**Extensibility - Inspecting WCF Messages**

****

**Table of Contents**

[**Introduction on Extensibility 2**](#_Toc467074550)

[**WCF Service Stack 3**](#_Toc467074551)

[**Message Inspector 5**](#_Toc467074552)

[**Configure Message Inspectors 7**](#_Toc467074553)

[**Step 1 7**](#_Toc467074554)

[**Step 2 9**](#_Toc467074555)

[**Step 3 11**](#_Toc467074556)

[**Output 11**](#_Toc467074557)

[**Summary 12**](#_Toc467074561)

# Introduction on Extensibility

Extensibility = Extend the functionality.

Guys, before we jump into WCF Extensibility, I would like to give you, a brief on Extensibility which is nothing but a software design principle to extend the software functionality, with a minimal impact or no impact into the existing functionalities. In simpler words, Feature X would not affect by the extended Feature Y.

WCF pipeline is easily extendable allowing the developer to extend in various locations in the Client and the Service to implement the custom logic if needed. You can extend your own binding, channel and behavior. Let us discuss this in detail.

# WCF Service Stack

I would suggest you to read my old article **WCF Runtime Architecture and SOA implementation** to understand the WCF pipeline architecture completely before you jump into Extensibility. In the snapshot given below, you can understand the processes involved in sending the data from the client to the Server and vice versa.

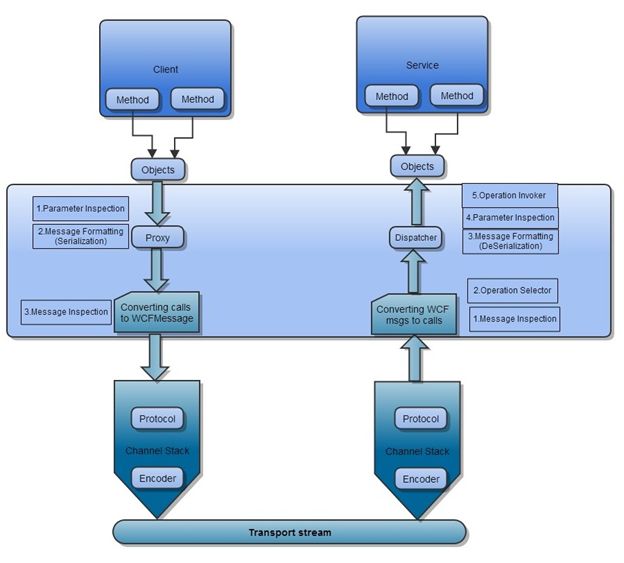


Figure 1.0 WCF Runtime Architecture

The data travels from the client application to the stream by converting calls to WCF message and it converts the messages into calls and objects on the Server side. There are five Extensible Endpoints on the Server side.

1. Message Inspection
2. Operation Selector
3. Message Formatting (De-Serialization)
4. Parameter Inspection and
5. Operation Invoker

WCF has multiple extensible points, which allow you to modify runtime behavior. You can ask me a question like, what is the usage of extensible points ? Why should I modify the runtime behavior using the extensible points? The following snapshot answers these questions

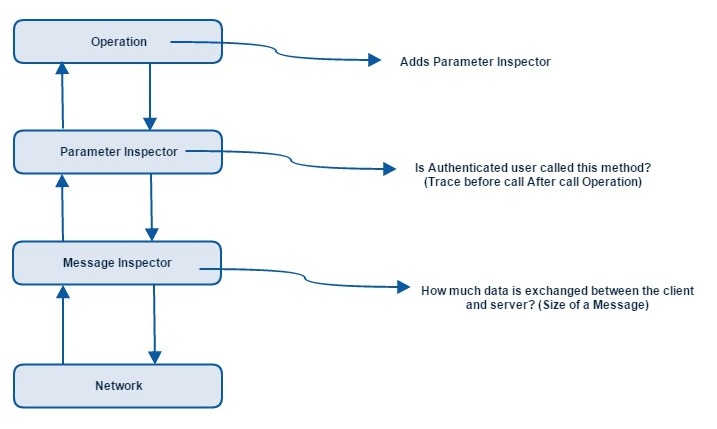


Figure 2.0 WCF Extensible points

From the Message Inspector, you can find that

* How much data exchanged between the Client and the Server?
* What is the Size of the Message?
* How long will it take to reach the Message from the Client to the Server?
* How much time the Service Method will take to process it?

From the Parameter Inspector, you can find that

* Which clients are making calls to the Service?
* Is the User an Authenticated one?

In this article, we are going to see Message Inspector in detail

# Message Inspector

Message Inspector is the most extensively used extensible point, which allows the developer to log, modify and replace the complete message. The following two interfaces help us to log and modify the messages in WCF pipeline

* IDispatchMessageInspector
* IClientMessageInspector

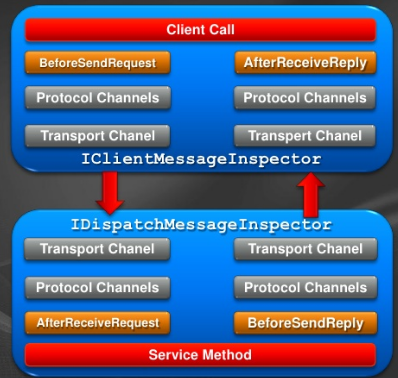


Figure 3.0 Message Inspectors

The implementation of **IDispatchMessageInspector- AfterRecieveRequest** method helps us to inspect the incoming message, which received on the wire and before it dispatched to the Service method. The implementation of **IDispatchMessageInspector-BeforeSendReply** method helps us to receive the outgoing message from the Service method and before it sent on the wire.

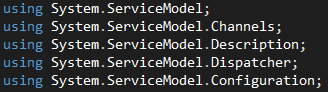
The implementation of **IClientMessageInspector-BeforeSendRequest** method helps us to receive and inspect the message, which created from the client application and before it sent to the wire. **IClientMessageInspector-AfterRecieveReply** method helps us to receive the message (response message) from the Server and before it reaches the client application. Let us talk about it in detail.

# Configure Message Inspectors

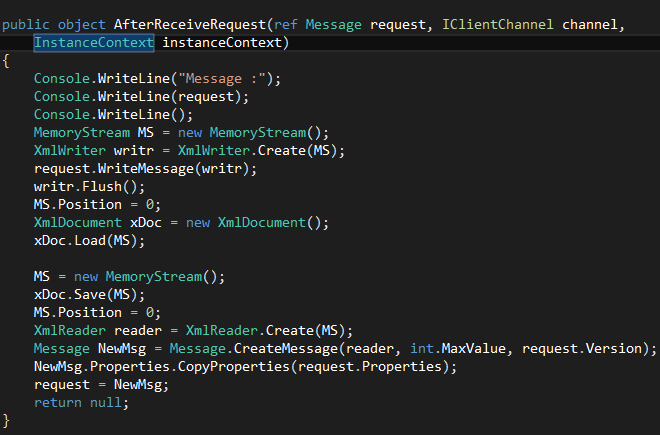
### Step 1

Create a separate class library and refer the following assemblies and namespaces

* System.ServiceModel
* System.Runtime.Serialization



Next, Implement IDispatchMessageInspector methods **AfterRecieveRequest** and **BeforeSendReply.  AfterRecieveRequest**helps us to identify, log, modify or Replace the incoming message, which received from the client and before it, send to the Service method.



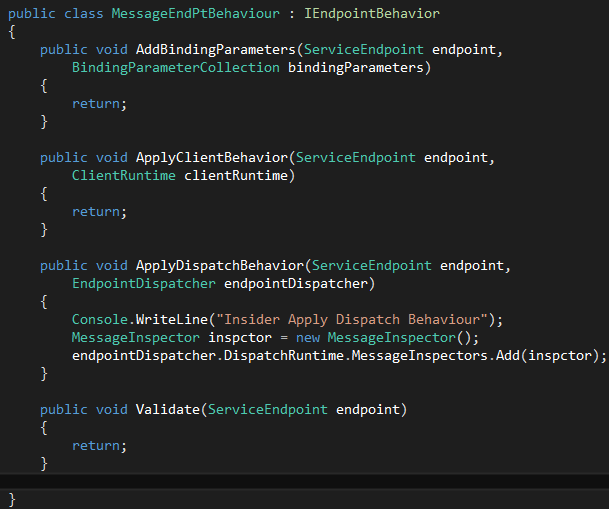
In the snapshot given above, you can see the parameters Message (reference param), Channel and InstanceContext. Channel parameter helps us to identify through which channel (basicHttp, netTcp) the message received. With the help of InstanceContext parameter, you can find out any InstanceContext(PerCall,PerSession) associated with this Service. The message object passed by the reference. The beauty of the Message object is that once you consumed (read, write, copied) the object, it cannot be used again. The message object is a read once the stream, once consumed it cannot used again. The responsibility of the Message inspector’s responsibility is to recreate the message. The source above shows that how the message is consumed and recreated. In our case, I have not changed the original data rather I have recreated the original data. The following methods would cause the message to consume.

* CreateBufferedCopy
* GetBody
* GetReaderAtBodyContents
* WriteBody
* WriteBodyContents
* WriteMessage

Accessing the message headers and properties does not have any issues, Accessing the message object once will be treated as consumed, and it cannot be consumed again

### Step 2

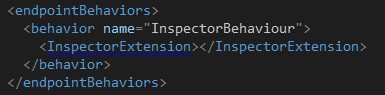
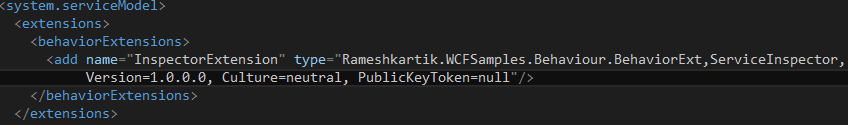
In order to add Message Inspector we have to create custom behavior extension class, which is derived from IServiceBehaviour or IEndpointBehaviour, In our case, I will implement IEndpointBehaviour



In the snapshot above, you can find that IEndpointBehaviour is implemented ApplyDispatchBehaviour and other methods. You need to create the Message Inspector object and add into Endpoint Dispatcher. Finally, you need to create the Behavior extension class which derived from BehaviourExtensionElement and override the CreateBehaviour method and BehaviourType property to return the MessageEndPtBehaviour objects (custom behavior objects).

### Step 3

As the last step, we have to update behavior settings in the App.config file as well

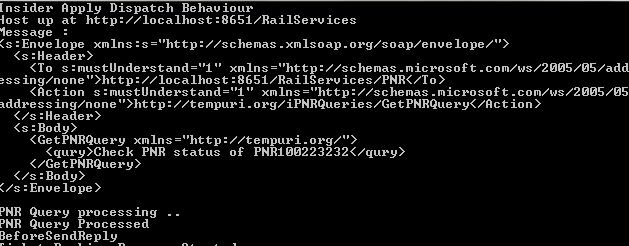


In the snapshot given above, Please mention the name of the Extension, type and Assembly name. Assign the extension name "InspectorExtension" into the endpointBehaviour as well.

Please refer the attached sourcecode for more details

# Output

Please refer the ServiceInspector Library in the Host application and Host the Server.If you clearly notice the output once the Client launched the application you can find that before the log I placed in the Service method triggers, log I placed in the ServiceInspector gets triggered.



# In the snapshot above, "PNR Query processing" and "PNR Query processed" are the logs placed inside

# the Service Method gets triggered only after the execution of AfterReceiveRequest method in

# the ServiceInspector.  Please refer the attached sourcecode for more details

# Summary

To inspect the messages in the WCF pipe line IDispatchMessageInspector and IClientMessageInspector interfaces implement methods AfterRecieveRequest and BeforeSendReply in the serviceside to recieve and inspect the incoming messages and outgoing messages. WCF run time behaviour can be easily extended with the help of Extensibility.