RabbitMQ Service Model

Table of Contents

An AMQP Binding for WCF

The Windows Communication Foundation (WCF) enabled protocol independent service oriented applications to be built; RabbitMQ.net extends the framework by providing a *Binding* and *Transport Binding Element* over AMQP. In the language of WCF, a *Binding* is a stack of *Binding Elements* which control all aspects of the service's communication (for example, Security, Message Format and Transactions). A specialized kind of Binding Element, the *Transport Binding*

Element specifies the protocol to be used for communication between a service and its clients (for example WS-HTTP, MSMQ or .Net Remoting over TCP).

The RabbitMQ Binding provides *OneWay* ('Fire and Forget'), *TwoWay* (Request/Reply) and *Duplex* (Asynchronous Callback) communication over AMQP with WS-ReliableSessions, WS-AtomicTransactions and Text (SOAP 1.2) message encoding. The binding can be configured from imperative code or using the standard WCF Configuration model.

A Transport Binding Element is also supplied and can be used in the construction of Custom Bindings[1] if the channel stack provided by the RabbitMQ Binding is insufficient. The transport binding must be configured with a Broker Hostname, Broker Port and Protocol Version prior to use.

Service Addressing

Services hosted using the RabbitMQ binding must be hosted at addresses under the **soap.amqp** scheme. The **amq.direct** exchange is used. The service name must not be omitted.

serviceAddress = "soap.amqp:///" serviceName

Known Limitations and Status

1. A TwoWay or Duplex service cannot have SessionMode = SessionMode.NotAllowed since a Reliable Session is required to maintain the reply channel.
2. Only SOAP Formatting is available, other formatters can be specified by building a CustomBinding on top of the RabbitMQTransportBindingElement
3. Service queue parameters (e.g. durability) are not configurable
4. Network faults are not reported to the binding

The RabbitMQ WCF binding has limited flexibility compared to the RabbitMQ .NET/C# AMQP client library. You are advised to use the .NET/C# AMQP client library if you require greater flexibility (e.g. control over durability of service queue) or if you require long-term support. No further development is planned for the WCF binding and support will cease after the next major release of RabbitMQ.

Building the Binding and Samples

The RabbitMQ binding to WCF and associated samples can be built automatically using Nant. For more information about Nant, visit http://nant.sourceforge.net/. To build the library and Sample Applications from a console window, change to the RabbitMQ.net drop location and execute:

---

1 http://msdn2.microsoft.com/library/system.servicemodel.channels.custombinding

nant build-wcf

nant wcf-examples

Alternatively, users of Microsoft Visual Studio should open the following C# projects:

src\wcf\RabbitMQ.ServiceModel\RabbitMQ.ServiceModel.csproj

src\wcf\Test\RabbitMQ.ServiceModel.Test.csproj

The WCF Binding is built into the RabbitMQ.ServiceModel.dll assembly and copied to the *bin* directory of the RabbitMQ.ServiceModel project and the sample applications are built into the *bin* directory of the Test project. To run the sample applications (verifying the build and your environment configuration) execute the RabbitMQ.ServiceModel.Test.exe application.

By default, the sample applications use a test broker which must be running at localhost. You can modify the broker hostname and port by opening and editing the appSettings section of the Application Configuration file (App.Config) for the Test Project.

The ABCs of WCF

Each Windows Communication Foundation service is built from three components, an *Address*, *Behaviours* and a *Contract*. For more information, see http://wcf.netfx3.com/.

Contract

A service contract is an interface decorated with the ServiceContractAttribute[2] and has one or more methods (or property accessors) decorated with the OperationContractAttribute. Typically the contract exists in an assembly that can be shared between client and server applications.

```
[ServiceContract]
public interface ICalculator
{
        [OperationContract]
        int Add(int x, int y);

        [OperationContract]
        int Subtract(int x, int y);
}
```

Behaviour

The contract for a service specifies what the operations the service agrees to provide, the behaviour specifies the implementation for that service. A behaviour is a class implementing the contract and optionally decorated with the ServiceBehavior Attribute.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)]
```

[2] http://msdn2.microsoft.com/library/System.ServiceModel.ServiceContractAttribute

```
public sealed class Calculator : ICalculator
{
        public int Add(int x, int y)
        {
                return x + y;
        }

        public int Subtract(int x, int y)
        {
                return x - y;
        }
}
```

Address

For a service to be useful, it must be reachable and therefore hosted. The two common hosting scenarios for WCF services are IIS and ServiceHost. IIS Hosting is untested and unsupported by the RabbitMQ binding and using System.ServiceModel.ServiceHost is the recommended hosting path. A service host instance is constructed with the type of service behaviour being hosted and a description of the endpoint(s) it will be published on. The endpoints consist of Addresses (e.g. soap.amp:///MyService) and Bindings; they may be specified directly as constructor arguments in imperative code or declaratively through WCF configuration files, both are supported by RabbitMQ.net.

Sample Services

The sample services referred to in this section are located in the src\wcf\Test project.

One Way Services

Operations on a service can be marked as One Way; this means there will be no response from the service when the operation completes. One Way operations always have return type void and have an OperationContractAttribute with IsOneWay set equal to true decorating them.

```
[OperationContract(IsOneWay=true)]
void Log(LogData entry);
```

If a service only contains one way operations the RabbitMQ binding can be used in an optimized *OneWayOnly* mode. In this mode, no reply queue is created for responses to be sent back to the client and the client does not listen for responses from the service. To enable OneWayOnly mode set the binding property or use the oneWay configuration attribute.

```
<rabbitMQBinding>
        <binding name="rabbitMQConfig"
                 hostame="localhost"
                 port="5672"
                 username="guest"
                 password="guest"
                 virtualHost="/"
                 oneWay="true"
                 maxmessagesize="8192"  />
</rabbitMQBinding>
```

The OneWayTest sample application is a simple logging service. Clients submit log entries to a server which displays them on the console. It demonstrates one way RPC over AMQP, SOAP Encoding to transmit complex data types over the wire and Singleton Instance Context Mode.

Two Way Services

Typically a service operates in a bi-directional, two way fashion where requests from the client are synchronously executed and a response returned to the caller. To support these services, the RabbitMQ binding uses the CompositeDuplexBindingElement[3], which constructs a uniquely named reply queue on the broker. Two Way services are not supported by the binding when it is in OneWayOnly mode.

The TwoWayTest sample application is a calculator service, whose operations take a pair of integers and return a third.

---

[3] http://msdn2.microsoft.com/library/system.servicemodel.channels.compositeduplexbindingelement

Sessionful Services

Each call to a service can be considered independent of all others with the service maintaining no state, often a more useful service maintains some state between calls. The RabbitMQ binding supports WS-ReliableSessions enable the object instances used to service requests to have a session-long lifetime and be associated with a single client session.

The SessionTest sample application is a cart service, allowing items to be added to a cart and a total calculated.

Duplex Services

A call to a two way service might start a long running process (for example, aggregating prices from a list of suppliers) and whilst the client requires a response, it is desirable that the client is not blocked for the duration of the call; instead, an asynchronous call is desired. Duplex services[4] allow the service to make calls to the client, and have a contract whose ServiceContractAttribute specifies a CallbackContract[5] type.

```
[ServiceContract(CallbackContract=typeof(IOrderCallback))]
public interface IOrderService
```

Duplex services are supported by the RabbitMQ binding because its channel stack includes the composite duplex binding element, they are not supported in OneWayOne mode. The DuplexTest sample application is an ordering service, which makes a callback to the client when an order is fulfilled.

Using the RabbitMQ Binding

Services

The recommended hosting scenario for services over AMQP is self hosting using System.ServiceModel.ServiceHost[6]. The ServiceHost **must** specify a base or absolute endpoint address under the soap.amqp scheme. An endpoint should then be added to the service using the RabbitMQBinding.

```
service = new ServiceHost(
        typeof(Calculator),
        new Uri("soap.amqp:///"));


service.AddServiceEndpoint(
        typeof(ICalculator),
        new RabbitMQBinding(
```

[4] http://msdn2.microsoft.com/library/ms731064.aspx
[5] http://msdn2.microsoft.com/library/system.servicemodel.servicecontractattribute.callbackcontract
[6] http://msdn2.microsoft.com/library/System.ServiceModel.ServiceHost

```
                "localhost",
                5672,
                "guest",
                "guest",
                "/",
                8192,
                Protocols.AMQP_0_9_1),
            "Calculator");
```

Clients

The recommended pattern for connecting to a service is by deriving from either ClientBase<T> or DuplexClientBase<T>. For Duplex Clients, the InstanceContext must be specified.

Configuration Files

Specifying details like the protocol version and broker address in source code tends to result in services which are very hard to manage and deploy. To avoid this, WCF provides a configuration mechanism using application configuration files (App.Config). The configuration file must be applied to the host or client assembly (typically an executable) and **not** to a library which contains the service contract or behaviours. To declaratively configure a service, the RabbitMQBindingSection must be imported into the system.serviceModel section of the configuration file:

```
  <extensions>
    <bindingExtensions>
      <add
        name="rabbitMQBinding"
        type="RabbitMQ.ServiceModel.RabbitMQBindingSection,
RabbitMQ.ServiceModel, Version=1.0.110.0, Culture=neutral,
PublicKeyToken=null"/>
    </bindingExtensions>
  </extensions>
```

With the extension imported, the rabbitMQBinding can be declared[7] and configured:

```
  <bindings>
    <rabbitMQBinding>
      <binding
        name="rabbitMQConfig"
        hostname="localhost"
        port="5672"
        maxmessagesize="8192"
        version="AMQP_0_9_1" />
    </rabbitMQBinding>
```

[7] Note that in Visual Studio, IntelliSense® will incorrectly report that the rabbitMQBinding is an invalid child of the bindings Node.

```
</bindings>
```

Service Configuration

A service is configured by declaring the contract, endpoint and binding. Multiple services and bindings can be specified in a single configuration file.

```
<services>
  <service name="Calculator">
    <host>
      <baseAddresses>
        <add baseAddress="soap.amq:///" />
      </baseAddresses>
    </host>
    <endpoint
      address="Calculator"
      binding="rabbitMQBinding"
      bindingConfiguration="rabbitMQConfig"
      contract="ICalculator"/>
  </service>
</services>
```

To run the service, simply create a new ServiceHost instance passing in the service behaviour (as specified in config).

```
host = new ServiceHost(typeof(Calculator));
host.Open();
```

Client Configuration

To build a client whose settings are derived from configuration, expose a constructor for your ClientBase<T> derived class calling the ClientBase(string).

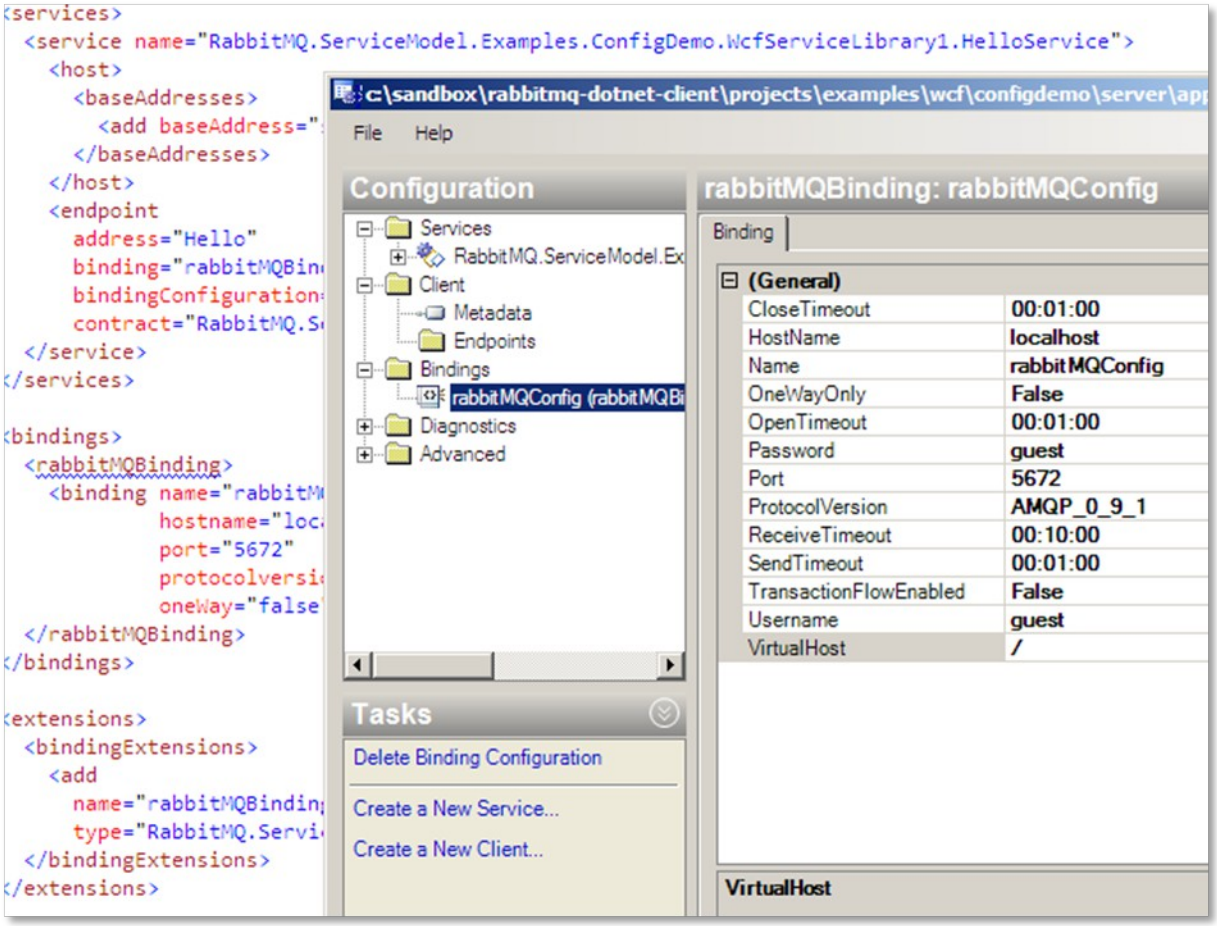
```
public class CalculatorClient : ClientBase<ICalculator>, ICalculator
{

    public CalculatorClient(string configurationName)
        : base(configurationName) { }
```

Construct the class passing the client endpoint name as specified in configuration.

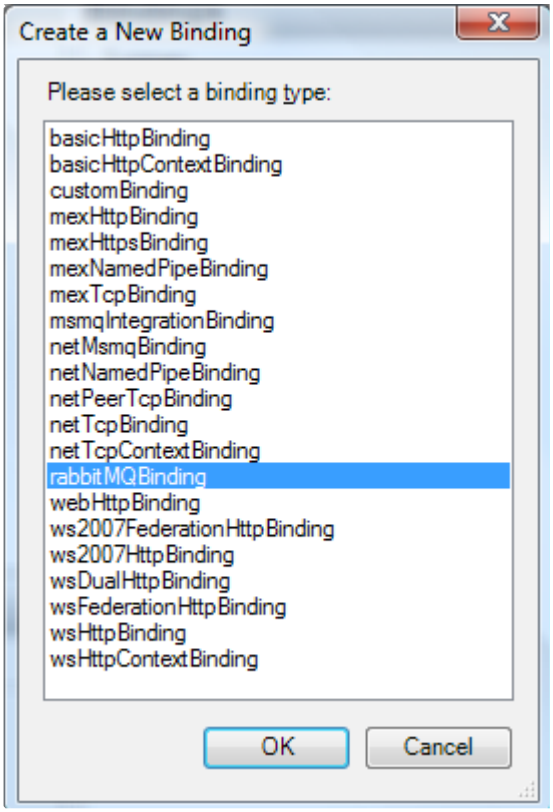
```
<client>
  <endpoint address="soap.amq:///Calculator"
            binding="rabbitMQBinding"
            bindingConfiguration="rabbitMQConfig"
            contract=" ICalculator"
            name="AMQPCalculatorService" />
</client>
```

The RabbitMQ WCF libraries also have full support for the WCF Configuration Editor Tool (available from the .Net framework 3.0 SDK and Visual Studio 2008).



Using the WCF Configuration Utility for Editing a RabbitMQ Service Configuration File

To add a RabbitMQ binding to an existing (or new) configuration file, open the service configuration editor[8] and expand the Advanced > Extensions > binding extensions node. Click the 'new' button and select the RabbitMQBindingSection type from the RabbitMQ.ServiceModel.dll assembly. The rabbitMQ binding for WCF is now available within the configuration tool. You can now create a binding for any services configured within the file over AMQP by right clicking the Bindings node and choosing 'New Binding'. In the following list (right) select rabbitMQbinding and click OK.



Integration with the configuration system and toolset means that updating existing applications to benefit from the scalability and robustness of RabbitMQ is very straightforward.

[8] This is installed as part of the Windows SDK to %ProgramFiles%\Microsoft SDKs\Windows\v6.0A\bin

Reference

RabbitMQBinding

A windows communication foundation binding over AMQP. By default, the RabbitMQBinding generated the following binding element stack:

- TransactionFlowBindingElement
- ReliableSessionBindingElement
- CompositeDuplexBindingElement
- TextMessageEncodingBindingElement
- RabbitMQTransportBindingElement

Public Constructors

Name	Description
RabbitMQBinding()	Creates a new instance of the RabbitMQBinding class initialized to use the Protocols.DefaultProtocol. The broker hostname must be set before use
RabbitMQBinding(Hostname, Port)	Uses the default protocol and the broker specified by the given hostname and port
RabbitMQBinding(Hostname, Port, IProtocol)	Uses the broker and protocol specified
RabbitMQBinding(Hostname, Port, Username, Password, VirtualHost, MaxMessageSize, IProtocol)	Uses the broker, login, virtual host, maximum message size and protocol specified
RabbitMQBinding(IProtocol)	Uses the specified protocol. The broker hostname must be set before use.

Declared Public Properties

Name	Description
HostName : string	Specifies the broker hostname that the binding should connect to.
Port : int	Specifies the broker port that the binding should connect to.
BrokerProtocol : IProtocol	Specifies the version of the AMQP protocol that should be used to communicate with the broker
ConnectionParameters : ConnectionParameters	Gets the parameters used to connect to the broker
MaxMessageSize: long	Specifies the maximum encoded message size
OneWayOnly : bool	Specifies whether or not the CompositeDuplex and ReliableSession binding elements are added to the channel stack.

ReliableSession : System.ServiceModel.ReliableSession	Gets the reliable session parameters for this binding instance
Scheme: string	Gets the scheme used by the binding, soap.amqp
TransactionFlow : bool	Determines whether or not the TransactionFlowBindingElement will be added to the channel stack

RabbitMQTransportBindingElement

Represents the binding element used to specify AMQP transport for transmitting messages.

Public Constructors

Name	Description
RabbitMQTransportBindingElement()	Creates a new instance of the RabbitMQTransportBindingElement Class using the default protocol.

Declared Public Properties

Name	Description
HostName : string	Specifies the broker hostname that the binding should connect to.
Port : int	Specifies the broker port that the binding should connect to.
BrokerProtocol : IProtocol	Specifies the version of the AMQP protocol that should be used to communicate with the broker
MaxReceivedMessageSize : long	The largest receivable encoded message
Scheme: string	Gets the scheme used by the binding, soap.amqp

RabbitMQBindingConfigurationElement

Represents the configuration for a RabbitMQBinding. The configuration element should be imported into the client and server configuration files to provide declarative configuration of a AMQP bound service.

Public Constructors

Name	Description
RabbitMQBindingConfigurationElement()	Creates a new instance of the RabbitMQBindingConfigurationElement Class.
RabbitMQBindingConfigurationElement(string)	Creates a new instance of the RabbitMQBindingConfigurationElement Class initialized with values from the specified configuration.

Declared Public Properties

Name	Description
HostName : string	Specifies the broker hostname that the binding should connect to.
Port : int	Specifies the broker address that the binding should connect to.
MaxMessageSize : long	Specifies the maximum encoded message size
OneWayOnly : bool	Specifies whether or not the CompositeDuplex and ReliableSession binding elements are added to the channel stack.
Password : string	Password to use when authenticating with the broker
Protocol : IProtocol	Gets the protocol version specified by the current configuration
ProtocolVersion : string	Specifies the protocol version to use when communicating with the broker
TransactionFlowEnabled : bool	Specifies whether or not WS-AtomicTransactions are supported by the binding
Username : string	The username to use when authenticating with the broker
VirtualHost : string	The virtual host to access.