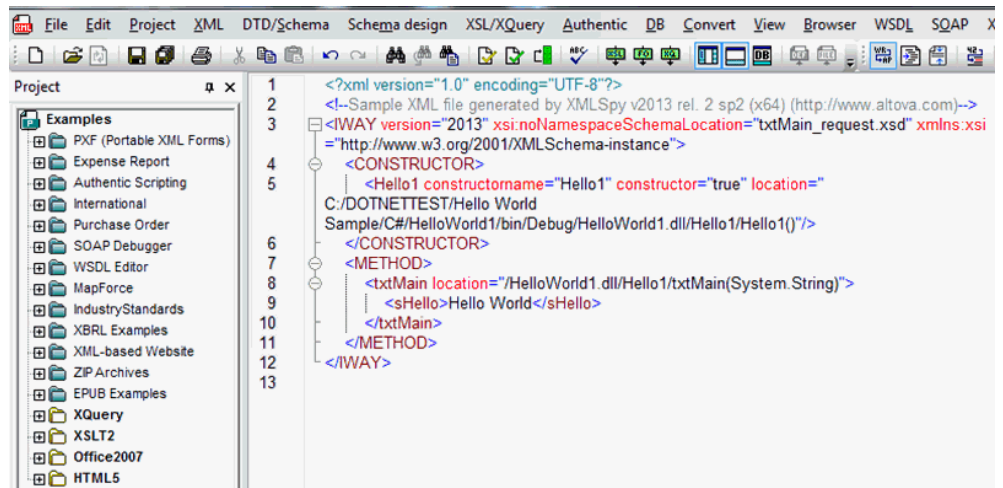


- Expand the newly created *Hello* service, expand the *Methods* folder, and then double-click the method you created (for example, *txtMain*).

The iWay Business Service Operation page appears, as shown in the following image.

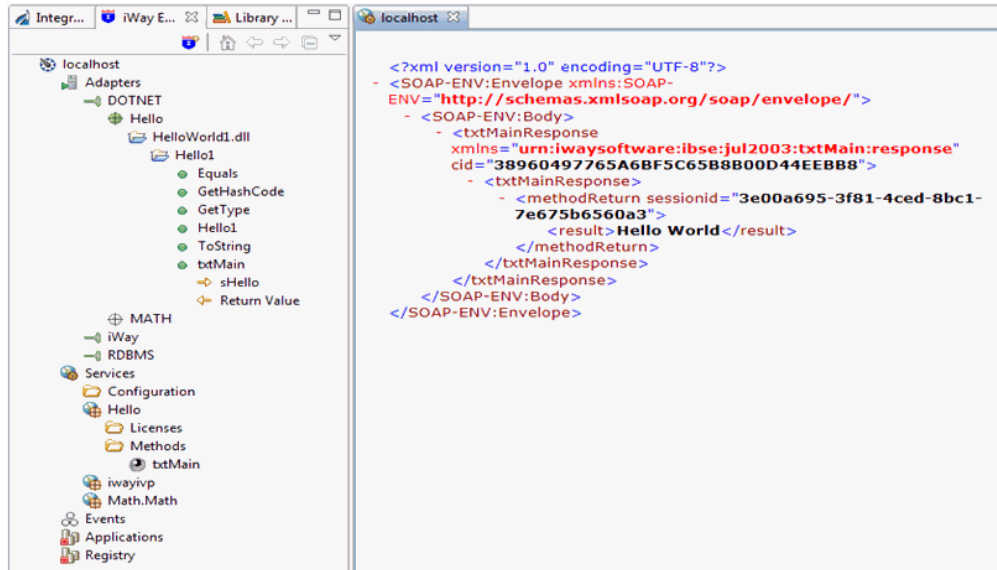


- Export a request schema to generate and modify a sample XML file using an XML editor, such as XmlSpy, as shown in the following image.



- Copy and paste the sample XML file to the *input XML* folder of the iWay Business Service Operation page, and then select the *Invoke* option.

The result is displayed, as shown in the following image.



Invoking the iWay .NET Technology Adapter Through a Process Flow

To invoke the iWay .NET Technology Adapter through a process flow, the process flow must either contain an adapter object which is associated with a target of the iWay .NET Technology Adapter or a web service object which is associated with a business service of the iWay .NET Technology Adapter. Users can directly run the process flow or use a channel to run it.

Note that the input documents that users use to run the adapter process flow in the previous section and web service process flow may be similar but not the same.

- ☐ ATXMLDATA = XMLDATA
- ☐ WSXMLDATA = RTXMLDATA (if the web service object has an empty body).
- ☐ WSXMLDATA = ANYXML (if the web service object has content in its body and the body is formed by RTXMLDATA).

Based from the list above:

ANYXML represents any XML format (for example, <test></test>).

XMLDATA represents the format of the input document which users use to invoke the iWay .NET Technology Adapter through an iWay business service, as shown in the following sample.

```
<IWAY version="2013" xsi:noNamespaceSchemaLocation="txtMain_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CONSTRUCTOR sessionid="String">
    <Hello1 constructormname="Hello1" constructor="true" location="C:/DOTNETTEST/Hello
World Sample/C#/HelloWorld1/bin/Debug/HelloWorld1.dll/Hello1/Hello1()"/>
  </CONSTRUCTOR>
  <METHOD>
    <txtMain location="HelloWorld1.dll/Hello1/txtMain(System.String)">
      <sHello>String</sHello>
    </txtMain>
  </METHOD>
</IWAY>
```

RTXMLDATA represents XMLDATA with extra added to the root element (for example, <ROOT>XMLDATA</ROOT>).

```
<ROOT>
  <IWAY version="2013" xsi:noNamespaceSchemaLocation="txtMain_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <CONSTRUCTOR sessionid="String">
      <Hello1 constructormname="Hello1" constructor="true" location="C:/DOTNETTEST/Hello
World Sample/C#/HelloWorld1/bin/Debug/HelloWorld1.dll/Hello1/Hello1()"/>
    </CONSTRUCTOR>
    <METHOD>
      <txtMain location="HelloWorld1.dll/Hello1/txtMain(System.String)">
        <sHello>String</sHello>
      </txtMain>
    </METHOD>
  </IWAY>
</ROOT>
```

ATXMLDATA represents the format of the input document which users use to invoke the adapter process flow.

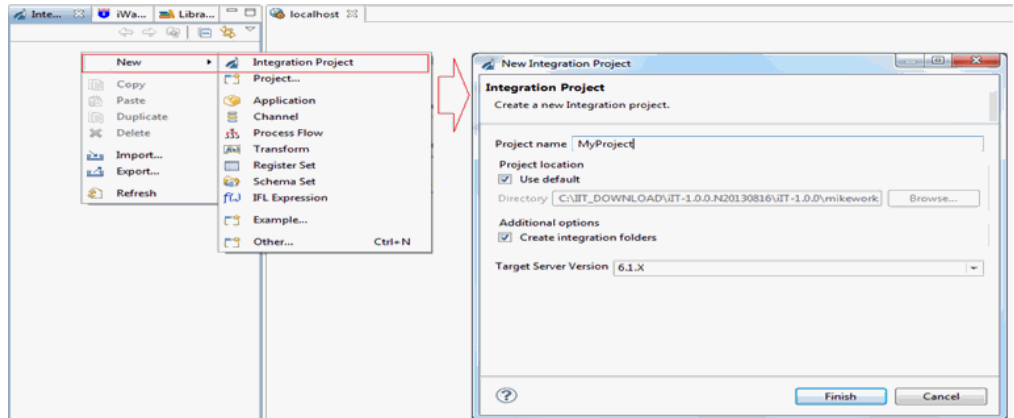
WSXMLDATA represents the format of the input document which users use to invoke the web service process flow.

The following procedures will be using the Hello World sample in the HelloWorld1.dll file.

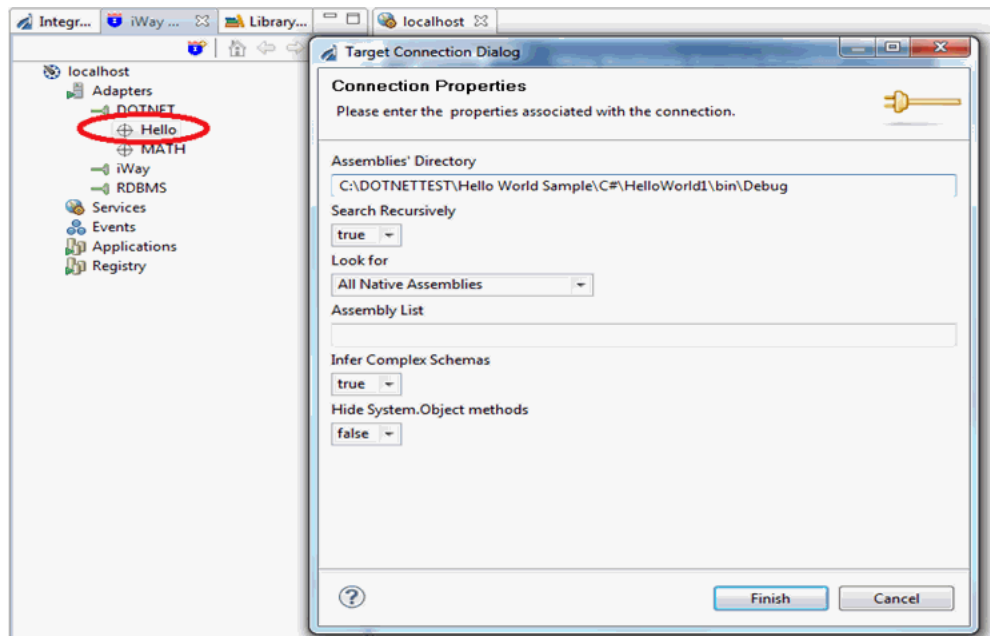
Procedure: How to Run an Adapter Process Flow to Invoke the iWay .NET Adapter

To run an adapter process flow to invoke the iWay .NET Technology Adapter:

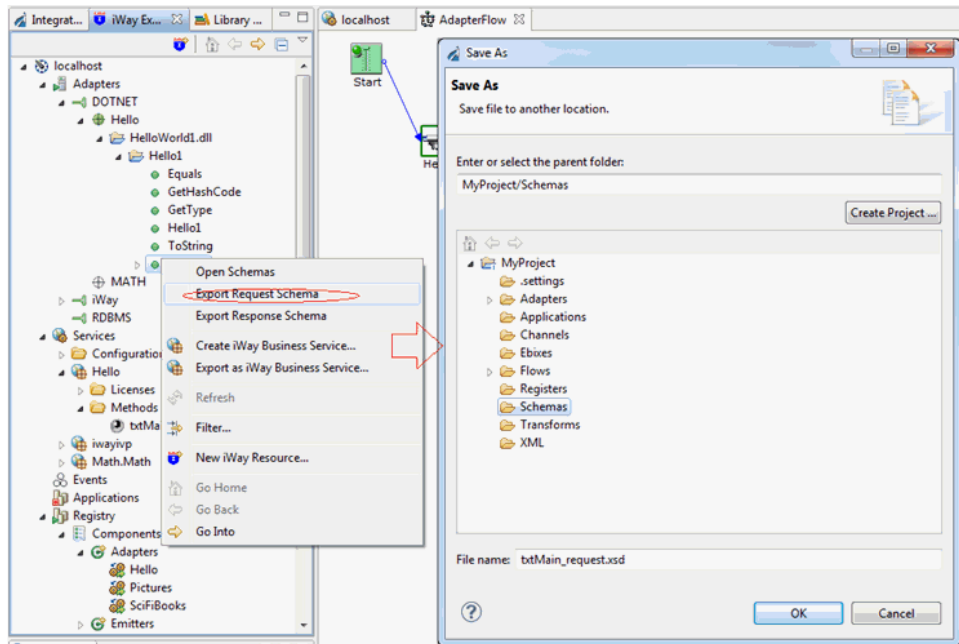
1. Create a new integration project (for example *MyProject*), using Integration Explorer, as shown in the following image.



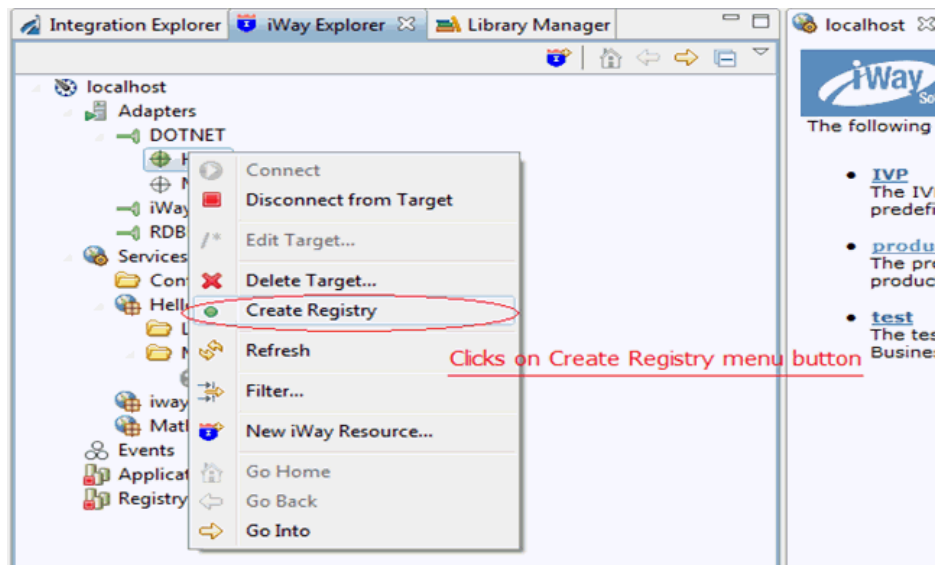
2. Add a new target (for example, *Hello*) for the iWay .NET Technology Adapter using iWay Explorer, as shown in the following image.



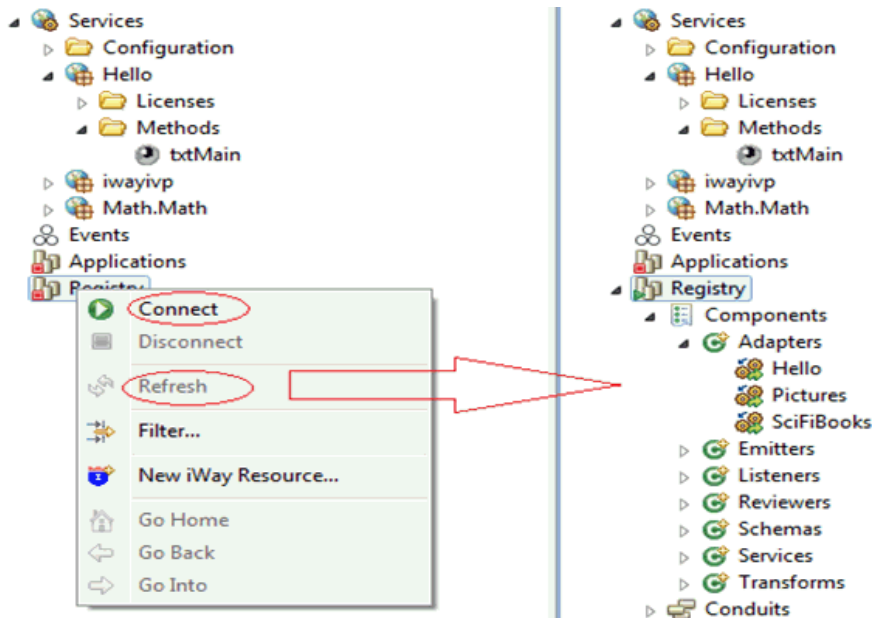
3. Export a request schema with a method (for example, *txtMain*) to an integration project, as shown in the following image.



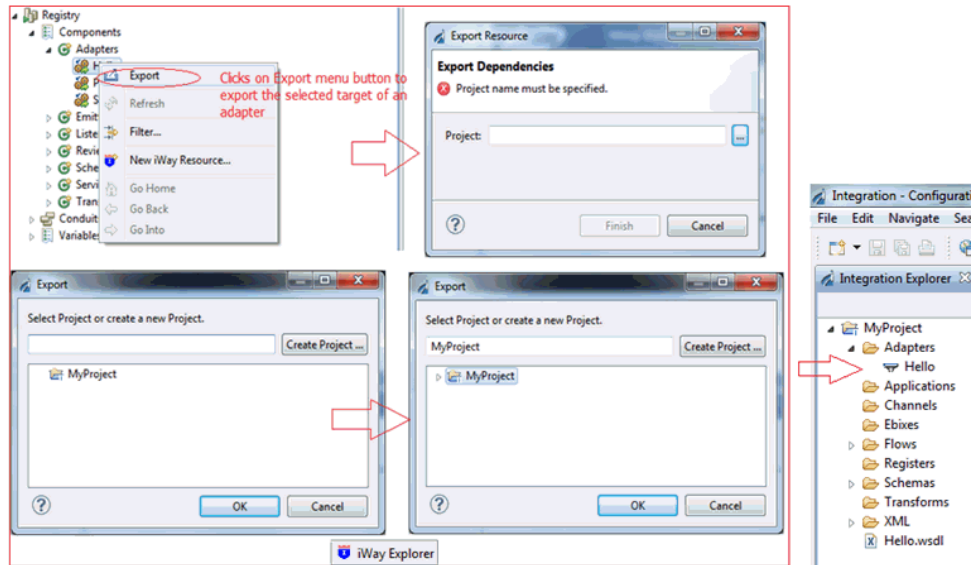
4. Create a registry for the target by right-clicking *Hello* and then selecting *Create Registry* from the menu list, as shown in the following image.



The newly created registry item, *Hello*, will be added under the Adapters node, after expanding the Components object from the Registry node. Users can expand the Registry node by right-clicking on Registry, and selecting the *Connect* or *Refresh* option from the menu list to view it, as shown below.

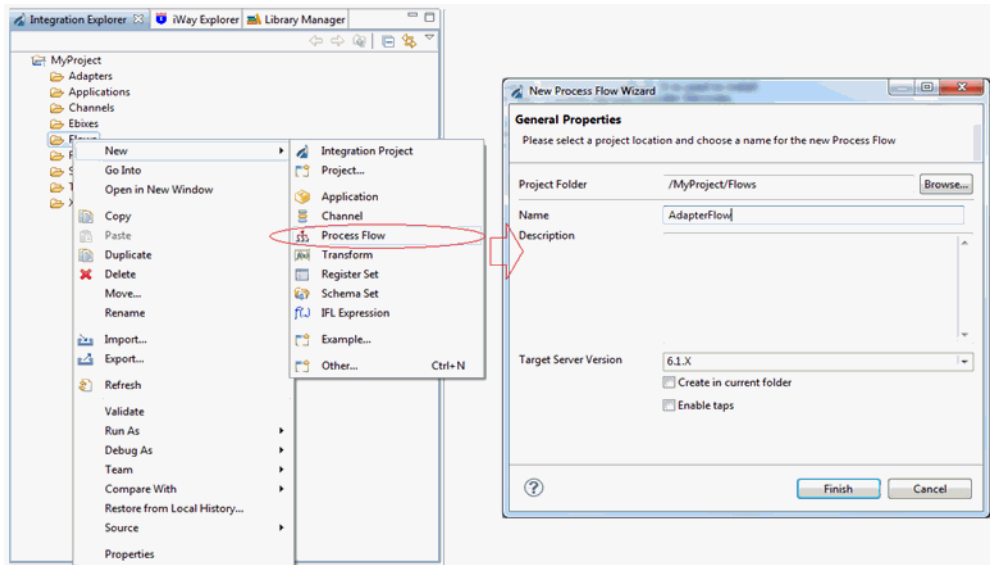


- Export the newly created registry item, *Hello*, to the newly created integration project, *MyProject*, as shown in the following image.

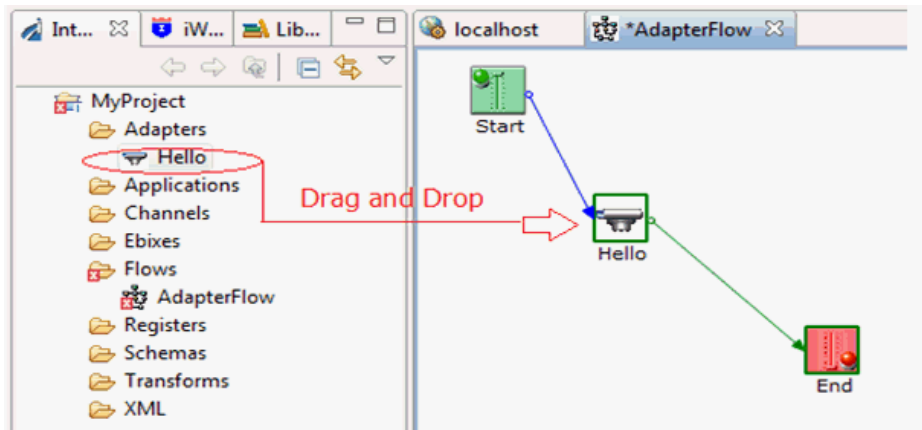


- Create a new process flow by right-clicking *Flows*, selecting *New* from the menu list, and then clicking *Process Flow*.

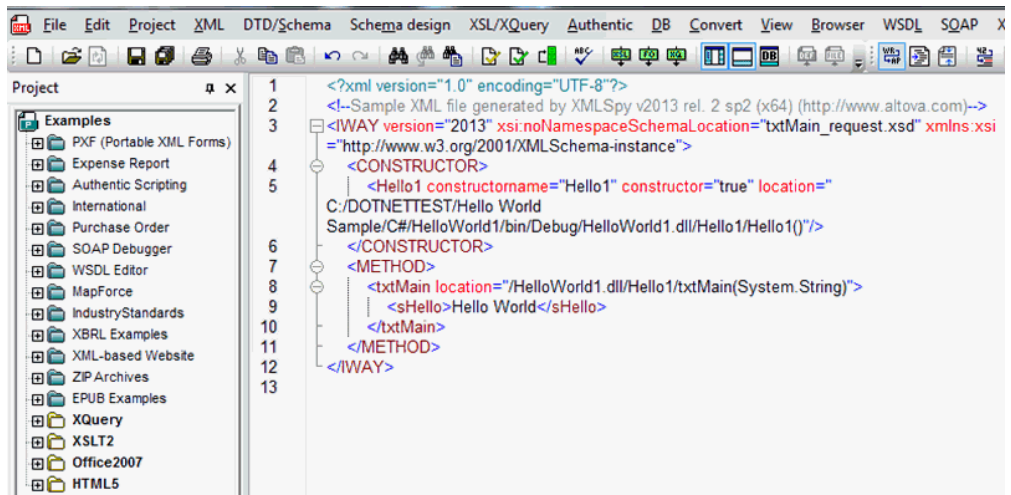
The New Process Flow Wizard appears, as shown in the following image.



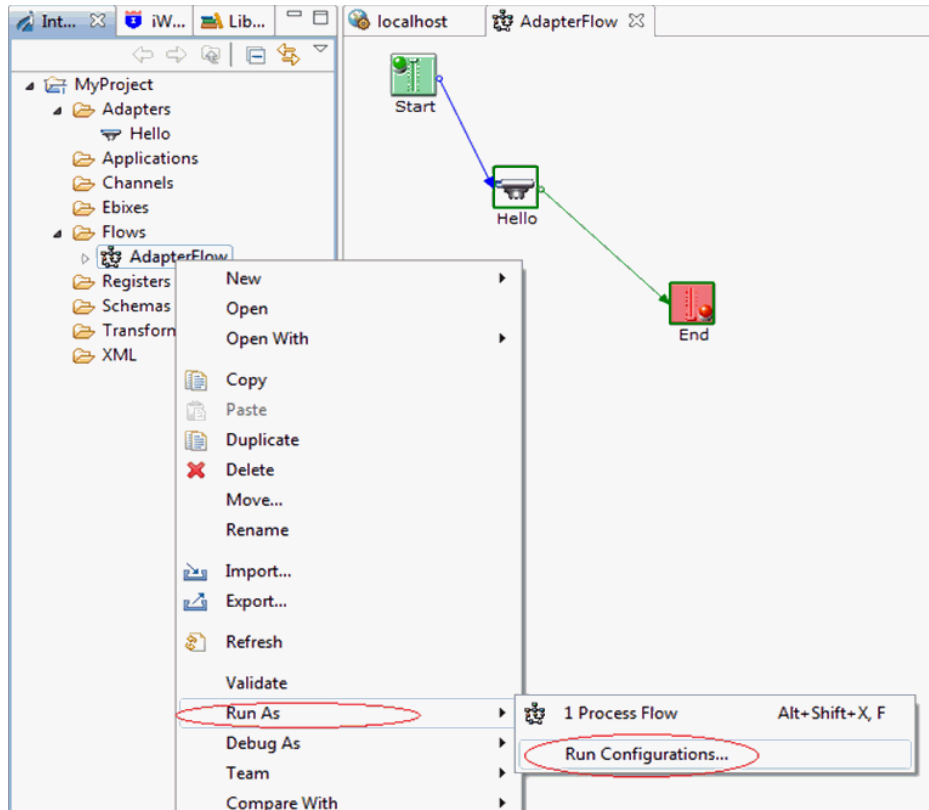
7. Using Integration Explorer, create a process flow that connects the *Start* object to the adapter (*Hello*), and then the adapter to the *End* object, as shown in the following image.



8. Use the exported request schema from Step 3 to generate and modify a sample XML file using an XML editor, such as XmlSpy, as shown in the following image.

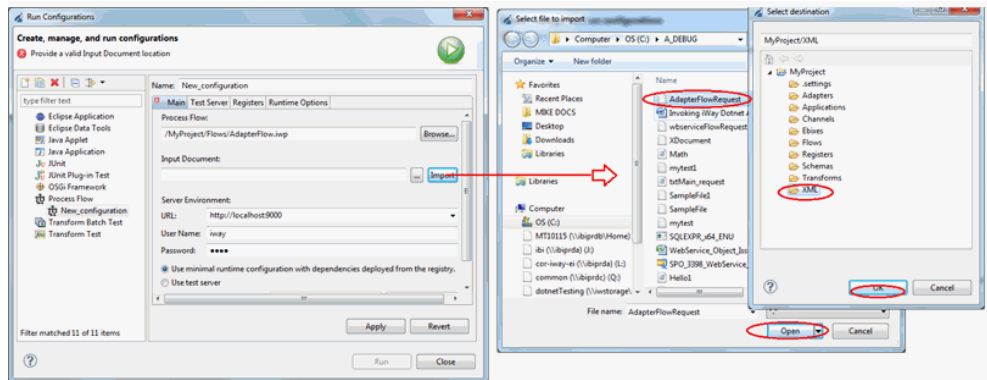


9. In the Integration Explorer tab, expand the Flows folder, and right-click *AdapterFlow*, select *Run As* from the menu list, and click *Run Configurations*, as shown in the following image.

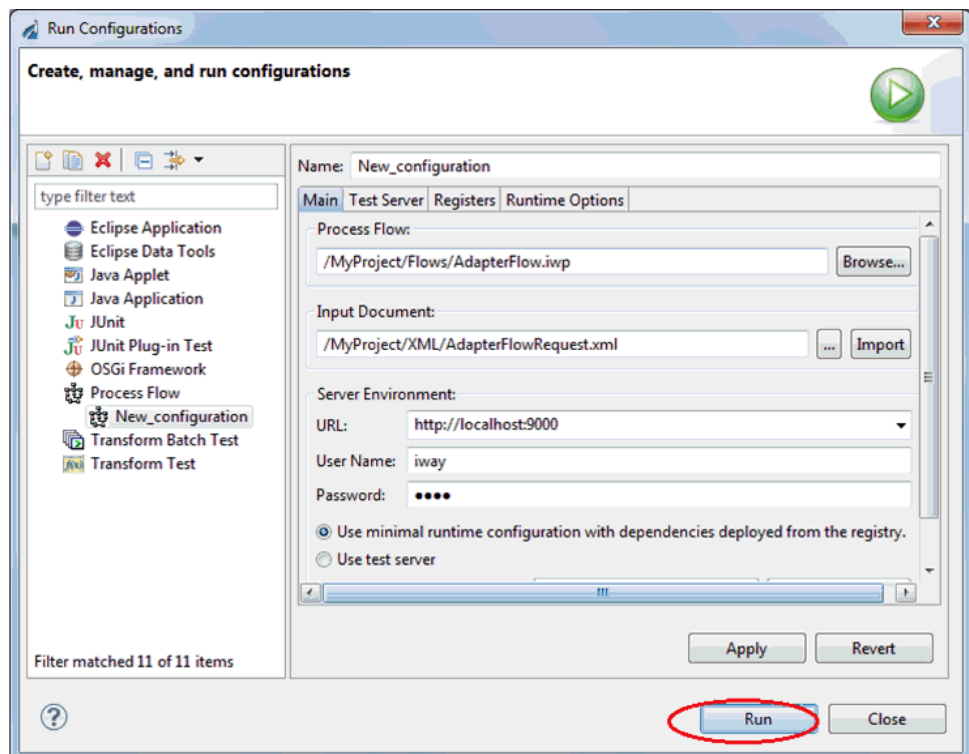


The Run Configurations window appears.

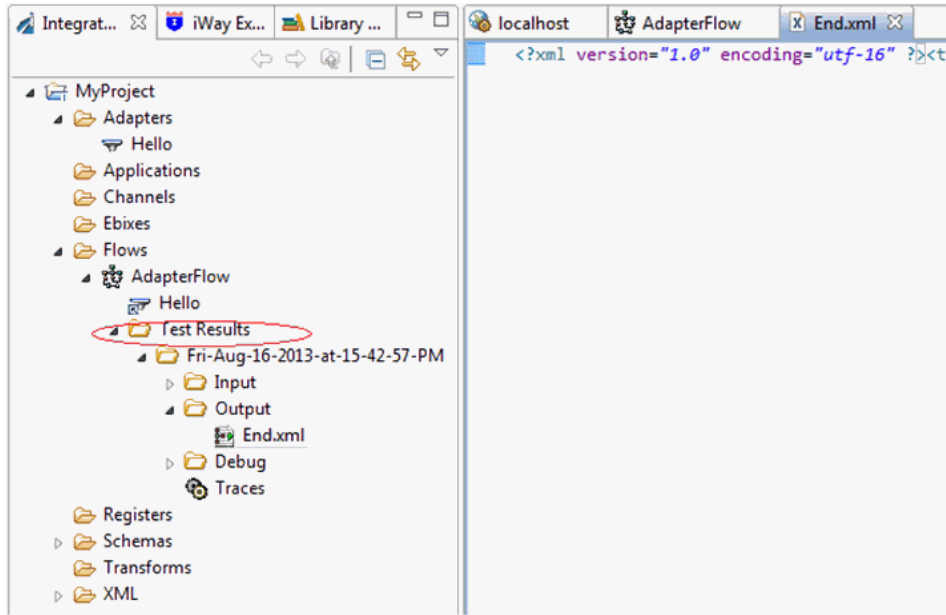
10. In the Input Document field, click the *Import* button and import the generated sample XML file used in Step 8.



11. Run the process flow after it has successfully compiled, as shown in the following image.



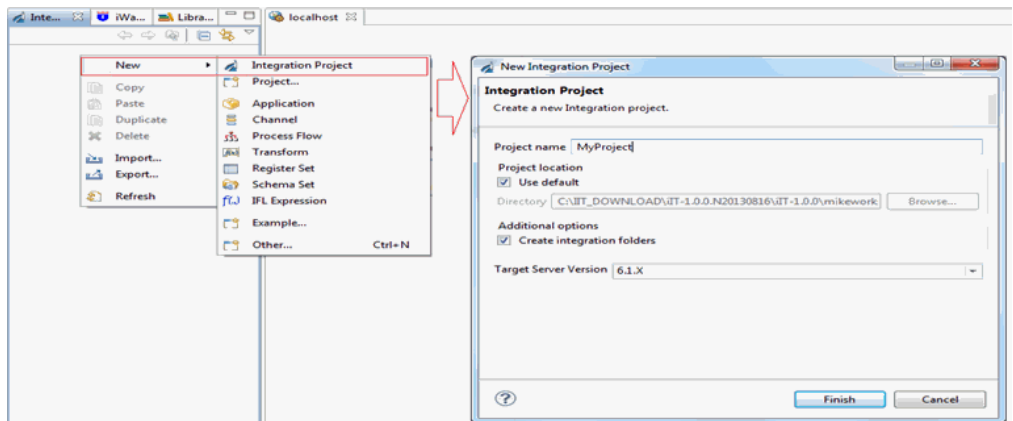
The result will be stored in a folder under Test Results, as shown in the image below.



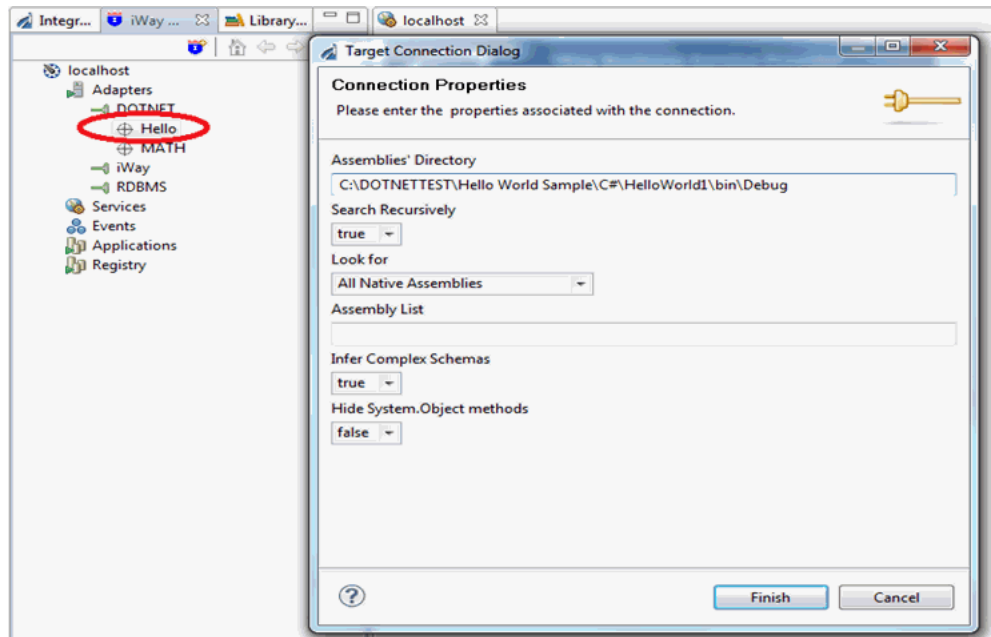
Procedure: How to Run a Web Service Process Flow to Invoke the iWay .NET Adapter

To run a web service process flow to invoke the iWay .NET Technology Adapter:

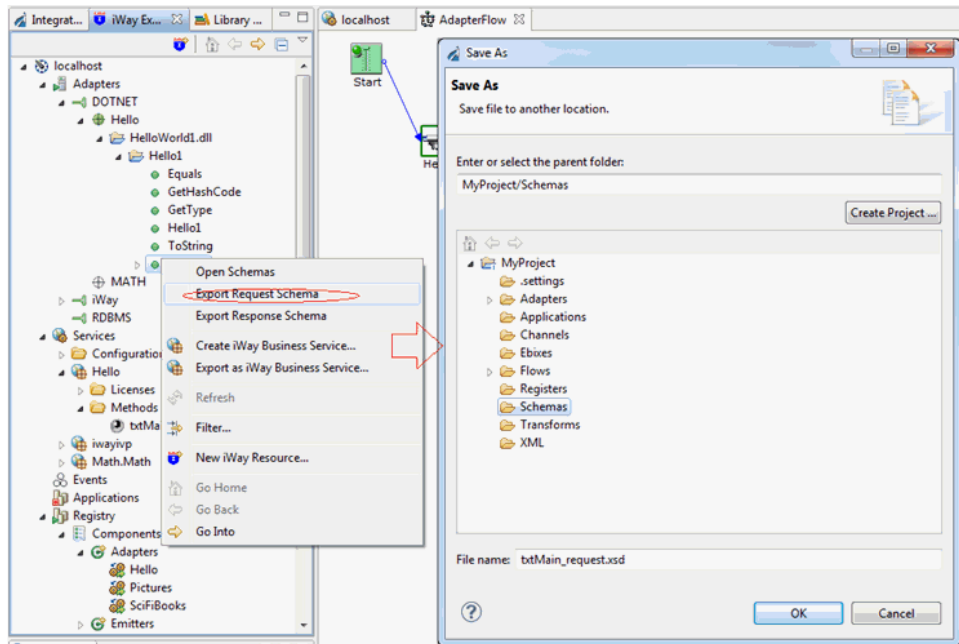
1. Create a new integration project (for example *MyProject*), using Integration Explorer, as shown in the following image.



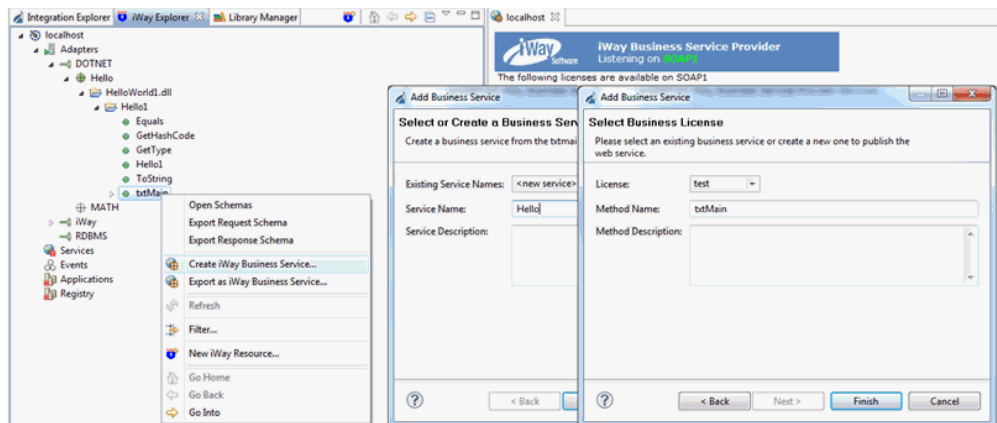
2. Add a new target (for example, *Hello*) for the iWay .NET Technology Adapter using iWay Explorer, as shown in the following image.



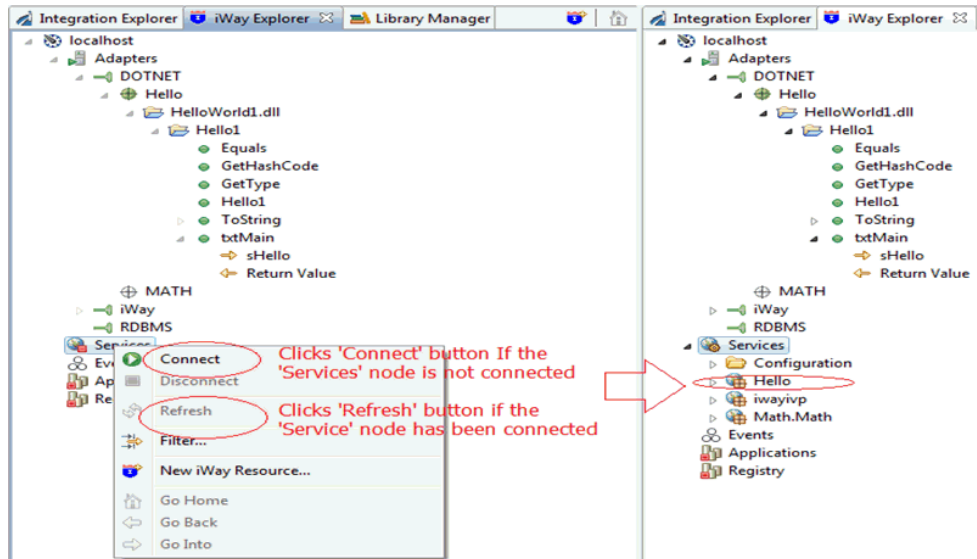
3. Export a request schema with a method (for example, *txtMain*) to an integration project, as shown in the following image.



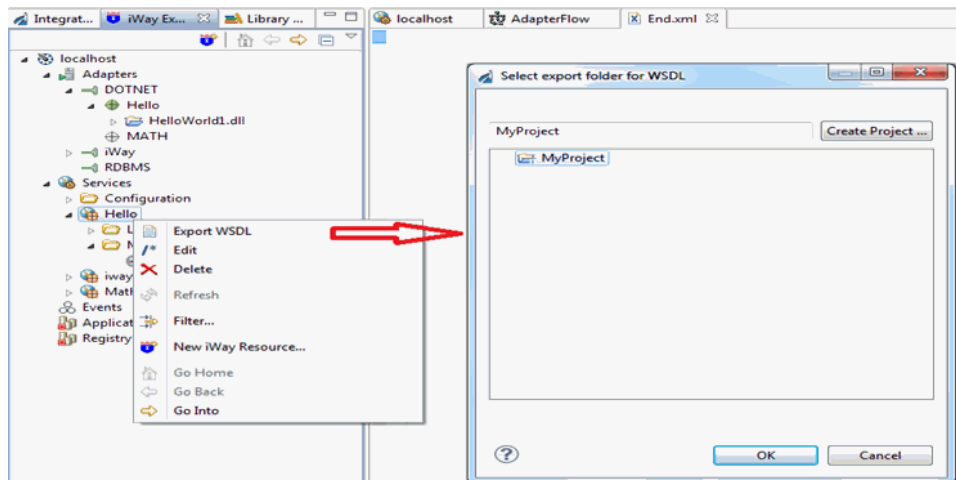
4. Create an iWay business service (for example, *Hello*) for the method created in the previous step (for example, *txtMain*), as shown in the following image.



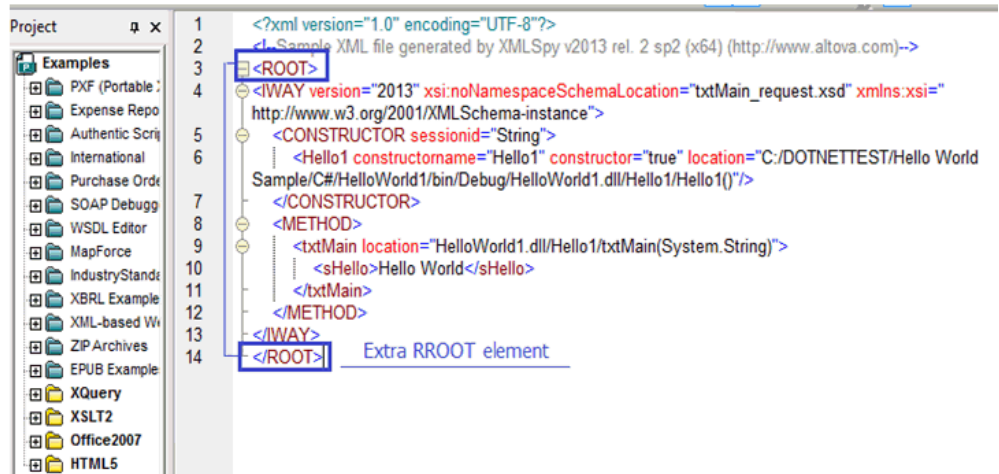
- Display the newly created service (for example, *Hello*), by right-clicking the Services node and selecting *Connect* from the menu list if the Services node is not connected, or *Refresh* if the Services node has already been connected.



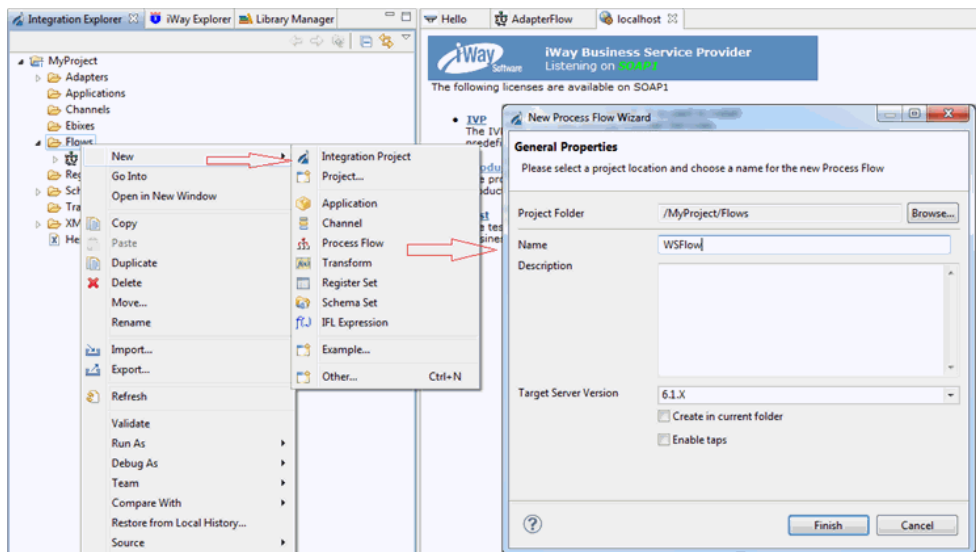
- Export the newly created iWay business service item, *Hello*, to the newly created integration project, *MyProject*, as shown in the following image.



7. Use the exported request schema from Step 3 to generate and modify a sample XML file using an XML editor, such as XmlSpy, and add extra ROOT elements to the XML file, as shown in the following image.

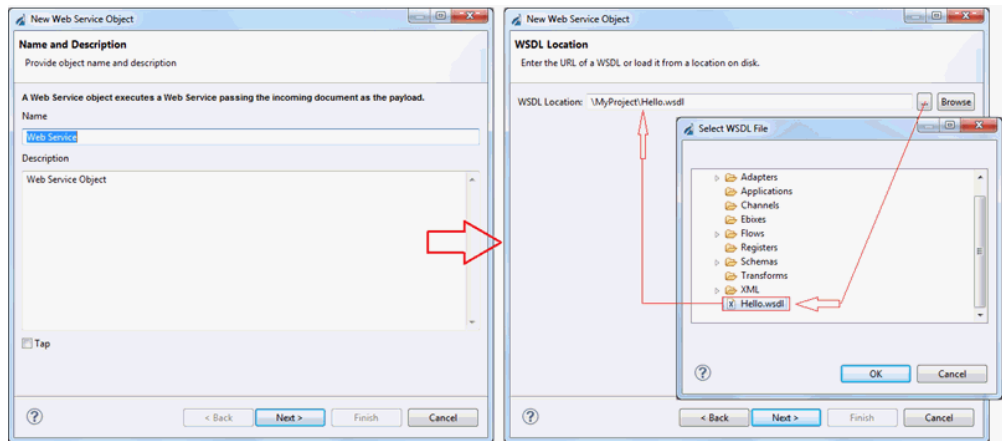


8. In the Integration Explorer tab, expand the Flows folder, and right-click *AdapterFlow*, select *New* from the menu list, and click *Process Flow*, as shown in the following image.

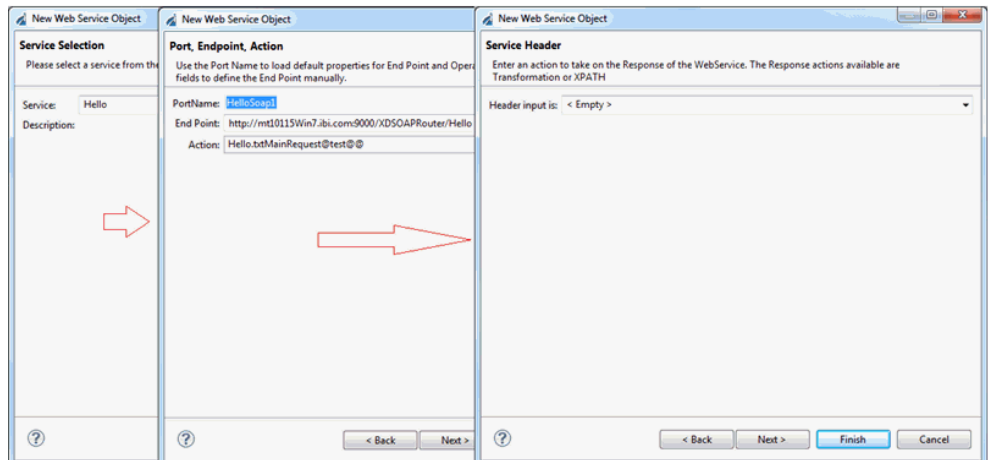


9. Provide a name and description for the new process flow and click *Next*.

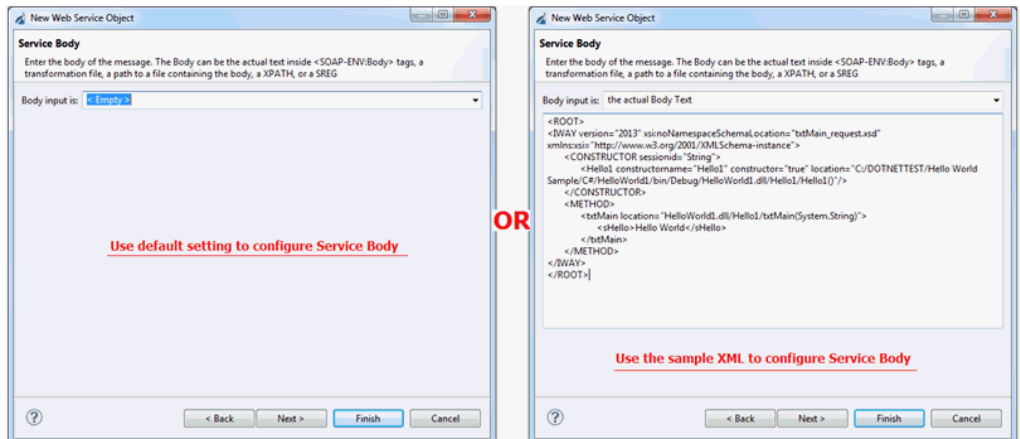
10. For the WSDL Location field, click *Browse* and select the iWay business service (for example, Hello.wsdl), as shown in the following image.



11. Click *OK* and then *Next* to continue.
12. In the Service Selection pane, provide a Service name (for example, Hello), and click *Next*.
13. In the Port, Endpoint, Action pane, provide a port name (for example, HelloSoap1), and click *Next*.
14. In the Service Header pane, select a header input from the drop-down list, as shown in the following image.

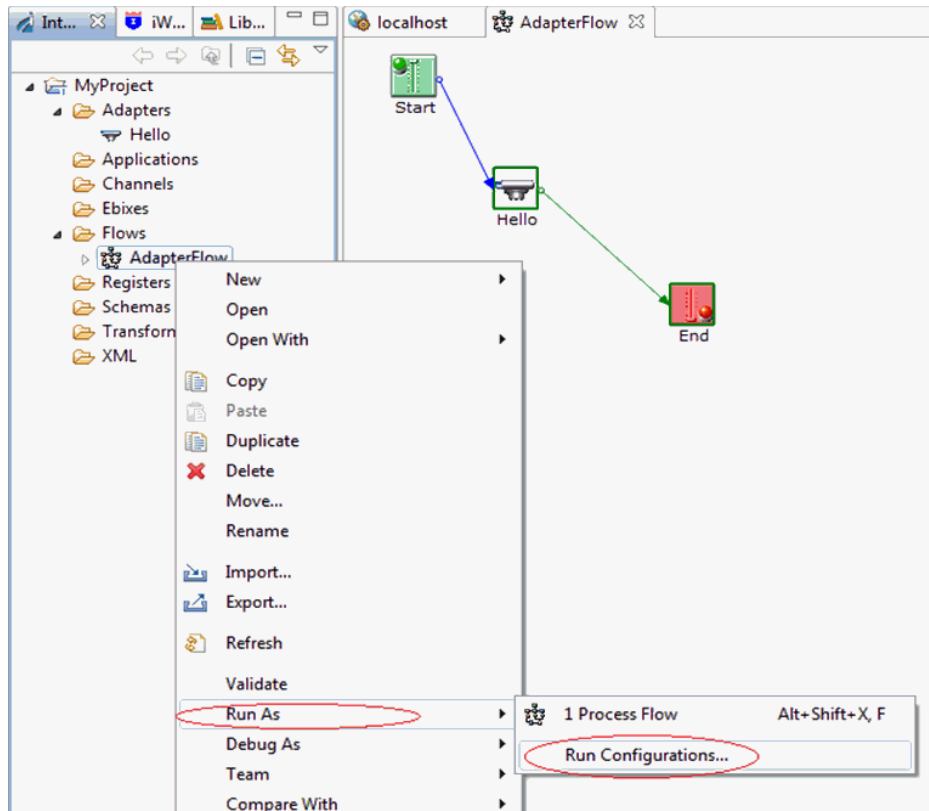


15. Use either the default setting (empty service body) or the sample XML to configure the Service Body of the web service object, as shown in the following image.



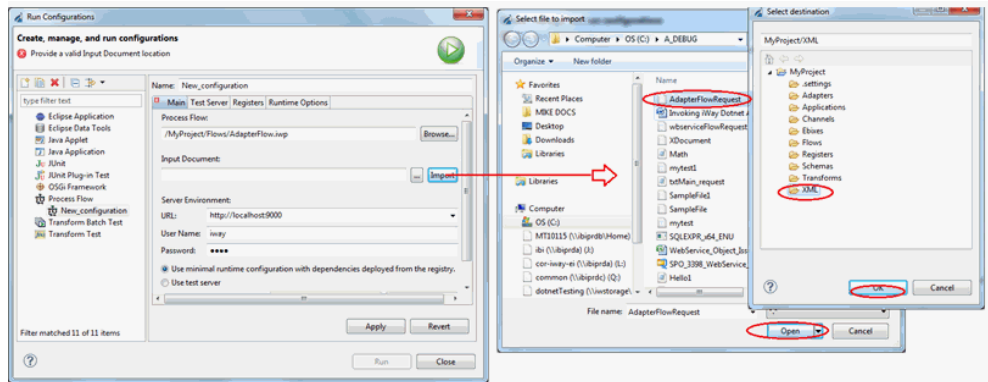
16. If the web service object has an empty body, use the generated sample XML file from the previous step. Otherwise, use any XML test data (for example, <test></test>).

17. In the Integration Explorer tab, expand the Flows folder, and right-click *AdapterFlow*, select *Run As* from the menu list, and click *Run Configurations*, as shown in the following image.

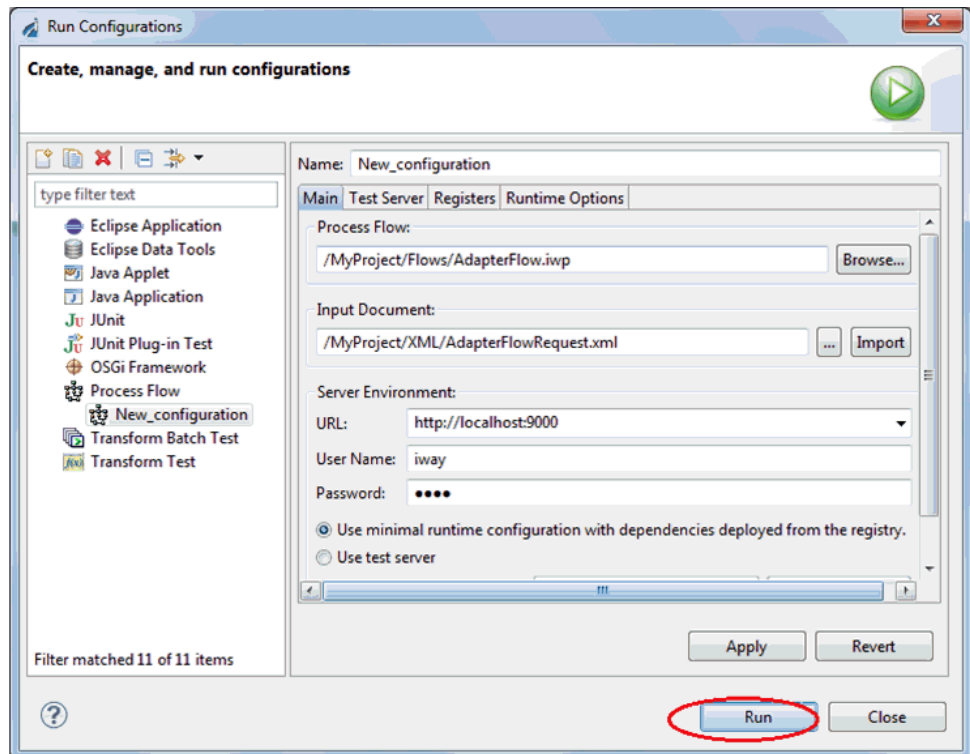


The Run Configurations window appears.

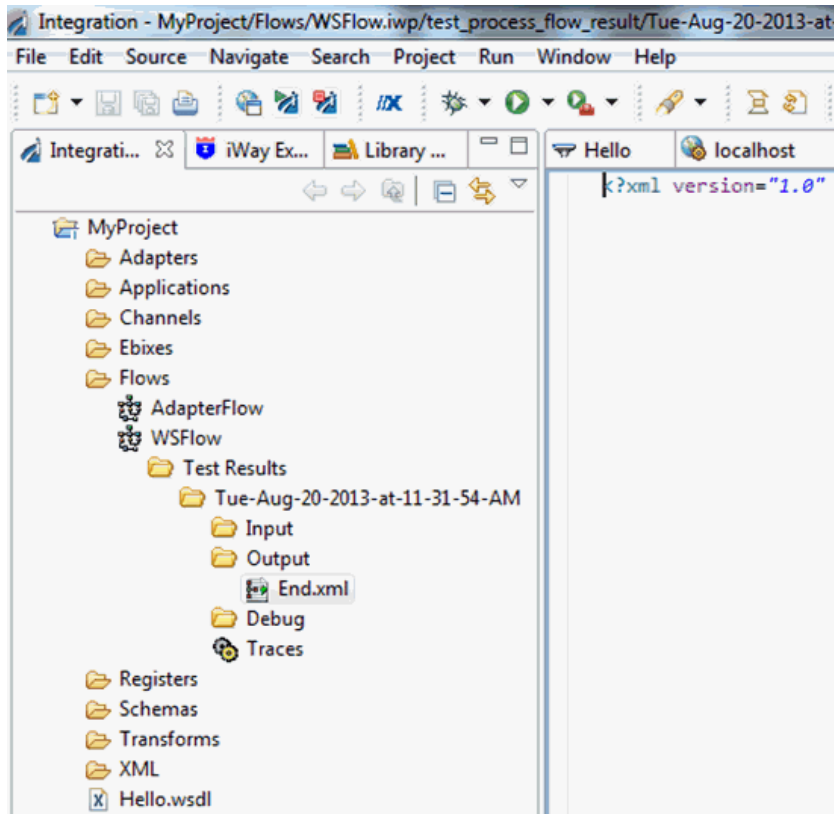
18. In the Input Document field, click the *Import* button and import the generated sample XML file used in Step 7.



19. Run the process flow after it is successfully compiled, as shown in the following image.



The result will be stored in a folder under Test Results, as shown in the image below.



Configuring the Adapter in an iWay Environment

After you successfully configure the adapter to represent a particular adapter target, the adapter can be assigned to a Service Manager listener.

This section also includes information about processing adapter requests.

In this chapter:

- ❑ [Configuring the Adapter in iWay Service Manager](#)

Configuring the Adapter in iWay Service Manager

Before configuring the adapter in iWay Service Manager, you must first create a target, which represents a connection to a backend system, using iWay Explorer. For more information on configuring targets and connections using iWay Explorer, see [Design Time Concepts and Configuration Tasks](#) on page 35 or the *iWay Explorer User's Guide*.

You configure the adapter in the iWay Service Manager console. The configuration process creates run-time connection and persistent data files within Service Manager. The configuration process interrogates the Service Manager repository entries that were built when the target and connection were created using iWay Explorer. The defined adapter process creates the run-time repository based on the design-time repository.

Procedure: How to Define an Adapter

To define an adapter:

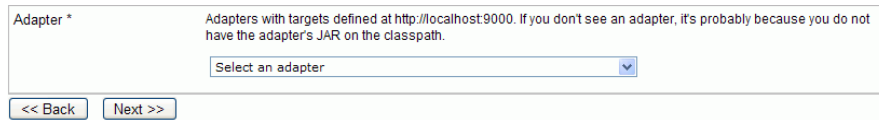
1. In the Service Manager console, select *Registry*, then *Adapters*.
2. Click *Add*.

The iBSP URL pane opens, as shown in the following image.

Provide Repository Url for the new Adapter	
iBSP URL *	Repository of available adapters with user defined targets
<input type="text" value="http://localhost:9000"/>	
<< Back	Next >>

3. Enter your iBSP URL, which is the location of the Service Manager repository, for example, <http://localhost:9000>. This field is required.
4. Click *Next*.

An adapter selection pane opens, as shown in the following image.



Adapter *

Adapters with targets defined at http://localhost:9000. If you don't see an adapter, it's probably because you do not have the adapter's JAR on the classpath.

Select an adapter ▼

<< Back Next >>

5. From the Adapter drop-down list, select the Adapter, then click *Next*.
6. From the Target drop-down list, select a target you configured for the adapter in iWay Explorer, then click *Next*.

The connection information associated with the target selected is displayed.




Set properties of the new Adapter	
Adapter	DOTNET
Target	DotNet
Create Error Document	If on, an error document will be returned when an error occurs <input type="checkbox"/> On
Persist Connection	If on, adapter connection will be reused between executes <input type="checkbox"/> On
Adapter Connection Properties	
Assemblies' Directory *	C:\Program Files\iway7\DotNetSamples
Search Recursively	true Pick one ▼

- a. Select whether to return an error document when an error occurs.
 - b. Select whether an adapter connection will be reused between executes.
For more information on the Persist Connection parameter, see [Using the Persist Connection Parameter](#) on page 66.
 - c. Review the connection information you specified in iWay Explorer. You can change or update any information.
7. Click *Next*.
 8. Provide a name and, optionally, a description, for the adapter, and click *Finish*.

The adapter appears in the adapters list, as shown in the following image.

Adapters

☐ Filter




<input type="checkbox"/>	Name	Target	References	Description
<input type="checkbox"/>	Baan2_iSM	Baan		Second try
<input type="checkbox"/>	DotNet_iSM	DOTNET		.Net adapter
<input type="checkbox"/>	Lawson_iSM	Lawson		none

Procedure: How to Modify or Update an Adapter Connection

The following image shows the Adapter Defines pane which displays the name of the adapter and the description (optional).

Adapters

☐ Filter

<input type="checkbox"/>	Name	Target	References	Description
<input type="checkbox"/>	Baan2_iSM	Baan		Second try
<input type="checkbox"/>	DotNet_iSM	DOTNET		.Net adapter
<input type="checkbox"/>	Lawson_iSM	Lawson		none

To modify or update an adapter connection:

1. From the Adapters list, click the adapter reference you defined, in this example, [DotNet_iSM](#).

The pane that displays the target connection information opens. You cannot change the name of the adapter or the target, but you can edit the connection information.

2. After you modify the connection information, click *Update Connection Properties*.
3. After you make changes or additions to the adapter target in iWay Explorer, click *Update Adapter Data*.
4. Click *Finish*.

Samples and Reference Guide

This section provides samples and reference information for the iWay .NET Technology Adapter.

In this appendix:

- ☐ [Using COM Interop in the iWay .NET Technology Adapter](#)
 - ☐ [Generating XML Schema in Legacy Mode](#)
 - ☐ [Generating XML Schema in the Current Adapter Version](#)
 - ☐ [Modifying XML Schemas](#)
 - ☐ [Additional Sample XML Documents](#)
-

Using COM Interop in the iWay .NET Technology Adapter

The code that operates within the Microsoft .NET Common Language Runtime (CLR) is called *managed code*. Managed code has access to all of the services provided by the CLR, such as security, versioning support, garbage collection, and cross-language integration. The iWay .NET Technology Adapter is implemented by managed codes. Code that does not operate within the CLR is called *unmanaged code*. Managed components depend on the CLR and expect other components with which they interact, to depend on the CLR as well.

In order for the iWay .NET Technology Adapter to use a COM object, wrapper objects, called Runtime-Callable Wrappers (RCW), must be generated. The RCW objects cater for the difference in lifetime management between .NET and COM. RCW objects are .NET objects that manage the reference count of a COM object, as well as deal with the organization of parameters and return types for the COM object methods.

RCW objects are manufactured at runtime by the CLR using information found in an Interop Assembly, which is an assembly containing definitions of COM types that can be used from managed code.

The simplest way to generate an Interop Assembly is to use the Type Library Importer tool (TlbImp.exe), which is a command-line tool provided by the Microsoft .NET Framework SDK. The Type Library Importer tool converts a COM type library into .NET Framework metadata, effectively creating a managed wrapper that can be called from any managed language.

You can select many options using the Type Library Importer tool. The most important option is `/out`, which allows you to specify a name for the resulting .NET assembly. The following command line converts a COM component called `comp.dll` to a matching .NET assembly called `interop.NETcomp.dll`:

```
tlbimp comp.dll /out: interop.NETcomp.dll
```

You can also use the Microsoft Intermediate Language (MSIL) Disassembler tool (`Ildasm.exe`) to examine the resulting DLL. The MSIL Disassembler tool is also a command-line tool provided by the Microsoft .NET Framework SDK. This tool parses any .NET Framework .exe or .dll assembly, and shows the information in a legible format. The MSIL Disassembler tool shows more than just the MSIL code, it also displays namespaces and types, including their interfaces.

For example, to display the contents of the `interop.NETcomp.dll` file, use the following command:

```
Ildasm interop.NETcomp.dll
```

After creating an Interop Assembly for an unmanaged COM component, you can run the assembly just like running other C# components in the iWay .NET Technology Adapter.

Generating XML Schema in Legacy Mode

In the legacy version of the iWay .NET Technology Adapter, the following primary functions are used to handle XML schema generation for a method:

☐ `ComposeDocMethod`

☐ `ComposeRpcMethod`

Note: In legacy mode, there is limited support for defined complex types. It is recommended to use the new dynamic schema methods for methods with complex types.

The earlier version of the iWay .NET Technology Adapter requires a custom .NET wrapper, or at least recompilation, to consume assemblies. The custom .NET wrapper references `iwclr.dll` and wraps a target .NET assembly to provide schemas or descriptions for methods by adding special attributes, which will be retrieved by `iwclr.dll`. The following primary attributes are provided by a wrapper for a method:

☐ `XmlInXmlOutAttribute`

```
[XmlInXmlOutAttribute(METHODDESCRIPTION, ROOTELEMENT, RAWINPUTSCHEMA,  
OUTPUTDATANAME, RAWOUTPUTSCHEMA)]  
public XmlElement Add (XmlElement input)  
{  
    //codes  
}
```

❑ ParamsInParamsOutAttribute

```
[ParamsInParamsOutAttribute(METHODDESCRIPTION)] public double Sqrt
(double number)
{ //code }
```

The following table lists and describes the key words that are allowed.

Key Word	Type	Description
<code>METHODDESCRIPTION</code>	String	The description of a method.
<code>SCHEMAROOTELEMENT</code>	String	The name of a root element (normally, It is the name of a method).
<code>RAWINPUTSCHEMA</code>	String	Raw request schema (a schema is retrieved from a attribute of a wrapper).
<code>OUTPUTDATANAME</code>	String	The name of an output data.
<code>RAWOUTPUTSCHEMA</code>	String	Raw response schema (a schema is retrieved from a attribute of a wrapper).

ComposeDocMethod Function

If a custom .NET wrapper for a method has XmlInXmlOutAttribute attributes, then the ComposeDocMethod function is called. The function ComposeDocMethod only supports the method that accepts one XmlElement type input parameter and returns XmlElement type data.

ComposeRpcMethod Function

If a custom wrapper has ParamsInParamsOutAttribute attributes, then the function ComposeRpcMethod is called. The function ComposeRpcMethod can only support the method that accepts some simple .NET type input parameter(s) (for example, int, String, and so on), and returns some simple .NET type data.

Generating XML Schema in the Current Adapter Version

The following table lists and describes the key words that are allowed.

Key Word	Type	Description
[AssemblyName]	String	The name of an assembly. For example, Math.dll.
[Namespace]	String	The namespace which is used in an assembly.
[CustomTypeName]	String	The name of a complex custom type.
[ClassName]	String	The name of a class.
[FRAMEWORKTYPE]	String	<p>The name of a .NET Framework type.</p> <p>To convert a C# built-in type to a .NET Framework type, a Built-In Types Table is available on the following webpage: http://msdn.microsoft.com/en-us/library/ya5y69ds.aspx.</p> <p>The .Net Framework type of a complex custom type has the following format:</p> <p>[Namespace].[CustomTypeName]</p>
[Method signature]	String	<p>The part of the method declaration. It is the combination of the method name and the parameter type list.</p> <p>In a parameter type list, the name of a .NET Framework type to present a C# type is used.</p> <p>[Method signature] sample: A complex custom type named CUSTOM is defined in a namespace NS.</p> <p>C# function:</p> <pre>void setValue(CUSTOM obj, int input)</pre> <p>[Method signature]:</p> <pre>setValue(NS.CUSTOM, System.Int32)</pre>

Key Word	Type	Description
<code>[iWayURI]</code>	String	<p>In a request schema:</p> <code>[iWayURI]="urn:iwaysoftware: ibse:jul2003:" + METHODNAME.</code> <code>[iWayURI]="urn:iwaysoftware: ibse:jul2003:" + METHODNAME</code> <p>In a response schema:</p> <code>[iWayURI]="urn:iwaysoftware: ibse:jul2003:" + METHODRESPONSEE</code>
<code>INOUTDATANAME</code>	String	<p>For a request schema, it is the name of an input parameter.</p> <p>For a response schema, it is equal to result.</p>
<code>NAMESPACECOMPLEXTYPE</code>	String	<p>The name of a custom type with the following namespace:</p> <code>[Namespace].[CustomTypeName]</code>
<code>TNSNAMESPACECOMPLEXTYPE</code>	String	<p><code>NAMESPACECOMPLEXTYPE</code> with namespace prefixes <i>tns</i>.</p> <p>Format:</p> <code>tns:[Namespace].[CustomTypeName]</code> <p>For example: <code>tns:Math.ComplexNumber</code></p>
<code>XSTYPE</code>	String	<p>The XML built-in data type with namespace prefix <i>xs</i>.</p> <p>For example: <code>xs:int</code>, <code>xs:string</code> and so on.</p>
<code>SUBELEMENTNAME</code>	String	The name of a sub-element.
<code>SUBCOMPLEXNAME</code>	String	The name of a sub-element, which has a complex custom type.
<code>METHODNAME</code>	String	The name of a method.
<code>METHODRESPONSE</code>	String	It is equal to <code>METHODNAME+"Response"</code> .

Key Word	Type	Description
<code>METHODLOGICPATH</code>	String	<p>The logic path of a method.</p> <p>Format: <code>/[AssemblyName]/[Namespace].[ClassName]/[Method signature]</code></p> <p>For example: <code>/Server.dll/Server.TestServer/GetVehicle(System.Int32)</code></p>
<code>PARAMETERNAME</code>	String	The name of an input or output parameter.

In the current adapter version, XML request and response schemas can be generated.

An element is used to represent a method, and sub-elements under the sequence node of the element are used to represent input parameters in a request schema. Similar to earlier versions of the adapter, an element is still used to describe return data and type in a response schema.

Whether you are using the previous version of the adapter or the current version, a .NET built-in type data is described by one element node that contains the name and type attributes. The value of the name attribute is the name of the data, and the value of the type attribute is the name of XML built-in type with namespace prefix xs. Unlike the early version of the adapter, the current version of the adapter is also able to create schema for complex custom data types.

Complex Custom Type Representation Rules

To represent a complex custom data type, two rules are defined.

Rule A

For a complex custom type input or output, a type attribute to the element is applied and referred to the name of the complex custom type to be used, as shown in the following image.

```
<xs:element name=INOUTDATANAME type= TNSNAMESPACECOMPLEXTYPE />
<xs:complexType name= NAMESPACECOMPLEXTYPE >
  <xs:sequence>
    <!-- This sequence node may contain multi sub-elements -->
    <!-- A sub-element may have a .Net built-in type or complex type -->
    <xs:element name= SUBELEMENTNAME type= XSTYPE />
    <xs:element name= SUBCOMPLEXNAME >
      <!-- The sub-element has a complex custom type. We can follow the-->
      <!-- Rule_B to describe the complex type of the element, and so on.-->
      ...
    </xs:sequence>
  </xs:complexType>
```

Rule B

A complex custom type data may contain complex custom type sub-data. An element can be directly declared by naming the element for the complex custom type sub-data, as shown in the following image.

```
<xs:element name = SUBCOMPLEXNAME >
  < xs:complexType>
    < xs:sequence >
      <!-- This sequence node may contain multi sub-elements -->
      <!-- A sub-element may have a .Net built-in type or complex type -->
      <xs:element name= SUBELEMENTNAME type= XSTYPE />
      <xs:element name= SUBCOMPLEXNAME >
        <!-- The sub-element has a complex custom type. We can follow the-->
        <!-- Rule_B to describe the complex type of the element, and so on.-->
        ...
      </ xs:sequence >
    </xs:complexType >
  </ xs:element>
```

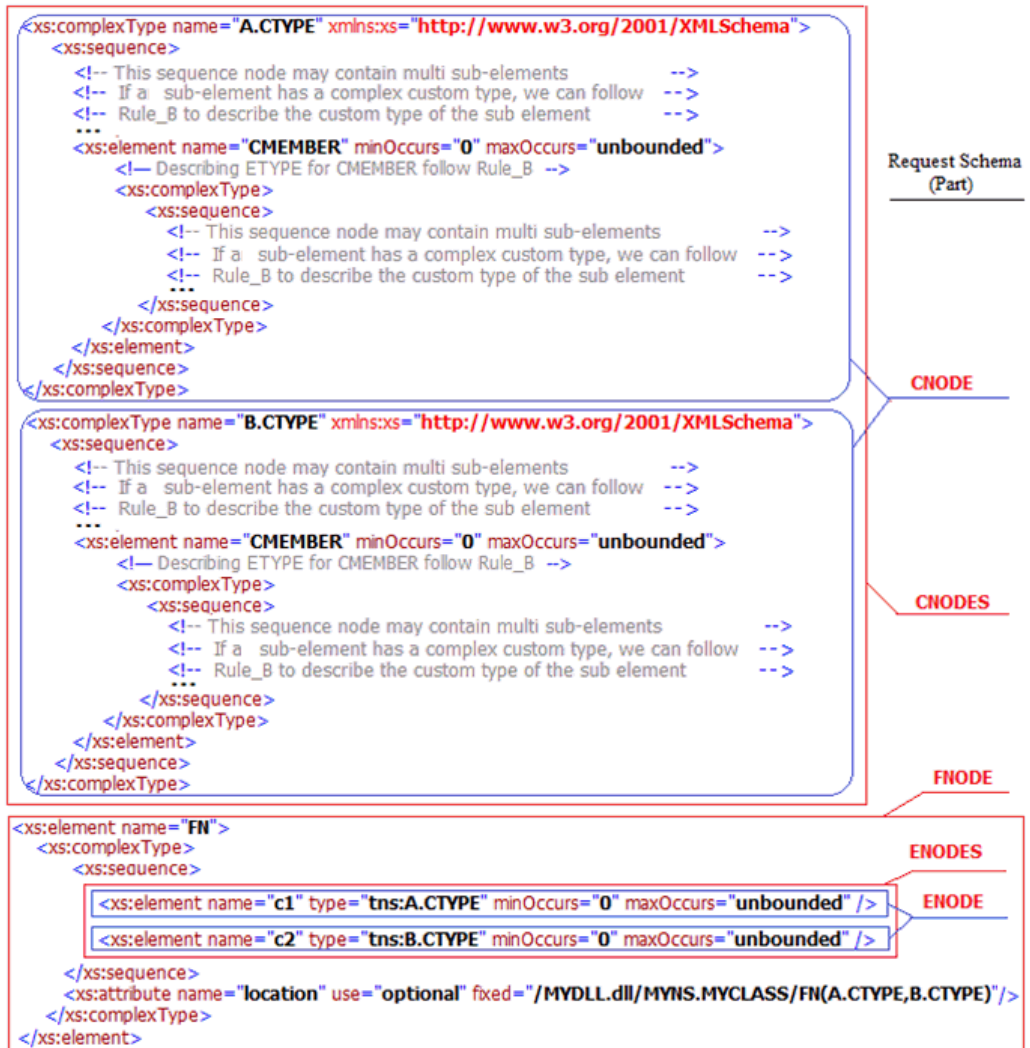
Namespace and Target Namespace

The *xmlns:xs* attribute is declared with the value *http://www.w3.org/2001/XMLSchema* in all XML request and response schemas. Each of the constructs that are defined by the XML schema language will need to be qualified with the *xs* prefix.

There are no namespace issues if a method only has .NET built-in type inputs or outputs. Namespace issues are only considered if a method has complex custom type inputs or outputs.

Consider the following example. MYDLL.dll is a .NET assembly that contains a class called MYCLASS that is defined in the MYNS namespace. MYCLASS has a function called FN with prototype A.CTYPE FN (A.CTYPE c1, B.CTYPE c2). In the function, A and B are different namespaces and both define a complex custom type named CTYPE. Even though CTYPE in A contains a member variable called CMEMBER that has a complex custom type ETYPE, and CTYPE in B also contains a member variable called CMEMBER that has a complex custom type ETYPE, CTYPE in A and CTYPE in B are different. This causes the name conflict in the request and response schemas if the data and types are not properly described.

The following image shows the structure of a XML request schema.



The following image shows the structure of an XML response schema.



In order to solve namespace problems in the XML request and response schemas for a method that supports complex custom type inputs or outputs, full names are used to represent custom types for input parameters or output data. The full name of a custom type has the following format:

`[Namespace].[CustomTypeName]`

The `targetNamespace` and `xmlns:tns` attributes in the request and response schemas are declared. All custom type(s) of input parameter(s) or returning data belong to the namespace [iwayURI] as defined by the `targetNamespace` attribute, and the prefix is *tns* as defined by the `xmlns:tns` attribute.

```
targetNamespace=[iwayURI] (reference key table)
xmlns:tns      =[iwayURI]
xmlns:xs       ="http://www.w3.org/2001/XMLSchema"
```

Following the previously mentioned solutions and the Complex Custom Type Representation Rules, you can create elements to represent input or output data which have complex types, and describe the complex types by generating `xs:complexType` nodes.

As shown in the images that depict the structure of XML request and response schemas, the XML data are the parts of the request and response schemas of the MyDll.dll sample. The current release of the adapter generated ENODEs based on Rule_A and generated CNODEs based on Rule_B. An ENODE represents a parameter or return data and a CNODE describes the custom type of a parameter or return data. The sub data CMEMBER is also described by an `xs:complexType` node based on Rule_B.

Moreover, if a sub data has custom type members which can be made public, you can also follow Rule_B to describe them, and so on. How deep the complex type sub data can be extended depends on how the object of DataSet class infers schema from the data and loads the data.

The request and response schemas generated by this technique can handle multiple types with the same name from different DLLs or objects and support objects referencing variables of complex types with same names in the request and response schemas.

Formats of XML Request and Response Schemas

Formatting of XML request schemas can be based on the following scenarios:

- ☐ **Situation A.** A method without input parameters.
- ☐ **Situation B.** A method with simple .NET built-in type input parameters.
- ☐ **Situation C.** A method with complex custom or mix (simple and complex) type input parameters.

Formatting of XML response schemas can be based on the following scenarios:

- ☐ **Situation A1.** A method does not return a result.
- ☐ **Situation B1.** A method returns a simple .NET built-in type data.

- ❑ **Situation C1.** A method returns a complex custom type.
- ❑ **Situation D1.** A method returns a data (simple type or complex custom type) and output simple type or complex custom type parameter(s).

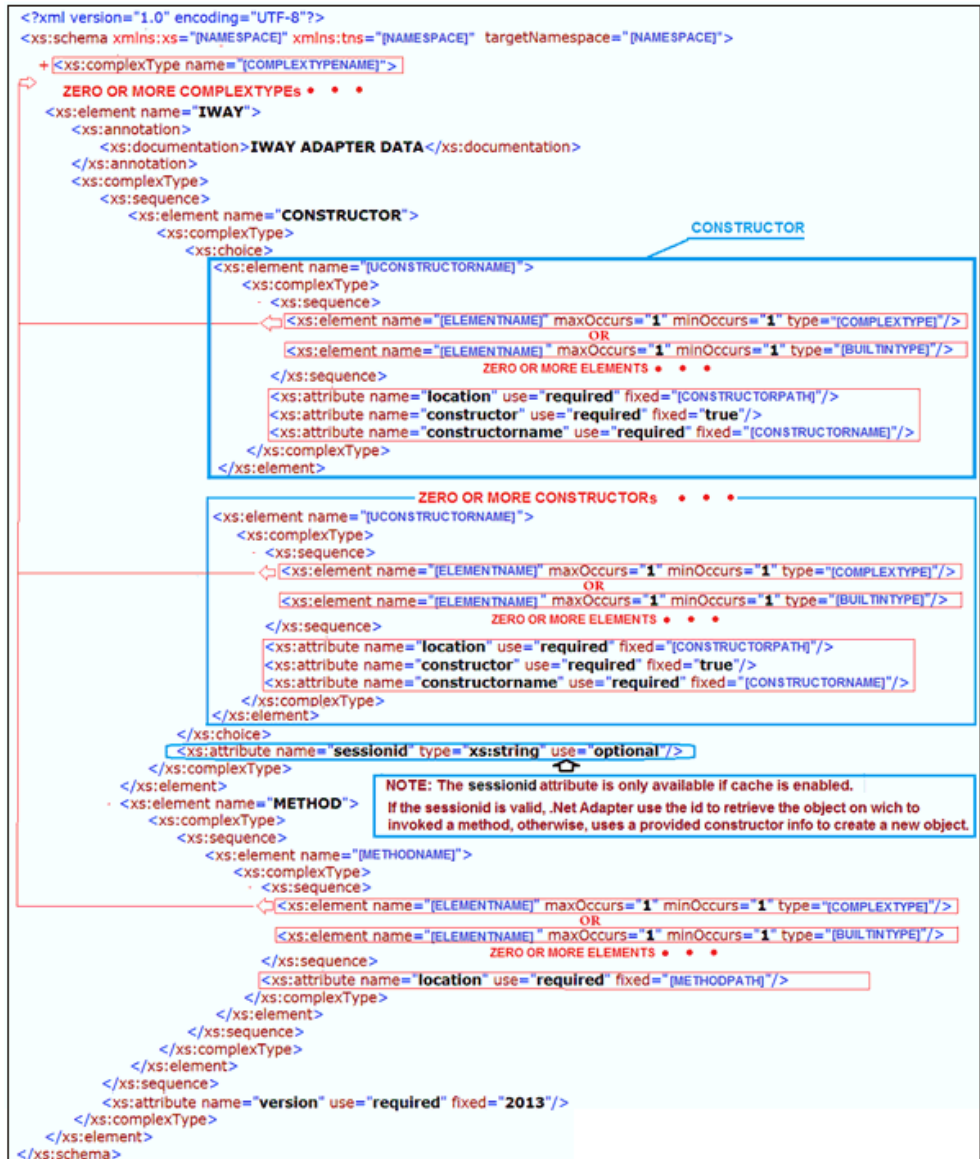
The legacy version of the iWay .NET Technology Adapter only supports functions with some simple type inputs and outputs. The format of an XML request schema for the current adapter version is similar to the early version under Situation A and Situation B. The format of an XML response schema for the current adapter version has more information than the earlier version under Situation A1 and Situation B1. Only the current version of the adapter can support Situation C, Situation C1, and Situation D1.

Modifying XML Schemas

Every method and constructor has specific XML request and response schemas. A constructor item will be displayed under an Assembly node within the hierarchy of iWay Explorer only if users enable the cache for that Assembly. In order to allow users to reuse the object on which to invoke a method, a new string data type called *sessionid* is defined in the schemas that are generated in iWay Explorer. This identifies a unique object. When users want to execute a method of an instance of a class, they can either pass the information of a constructor through a schema to create a new instance or pass a *sessionid* through schema to reuse a previously created instance for the class. For a cacheable Assembly, a *sessionid* is returned in a response from a constructor request or method request.

Method Schemas

The following image is an example of a request schema for a method.



The following image is an example of a response schema for a method.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="[NAMESPACE]" xmlns:tns="[NAMESPACE]" targetNamespace="[NAMESPACE]">
  <xs:element name="[METHODRESPONSE]">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="methodReturn">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="result" maxOccurs="unbounded" minOccurs="0" type="[COMPLEXTYPE]"/>
              OR
              <xs:element name="result" maxOccurs="unbounded" minOccurs="0" type="[BUILTINTYPE]"/>
            </xs:sequence>
            <xs:attribute name="status" type="xs:string" use="optional"/>
            <xs:attribute name="description" type="xs:string" use="optional"/>
            <xs:attribute name="encode" type="xs:string" use="optional"/>
            <xs:attribute name="sessionId" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="[COMPLEXTYPENAME]">
    ZERO OR MORE COMPLEX TYPEs . . .
  </xs:complexType>
</xs:schema>
```

Schema Properties Reference

In the XML request and response schemas for constructors and methods provided in this example:

[NAMESPACE]

Is the name of a namespace.

[COMPLEXTYPENAME]

Is the name of a complex type.

[CONSTRUCTORNAME]

Is the name of a constructor of a class.

[ELEMENTNAME]

Is the name of an argument of a constructor or a name of a parameter of a method.

[COMPLEXTYPE]

Is a complex type, which is equal to:

"tns:" + [COMPLEXTYPENAME]

[BUILTINTYPENAME]

Is the name of a built-in data type.

[BUILTINTYPE]

Is a built-in data type, which is equal to:

"xs:" + [BUILTINTYPENAME]

[ASSEMBLYPATH]

Is the location of an Assembly.

[METHODSIGNATURE]

Is the signature of a method. This is a combination of the name of the method and the number and types of parameters (and their order).

[CONSTRUCTORSIGNATURE]

Is the signature of a constructor. This is a combination of the name of the constructor and the number and types of arguments (and their order).

[FULLCLASSNAME]

Is the combination of the namespace of the class along with the period character (.) and the name of the class.

[CONSTRUCTORPATH]

Is the following:

[ASSEMBLYPATH] / [FULLCLASSNAME] / [CONSTRUCTORSIGNATURE]

[METHODPATH]

Is the following:

[ASSEMBLYPATH] / [FULLCLASSNAME] / [METHODSIGNATURE]

[CONSTRUCTORRESPONSE]

Is the combination of a name of a constructor along with the string *Response*.

[METHODRESPONSE]

Is the combination of a name of a method along with the string *Response*.

[UCONSTRUCTORNAME]

Is a unique name of an identified onstructor of a class.

If a class only contains one constructor, then:

[UCONSTRUCTORNAME] = [CONSTRUCTORNAME]

If a class contains multiple constructors, then:

[UCONSTRUCTORNAME]=[CONSTRUCTORNAME]+"_"+string of a positive number

[METHODNAME]

Is the name of a method.

The following fragment represents a node of a complex type:

"<xs:complexType name="[COMPLEXTYPENAME]">

Expanding Nodes Based on Different Complex Types

You can expand nodes based on different complex types of nodes, as shown in the schema fragments in this section.

System.Collections.ArrayList Type

```
<xs:complexType name="System.Collections.ArrayList">
  <xs:sequence>
    <xs:element name="ArrayOfAnyType" minOccurs="1" maxOccurs="1">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="anyType" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="object" minOccurs="1" maxOccurs="1" type="xs:anyType"/>
              </xs:sequence>
              <xs:attribute name="type" type="xs:string" use="required"/>
              <xs:attribute name="typeIsEqualTo" use="required" fixed="The assembly-qualified name of the object's type"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="type" use="required" fixed="System.Collections.ArrayList"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

System.Collections.Generic.Dictionary Type

```
<xs:complexType name="[DICTIONARYNAME]">
  <xs:sequence>
    <xs:element name="[DICTIONARYKEYVALUENAME]" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Key" type="[COMPLEXTYPE]"/>
          OR
          <xs:element name="Key" type="[BUILTINTYPE]"/>
          <xs:element name="Value" type="[COMPLEXTYPE]"/>
          OR
          <xs:element name="Value" type="[BUILTINTYPE]"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

[DICTIONARYNAME]

Is the name of dictionary, which is the combination of the string *ArrayOfKeyValueOf* along with [DICTIONARYKEYVALUENAME].

[DICTIONARYKEYVALUENAME]

Is the data, which is the combination of the string of key type along with the string of value type.

Array or System.Collections.Generic.List Type

```
<xs:complexType name="[ARRAYNAME]">
  <xs:sequence>
    <xs:element name="[ELEMENTNAME]" minOccurs="1" maxOccurs="unbounded" type="[COMPLEXTYPE]"/>
    OR
    <xs:element name="[ELEMENTNAME]" minOccurs="1" maxOccurs="unbounded" type="[BUILTINTYPE]"/>
  </xs:sequence>
</xs:complexType>
```

[ARRAYNAME]

Is the name of an array, which is the combination of the string *ArrayOf* along with the string of a data type.

Pointer Type

```
<xs:complexType name="[POINTERNAME]">
  <xs:sequence>
    <xs:element name="[ELEMENTNAME]" minOccurs="1" maxOccurs="1" type="[COMPLEXTYPE]"/>
    OR
    <xs:element name="[ELEMENTNAME]" minOccurs="1" maxOccurs="1" type="[BUILTINTYPE]"/>
  </xs:sequence>
</xs:complexType>
```

Constructor Type

```
<xs:complexType name="[FULLCLASSNAME]">
  <xs:choice>
    <xs:element name="[UCONSTRUCTORNAME]">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="[ELEMENTNAME]" maxOccurs="1" minOccurs="1" type="[COMPLEXTYPE]"/>
          OR
          <xs:element name="[ELEMENTNAME]" maxOccurs="1" minOccurs="1" type="[BUILTINTYPE]"/>
        </xs:sequence>
        <xs:attribute name="location" use="required" fixed="[CONSTRUCTORPATH]"/>
        <xs:attribute name="constructor" use="required" fixed="true"/>
        <xs:attribute name="constructorname" use="required" fixed="[CONSTRUCTORNAME]"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="[UCONSTRUCTORNAME]">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="[ELEMENTNAME]" maxOccurs="1" minOccurs="1" type="[COMPLEXTYPE]"/>
          OR
          <xs:element name="[ELEMENTNAME]" maxOccurs="1" minOccurs="1" type="[BUILTINTYPE]"/>
        </xs:sequence>
        <xs:attribute name="location" use="required" fixed="[CONSTRUCTORPATH]"/>
        <xs:attribute name="constructor" use="required" fixed="true"/>
        <xs:attribute name="constructorname" use="required" fixed="[CONSTRUCTORNAME]"/>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

The full class name is the combination of the namespace of the class along with the character '.' and the name of the class.

ZERO OR MORE ELEMENTS • • •

ZERO OR MORE CONSTRUCTORS • • •

Complete XML Request Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:iwaysoftware:ibse:jul2003:Cosine"
targetNamespace="urn:iwaysoftware:ibse:jul2003:Cosine">
  - <xs:complexType name="ArrayOfComplex.ComplexNumber">
    - <xs:sequence>
      <xs:element name="Complex.ComplexNumber" minOccurs="1" maxOccurs="unbounded"
type="tns:Complex.ComplexNumber"/>
    </xs:sequence>
  </xs:complexType>
  - <xs:complexType name="Complex.ComplexNumber">
    - <xs:choice>
      - <xs:element name="ComplexNumber">
        - <xs:complexType>
          - <xs:sequence>
            <xs:element name="inputobj" minOccurs="1" maxOccurs="1" type="xs:int"/>
            </xs:sequence>
            <xs:attribute name="constructorname" fixed="ComplexNumber" use="required"/>
            <xs:attribute name="constructor" fixed="true" use="required"/>
            <xs:attribute name="location"
fixed="C:/Sample/Debug/Complex.dll/Complex.ComplexNumber/ComplexNumber(System.Int32)"
use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element name="IWAY">
    - <xs:annotation>
      <xs:documentation>IWAY ADAPTER DATA</xs:documentation>
    </xs:annotation>
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="CONSTRUCTOR">
          - <xs:complexType>
            - <xs:choice>
              - <xs:element name="Math">
                - <xs:complexType>
                  - <xs:sequence>
                    <xs:element name="mycomplexs" minOccurs="1" maxOccurs="1"
type="tns:ArrayOfComplex.ComplexNumber"/>
                    </xs:sequence>
                    <xs:attribute name="constructorname" fixed="Math" use="required"/>
                    <xs:attribute name="constructor" fixed="true" use="required"/>
                    <xs:attribute name="location" fixed="C:/Sample/Debug/Math.dll/Math.Math/Math
(Complex.ComplexNumber[]&)" use="required"/>
                  </xs:complexType>
                </xs:element>
              - <xs:element name="Math_1">
                - <xs:complexType>
                  - <xs:sequence>
                    <xs:element name="mystr" minOccurs="1" maxOccurs="1" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="constructorname" fixed="Math" use="required"/>
                    <xs:attribute name="constructor" fixed="true" use="required"/>
                    <xs:attribute name="location" fixed="C:/Sample/Debug/Math.dll/Math.Math/Math
(System.String)" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
            <xs:attribute name="sessionid" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      - <xs:element name="METHOD">
        - <xs:complexType>
          - <xs:sequence>
            - <xs:element name="Cosine">
              - <xs:complexType>
                - <xs:sequence>
                  <xs:element name="angle" minOccurs="1" maxOccurs="1" type="xs:double"/>
                  </xs:sequence>
                  <xs:attribute name="location" fixed="/Debug/Math.dll/Math.Math/Cosine
(System.Double)" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

Additional Sample XML Documents

This section provides additional sample XML documents for the iWay .NET Technology Adapter.

Constructor Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
- <tns:Math constructorname="Math" constructor="true" location="/Debug/Math.dll/Math.Math/Math
  (Complex.ComplexNumber&)" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:Math
  C:\XMLSample\WorkSample\DOTNETMATH\Math\bin\4_12_usecache\Math_1_request.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:Math">
  - <mycomplex>
    - <ComplexNumber_8 constructorname="ComplexNumber" constructor="true"
      location="C:/Example/bin/Complex.dll/Complex.ComplexNumber/ComplexNumber
      (System.Int32,System.Int32&)">
      <a>9738</a>
      <inoutval>44</inoutval>
    </ComplexNumber_8>
  </mycomplex>
</tns:Math>
```

Array Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
- <tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:gettwomergeComplexandComplexArray
  C:/Example/bin\4_12_usecache\gettwomergeComplexandComplexArray_request.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tns="urn:iwaysoftware:ibse:jul2003:gettwomergeComplexandComplexArray">
  - <CONSTRUCTOR sessionid="string">
    - <Math_3 location="C:/Example/bin/Math.dll/Math.Math/Math(System.Int32,System.String&)" constructor="true"
      constructorname="Math">
      <a>4144</a>
      <refstring>string</refstring>
    </Math_3>
  </CONSTRUCTOR>
  - <METHOD>
    - <gettwomergeComplexandComplexArray
      location="/Debug/Math.dll/Math.Math/gettwomergeComplexandComplexArray
      (Complex.ComplexNumber,Complex.ComplexNumber[])">
      - <c1>
        - <ComplexNumber_6 location="C:/Example/bin/Complex.dll/Complex.ComplexNumber/ComplexNumber
          (System.Int32,System.Int32)" constructor="true" constructorname="ComplexNumber">
          <r>-5187</r>
          <i>-1361</i>
        </ComplexNumber_6>
      </c1>
      - <c2>
        - <Complex.ComplexNumber>
          - <ComplexNumber_6 location="C:/Example/bin/Complex.dll/Complex.ComplexNumber/ComplexNumber
            (System.Int32,System.Int32)" constructor="true" constructorname="ComplexNumber">
            <r>8936</r>
            <i>2484</i>
          </ComplexNumber_6>
        </Complex.ComplexNumber>
      </c2>
    </gettwomergeComplexandComplexArray>
  </METHOD>
</tns:IWAY>
```

Pointer Example

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
- <tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:DisplayStruct
C:\XMLSample\WorkSample\DOTNETMATH\Math\bin\4_12_usecache\DisplayStruct_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:DisplayStruct">
  - <CONSTRUCTOR sessionid="string">
    - <Math_5 location="C:/Example/bin/Math.dll/Math.Math/Math(Complex.ComplexNumber,System.Int32)"
      constructor="true" constructortname="Math">
      - <mycomplex>
        - <ComplexNumber_6 location="C:/Example/bin/Complex.dll/Complex.ComplexNumber/ComplexNumber
          (System.Int32,System.Int32)" constructor="true" constructortname="ComplexNumber">
          <r>-7506</r>
          <i>-8024</i>
        </ComplexNumber_6>
      </mycomplex>
      <size>4495</size>
    </Math_5>
  </CONSTRUCTOR>
  - <METHOD>
    - <DisplayStruct location="/Math.dll/Math.Math/DisplayStruct(Math.structObj*)">
      - <obj>
        - <Math.structObj>
          - <structObj location="C:/Example/bin/Math.dll/Math.structObj/structObj(System.Int32,System.Char)"
            constructor="true" constructortname="structObj">
            <inputint>78</inputint>
            <inputchar>85</inputchar>
          </structObj>
        </Math.structObj>
      </obj>
    </DisplayStruct>
  </METHOD>
</tns:IWAY>

```

Dictionary Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:getOutputBaseonDictionary C:\bin\getOutputBaseonDictionary_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:getOutputBaseonDictionary">
  - <CONSTRUCTOR sessionid="string">
    - <Math_6 location="C:/Example/bin/Math.dll/Math.Math/Math(System.Int32,System.String,System.Int32)" constructor="true" constructorname="Math">
      <myval>17</myval>
      <mystr>hello world</mystr>
      <size>32</size>
    </Math_6>
  </CONSTRUCTOR>
  - <METHOD>
    - <getOutputBaseonDictionary location="/Math.dll/Math.Math/getOutputBaseonDictionary(System.Collections.Generic.Dictionary2[System.String,
mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089],[Complex.ComplexNumber, Complex, Version=65534.36290.0.0,
Culture=neutral, PublicKeyToken=null]])">
      - <mydictionary>
        - <StringComplex.ComplexNumber>
          <Key>string-0</Key>
          <Value>
            - <ComplexNumber_11 location="C:/XMLSample/WorkSample/DOTNETMATH/Math/bin/Debug/Complex.dll/Complex.ComplexNumber/
ComplexNumber(System.Int32,System.Int32,System.String,System.Int32[])" constructor="true" constructorname="ComplexNumber">
              <r>7535</r>
              <i>20</i>
            </ComplexNumber_11>
            <desc>test</desc>
          </Value>
          - <inputarray>
            <Int>17</Int>
            <Int>13</Int>
          </inputarray>
        </StringComplex.ComplexNumber>
        - <StringComplex.ComplexNumber>
          <Key>string-2</Key>
          <Value>
            - <ComplexNumber_4 location="C:/XMLSample/WorkSample/DOTNETMATH/Math/bin/Debug/Complex.dll/Complex.ComplexNumber/
ComplexNumber(System.Int32[])" constructor="true" constructorname="ComplexNumber">
              - <inputobj>
                <Int>6836</Int>
                <Int>29</Int>
              </inputobj>
            </ComplexNumber_4>
          </Value>
        </StringComplex.ComplexNumber>
      </mydictionary>
    </getOutputBaseonDictionary>
  </METHOD>
</tns:IWAY>
```

ArrayList Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:InOutArrayList C:\bin\InOutArrayList_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:InOutArrayList">
  <CONSTRUCTOR sessionid="string">
    <Math_8 location="C:/Example/bin/Math.dll/Math.Math/Math(System.String)" constructor="true" constructortname="Math">
      <mystr>Test only!</mystr>
    </Math_8>
  </CONSTRUCTOR>
  <METHOD>
    <InOutArrayList location="/Math.dll/Math.Math/InOutArrayList(System.Collections.ArrayList)">
      <list>
        <ArrayOfAnyType type="System.Collections.ArrayList">
          <anyType type="System.DateTime, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
typeIsEqualTo="The assembly-qualified name of the object's type">
            <object>2013-03-28T12:08:00.6120873-04:00</object>
          </anyType>
          <anyType type="SubComplex.SubComplexData, SubComplex, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
typeIsEqualTo="The assembly-qualified name of the object's type">
            <object>
              <SubComplexData_1 location="C:/Debug/SubComplex.dll/SubComplex.SubComplexData/SubComplexData(
System.String, System.Int32)" constructor="true" constructortname="SubComplexData">
                <desc>Hello World</desc>
                <arraysize>68</arraysize>
              </SubComplexData_1>
            </object>
          </anyType>
          <anyType type="System.String[], mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
typeIsEqualTo="The assembly-qualified name of the object's type">
            <object>
              <string>one</string>
              <string>two</string>
              <string>three</string>
              <string>four</string>
            </object>
          </anyType>
          <anyType type="System.Int32, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
typeIsEqualTo="The assembly-qualified name of the object's type">
            <object>12345678</object>
          </anyType>
          <anyType type="System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
typeIsEqualTo="The assembly-qualified name of the object's type">
            <object>Hello World</object>
          </anyType>
        </ArrayOfAnyType>
      </list>
    </InOutArrayList>
  </METHOD>
</tns:IWAY>
```

XMLElement Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:addTwoIntegerXmlElements C:\bin\addTwoIntegerXmlElements_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:addTwoIntegerXmlElements">
  <CONSTRUCTOR sessionid="string">
    <Math_6 location="C:/Example/bin/Math.dll/Math.Math/Math(System.Int32,System.String,System.Int32)" constructor="true" constructortname="Math">
      <myval>8931</myval>
      <mystr>Test</mystr>
      <size>16</size>
    </Math_6>
  </CONSTRUCTOR>
  <METHOD>
    <addTwoIntegerXmlElements location="/Math.dll/Math.Math/addTwoIntegerXmlElements(System.Xml.XmlElement,System.Xml.XmlElement)">
      <input>
        <or:XXX xmlns:or="AnyNamespace">19</or:XXX>
        <or:XXX xmlns:or="AnyNamespace">12</or:XXX>
      </input>
      <input1>
        <or:XXX xmlns:or="AnyNamespace">112</or:XXX>
        <or:XXX xmlns:or="AnyNamespace">120</or:XXX>
      </input1>
    </addTwoIntegerXmlElements>
  </METHOD>
</tns:IWAY>
```


List Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:setAndGetBases C:\bin\setAndGetBases_request.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:setAndGetBases">
  - <CONSTRUCTOR sessionid="string">
    - <Math_1 location="C:/Example/bin/Math.dll/Math.Math/Math(Complex.ComplexNumber&)" constructor="true" constructorname="Math">
      - <mycomplex>
        - <ComplexNumber_4 location="C:/Example/bin/Complex.dll/Complex.ComplexNumber/ComplexNumber(System.Int32[])"
          constructor="true" constructorname="ComplexNumber">
            - <inputobj>
              <Int>4531</Int>
              <Int>41</Int>
              <Int>51</Int>
            </inputobj>
          </ComplexNumber_4>
        </mycomplex>
      </Math_1>
    </CONSTRUCTOR>
  - <METHOD>
    - <setAndGetBases location="/Math.dll/Math.Math/setAndGetBases(System.Collections.Generic.List1[[Math.DerivedBase, Math,
      Version=65534.36290.0.0, Culture=neutral, PublicKeyToken=null]],System.Int32)">
      - <derivedobjlist>
        - <DerivedBase>
          - <DerivedBase_1 location="C:/Example/bin/Math.dll/Math.DerivedBase/DerivedBase(System.Int32,System.Int32)"
            constructor="true" constructorname="DerivedBase">
            <a>3149</a>
            <b>-7535</b>
          </DerivedBase_1>
        </DerivedBase>
        - <DerivedBase>
          - <DerivedBase_1 location="C:/XMLSample/WorkSample/DOTNETMATH/Math/bin/Debug/Math.dll/Math.DerivedBase/
            DerivedBase(System.Int32,System.Int32)" constructor="true" constructorname="DerivedBase">
            <a>69</a>
            <b>82</b>
          </DerivedBase_1>
        </DerivedBase>
      </derivedobjlist>
      <plusvalue>-3348</plusvalue>
    </setAndGetBases>
  </METHOD>
</tns:IWAY>
```

Getting SESSIONID Example

Users can get a SESSIONID by the Constructor Response, as shown in the following image.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <Math_1Response xmlns="urn:iwaysoftware:ibse:jul2003:Math_1:response" cid="C06BE5DB08D5C6E18773EE8B4F3E8AE">
      - <MathResponse>
        - <constructorReturn description="Constructor" status="Success">
          <sessionid>5f2d457c-dce8-4b5e-aa79-bc1ff465a13f</sessionid>
        </constructorReturn>
      </MathResponse>
    </Math_1Response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

CONSTRUCTOR RESPONSE

Users can get a SESSIONID by the Method Response, as shown in the following image.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <divideByResponse xmlns="urn:iwaysoftware:ibse:jul2003:divideBy:response" cid="0D32132C60567D3FC7DC65011EDA1D7F">
      <divideByResponse>
        <methodReturn sessionId="a594b38c-35c2-4f9c-b751-e552c5a61488">
          <result>
            <int>2404</int>
          </result>
        </methodReturn>
      </divideByResponse>
    </divideByResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

METHOD RESPONSE

Using SESSIONID Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:divideBy C:\bin\divideBy_request.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:divideBy">
  <CONSTRUCTOR sessionId="5e145833-1056-462a-a3c5-17149d65563a">
    <Math_3 location="C:/Example/bin/Math.dll/Math.Math/Math(System.Int32,System.String&)" constructor="true"
      constructorname="Math">
      <a>-1087</a>
      <refstring>string</refstring>
    </Math_3>
  </CONSTRUCTOR>
  <METHOD>
    <divideBy location="/Math.dll/Math.Math/divideBy(System.Int32,System.Int32)">
      <a>4806</a>
      <b>2</b>
    </divideBy>
  </METHOD>
</tns:IWAY>
```

CONSTRUCTOR INFO

The iWay .NET Technology Adapter will detect if the value of the SESSIONID is valid. If the value of the SESSIONID is valid, then the section labeled as CONSTRUCTOR INFO in the above image can be ignored. The XML schema can be simplified, as shown in the following image.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Liquid XML Studio - FREE Community Edition 7.1.6.1440 (http://www.liquid-technologies.com) -->
<tns:IWAY version="2013" xsi:schemaLocation="urn:iwaysoftware:ibse:jul2003:divideBy C:\bin\divideBy_request.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:iwaysoftware:ibse:jul2003:divideBy">
  <CONSTRUCTOR sessionId="5e145833-1056-462a-a3c5-17149d65563a">
  </CONSTRUCTOR>
  <METHOD>
    <divideBy location="/Math.dll/Math.Math/divideBy(System.Int32,System.Int32)">
      <a>4806</a>
      <b>2</b>
    </divideBy>
  </METHOD>
</tns:IWAY>
```

Otherwise, the value of the SESSIONID will be ignored and the section labeled as CONSTRUCTOR INFO in the previous image will be used by the iWay .NET Technology Adapter.

Known Issues and Limitations

This appendix describes known issues and limitations for iWay .NET Technology Adapter.

In this appendix:

- ☐ [Supported and Unsupported Areas](#)
 - ☐ [Tested Application Scope](#)
-

Supported and Unsupported Areas

This section provides a list of features that are supported and not supported.

Note: This is not a complete list and anything not on this list is implied to be not supported. For more information, contact your customer support representative.

.Net Application Essentials

The following is a list of areas that are not supported:

- ☐ Widows Store, Widows Forms, XAML, any display methods
- ☐ Dynamic assemblies
- ☐ Network I/O
- ☐ SOAP Serialization
- ☐ Write to console

Using .NET serialization classes are not supported for integration.

.Net Data and Modeling

The following list is arranged by features that are not supported, not supported for integration, and not supported Entity Framework restrictions (.NET areas).

Not Supported

- ☐ WCF

Not Supported for Integration

- ☐ Writing to a file or XML file

Not Supported Entity Framework (.NET areas) Restrictions

- ☐ Windows Presentation Foundation and Windows forms
- ☐ Common Client technologies
- ☐ Windows Service applications
- ☐ Parallel and asynchronous processing
- ☐ Windows Communication Foundation
- ☐ Windows Identity Foundation
- ☐ Windows Workflow Foundation
- ☐ Platforms other than Windows client or server

.Net Framework 4.5 and 4 Scope List

The following is a technology support list for the .NET Framework that is arranged by features that are supported, not supported, limited support, limited and legacy support, and partially supported.

Supported

- ☐ 64-bit application development
- ☐ Application domains
- ☐ Assemblies
- ☐ Collections
- ☐ Common language runtime (CLR)

Not Supported

- ☐ .NET for Windows Store applications
- ☐ Accessibility
- ☐ Add-ins
- ☐ ASP.NET

- ☐ Assembly binding redirection

- ☐ Asynchronous programming

- ☐ Code DOM

Limited Support

- ☐ .NET Framework Class Library

- ☐ Common type system

Limited and Legacy Support

- ☐ Attributes

Partial Support

- ☐ ADO.NET

Configuration

This section lists configurations that are arranged by features that are supported, not supported, limited support, partially supported, and implicit only.

Supported

- ☐ Exceptions

- ☐ Generics

- ☐ Files and streams

- ☐ Interoperability

- ☐ Side-by-Side Execution in the .NET Framework

Not Supported

- ☐ Configuring Applications

- ☐ Data Service

- ☐ Debugging, tracing, and profiling

- ☐ Deploying applications

- ☐ Designers and the design environment

- ☐ Directory services
- ☐ Dynamic Language Runtime (DLR)
- ☐ GDI+
- ☐ Compressing files
- ☐ Image file handling
- ☐ Working with Images, Bitmaps, Icons, and Metafiles
- ☐ Images
- ☐ Lazy initialization
- ☐ Managed Extensibility Framework (MEF)
- ☐ Media and multimedia:
 - ☐ Graphics and Multimedia in Window Presentation Foundation
 - ☐ Graphics and Multimedia Portal
- ☐ Memory-mapped files
- ☐ Moving user interface elements
- ☐ MSBuild
- ☐ Network programming
- ☐ Out-of-band (NuGet) releases
- ☐ Parallel programming
- ☐ Portable Class Library
- ☐ Silverlight
- ☐ Transaction processing
- ☐ UI Automation
- ☐ WCF Data Services
- ☐ Windows Communication Foundation
- ☐ Windows Forms

- ☐ Windows Forms controls
- ☐ Windows Identity Foundation
- ☐ Windows Presentation Foundation (WPF)
- ☐ Windows services
- ☐ Windows Store applications
- ☐ Windows Workflow Foundation (WF)
- ☐ Zip files and archives

Limited Support

- ☐ Data access
- ☐ Globalization and localization
- ☐ I/O
- ☐ LINQ (Language-Integrated Query)
- ☐ Reflection
- ☐ Security in the .NET Framework
- ☐ Serialization
- ☐ Threading
- ☐ Windows Runtime
- ☐ XML documents and data

Partial Support

- ☐ Events

Implicit Only

- ☐ Garbage Collection

Classes in .NET are passed by reference, and cannot be used for integration as a result. Structures are passed by value, and the results can be passed between Java and .NET.

Native types in the common type library are supported by default. Defined types must have a constructor if instance-based or declared as static if not.

Serialization of data between Java and .NET is possible, but direct serialization or synchronization of objects is not currently supported. The underlying APIs of Java and .NET do not support the level of object graph serialization required.

The Entity Framework uses a conceptual model to access an underlying data object. Because the model may have any type of design, some model types are not useful with the adapter. Some models are designed for update only, or access only by a data control object in a form. The ideal framework model for the adapter returns an object or contains an object query. Then object query can be serialized through a structure and array list for integration.

Common Language Runtime Scope

The adapter uses the .NET unmanaged COM API to instantiate the .NET Common Language Runtime (CLR). The .NET CLR is not an emulated environment, but communication with the instance is governed by the adapter. There are several languages that can generate CLI code. The adapter has been extensively tested with C# and on a more limited basis with source from Visual Basic and F#. For adapter usage requirements with any other language, contact your customer support representative.

No explicit or implicit guarantee of compatibility or performance with user application components is given with the iWay .NET Technology Adapter. The scope of the framework is too wide and deep to do so. If you have specific questions about supported components of .NET or application compatibility, contact your contact your customer support representative.

Tested Application Scope

The following list provides a summary of the supported functionality that was tested with the iWay .NET Technology Adapter.

- ☐ Browsing Assemblies on a system.
- ☐ Opening an Assembly and exploring public serializable classes.
- ☐ Common type system data types.
- ☐ Invoking methods of an Assembly.
- ☐ Invoking methods of an Assembly by Assembly reference.
- ☐ Triggering a windows process from an Assembly reference.
- ☐ Calling functions in Assemblies (In and Out parameters).
- ☐ Invoking methods with different return types (void, string, int, and structure).
- ☐ Browsing classes and invoking methods on classes nested inside of other classes (nested classes).

- ☐ Using *collections* or *boxed* and *unboxed* properties SET and GET.
- ☐ Assemblies with a very large amount of classes and methods.
- ☐ Accessing Microsoft SQL Server versions 2008 and 2008 R2 using the .NET SQL Server provider.
- ☐ Dynamic build of SQL Server connection strings.
- ☐ Using stored SQL Server connection strings.
- ☐ Validating SQL Server connection strings.
- ☐ Validating SQL Server commands.
- ☐ Dynamically attaching a database to a running instance of SQL Server.
- ☐ Using *static* methods to invoke stored SQL statements on SQL Server (heap).
- ☐ Using *instance* methods to pass SQL commands to SQL Server (stack).
- ☐ Accessing database classes from a static assembly reference (heap).
- ☐ Accessing database classes from a instance assembly reference (stack).
- ☐ Calling database procedures retrieving up to 20,000 records.
- ☐ Common Type System Data Types:
 - ☐ C# Type
 - ☐ .NET Framework Type
 - ☐ bool System.Boolean
 - ☐ byte System.Byte
 - ☐ sbyte System.SByte
 - ☐ char System.Char
 - ☐ decimal System.Decimal
 - ☐ double System.Double
 - ☐ float System.Single
 - ☐ int System.Int32

- ☐ uint System.UInt32
- ☐ long System.Int64
- ☐ ulong System.UInt64
- ☐ object
- ☐ System.Object
- ☐ short System.Int16
- ☐ ushort System.UInt16
- ☐ string System.String
- ☐ Structures built from these types
- ☐ One dimension Array
- ☐ One dimension ArrayList
- ☐ Dictionary
- ☐ DataTable

A class is a reference to a location in memory. As a result, classes require a specific serializer helper to write to XML. A structure is a data object and can be directly serialized to XML.



Feedback

Customer success is our top priority. Connect with us today!

Information Builders Technical Content Management team is comprised of many talented individuals who work together to design and deliver quality technical documentation products. Your feedback supports our ongoing efforts!

You can also preview new innovations to get an early look at new content products and services. Your participation helps us create great experiences for every customer.

To send us feedback or make a connection, contact Sarah Buccellato, Technical Editor, Technical Content Management at Sarah_Buccellato@ibi.com.

To request permission to repurpose copyrighted material, please contact Frances Gambino, Vice President, Technical Content Management at Frances_Gambino@ibi.com.



iWay

/ iWay .NET Technology Adapter User's Guide

Version 7.0.x and Higher

DN3502306.0418

Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898