

DotNet Connector CookBook

Table Of Contents

Chapter 1	Introduction.....	5
1.1	WHAT IS A COOKBOOK?.....	6
1.2	WHO CAN USE THIS COOKBOOK?	6
1.3	PRE-REQUISITES	6
1.4	HOW TO USE THIS COOKBOOK.....	6
Chapter 2	Deployment of SAP Generic Connectors.....	7
2.1	STEPS FOR DEPLOYING SAP GENERIC CONNECTOR.....	8
2.2	DEPLOYMENT OF SAP APPLICATION SPECIFIC PROJECT ON IIS.	12
Chapter 3	Deployment for SP Generic Connectors.....	14
3.1	STEPS FOR DEPLOYING SP GENERIC CONNECTOR.....	15
Chapter 4	Implementation & Deployment of SP Application Specific Connector	21
4.1	STEPS FOR DEPLOYING SP APPLICATION SPECIFIC CONNECTOR.....	22
Chapter 5	Connectors in SSDG Architecture	26
5.1	REAL WORLD APPLICATION-TRADEMARK REQUEST.....	26
5.2	SCHEMA FOR TRADEMARK REQUEST AND RESPONSE.....	27
5.3	Executing Synchronous Submit Request at SAP end.....	29
Recipe 1	Synchronous Submit Request Execution.....	29
5.4	Processing of Trademark Request at SP end sent in Synchronous Mode.....	36
Recipe 2	Retrieve Request from SSDG in case of Synchronous Submit Request.	36
5.4.1	Steps involved in the implementing the Solution	37
5.5	Retrieving the result sent by SP Application to SAP.....	45
Recipe 3	Response from Trademark application.....	45
5.6	Executing Asynchronous Submit Request.....	47
Recipe 4	Asynchronous Submit Request Execution.....	47
5.7	Retrieving Acknowledgement from SSDG	50
Recipe 5	Retrieve the Acknowledgement from SSDG.....	50
5.8	Processing of Trademark Request at SP end sent in Asynchronous Mode	52
Recipe 6	Retrieve Request from SSDG in case of Asynchronous Submit Request	52
5.9	Submitting the Response from SP Application to SSDG	55
Recipe 7	Submit Response from SP Application to SSDG for Asynchronous Request.....	55
5.10	Sending the Acknowledgement back from SSDG to SP	58
Recipe 8	Submit Acknowledgement from SSDG to SP	58
5.11	Polling the SSDG to retrieve the Result.	60
Recipe 9	Poll the SSDG to retrieve the result submitted by SP Application.....	60
5.12	Deleting Request submitted to SSDG.....	64
Recipe 10	Executing Delete Request.....	64
5.13	Listing Requests submitted to SSDG.	67

Recipe 11 Executing List Request.....	67
Chapter 6 Inter-Gateway Communication	70
6.1 Executing Asynchronous Submit Request at SAP end.....	71
Recipe 12 Asynchronous Submit Request Execution.....	71
6.2 Polling the SSDG to retrieve the Result	72
Recipe 13 Poll the SSDG to retrieve the result submitted by SP Application.....	72
6.3 Submitting the Response from SP Application to SSDG.....	73
Recipe 14 Submit Response from SP Application to SSDG for Asynchronous Request.....	73
6.4 Deleting Requests submitted to SSDG.....	74
Recipe 15 Executing Delete Request.....	74
Additional Resources	76
Index	77

Table Of Figures

<i>Figure 1 : Create a New Web Application in Visual Studio</i>	9
<i>Figure 2 : Adding Reference to SAP Generic DLL.....</i>	10
<i>Figure 3 : Reference added as SAP Generic DLL and Log4NetDLL.....</i>	11
<i>Figure 4 : Renaming App.config file to Web.config for use in Web Applications</i>	12
<i>Figure 5 : Creating a new Virtual Directory in IIS</i>	13
<i>Figure 6 : Naming the directory as SAPApplication</i>	14
<i>Figure 7 : Creating Virtual Directory in IIS.....</i>	15
<i>Figure 8 : Naming of created Virtual directory in IIS</i>	16
<i>Figure 9 : Referring to folder for SPGeneric Connector.....</i>	17
<i>Figure 10 : Making Authentication related changes in IIS</i>	18
<i>Figure 11 : Testing the deployment of SP Generic Connector</i>	19
<i>Figure 12 : Editing Configuration Settings</i>	20
<i>Figure 13 : Creating Virtual Directory for SP Application Specific connector in IIS</i>	22
<i>Figure 14 : Naming Virtual Directory</i>	23
<i>Figure 15 : Referring SP Application folder.</i>	24
<i>Figure 16 : Making changes for Authentication Setting in IIS</i>	25
<i>Figure 17 : SAP calling trademark application in synchronous mode.....</i>	30
<i>Figure 18 : SAP asynchronously submits request to trademark application through SSDG.</i>	47
<i>Figure 19 : Trademark application response to SSDG for Asynchronous Request</i>	55
<i>Figure 20 : SAP Polling SSDG for response</i>	61
<i>Figure 21 : SAP calling delete request for previously submitted requests.</i>	64
<i>Figure 22 : List Request by SAP to track status of all requests and response from SSDG</i>	67
<i>Figure 23 : Inter-gateway communication between SAP and SP</i>	70
<i>Figure 24 : Request Forwarded by SSDG A to SSDG B after resolving the address.....</i>	71
<i>Figure 25 : Submit Poll by SAP on SSDG A to get address of SSDG B</i>	72
<i>Figure 26 : Response Submitted by SP Application to SSDG B</i>	73
<i>Figure 27 : Delete request on SSDG A</i>	75
<i>Figure 28 : Delete request on SSDG B</i>	75

Table of Code Snippets

<i>Snippet 1 : Schema for Trademark Request</i>	27
<i>Snippet 2 : Schema for Trademark Response</i>	28
<i>Snippet 3 : Snippet of code generated from Trademark Request Schema</i>	31
<i>Snippet 4 : Snippet to define method for XML conversion</i>	32
<i>Snippet 5 : Snippet to Implement SAP Application Connector</i>	35
<i>Snippet 6 : Snippet of code generated form Trademark Response Schema</i>	40
<i>Snippet 7 : Snippet defining de-conversion method of XML element and XML converter for</i> 42	
<i>Snippet 8 : Snippet for implementing synSubmitRequest() method</i>	44
<i>Snippet 9 : Snippet for retrieving Response received from Trademark Application</i>	46
<i>Snippet 10 : Snippet for using Generic Connector to submit Asynchronous Request to SSDG</i>	49
<i>Snippet 11 : Snippet for Retrieving Acknowledgement from SSDG</i>	51
<i>Snippet 12 : Snippet for implementing asynSubmitRequest() method</i>	53
<i>Snippet 13 : Snippet for Submitting Response to SSDG</i>	57
<i>Snippet 14 : Snippet for retrieving acknowledgement against submitResponse () from SP</i>	59
<i>Snippet 15 : Snippet for Poll Operation on SSDG</i>	62
<i>Snippet 16 : Snippet for Delete Operation on SSDG</i>	66
<i>Snippet 17 : Snippet for List Request Operation on SSDG</i>	69

Chapter 1 Introduction

1.1 What is a CookBook?

Typically, a cookbook is a collection of recipes or instructions that explain how to perform a certain task and what is needed to execute it. The basic objective of this Cookbook is to provide an essential problem-solving resource. Just as cookbooks for meals contain step-by-step directions for creating various dishes, this book provides recipes for using SAP and SP generic connector. Recipes are provided for quick reference about invoking the functionalities provided by generic connectors and use of these functionalities by application specific connector.

While deploying the SAP and SP application connectors, the developer is likely to face certain queries or problems. This cookbook tries to span all such problems that the developer may face during the phase of implementing the application connectors along with the solutions for these problems. This cookbook is a collection of code solutions to common and uncommon problems encountered while implementing the application specific connectors in .Net framework.

1.2 Who can use this CookBook?

Developers intending to build and deploy the application connectors in .Net framework may refer to the cookbook for finding solutions to commonly encountered problems. This cookbook is not meant to guide / instruct a developer about building a .Net specific application connector. Developers may seek reference from the Dotnet Connector Manual for instructions on developing the application connectors in .Net framework.

1.3 Pre-requisites

This book is intended for developers with at least some development experience in .Net framework. It is assumed that the reader understands the concepts of enterprise development and the basics of .Net framework and has gone through the DotNet Connector Development manual. The format used here specifically references problems and issues, avoiding the use / explanation of .Net keywords This manual can be read up at any point.

1.4 How to use this CookBook

This cookbook spans a total of 6 chapters each exploring, exploring and resolving the issues faced during implementing and working of the application specific connectors.

Chapter 1: Provides an introduction to what a CookBook essentially is and who can use this CookBook. It gives an overview about how the cookbook can be useful in solving some common issues faced by developers in the process of using application connectors with the generic connectors.

Chapter 2: Extensively covers the deployment of SAP generic connectors in a step-by-step approach.

Chapter 3: Explains the deployment of SP generic connectors in a sequential manner. Also testing of SP generic connector is conducted to ensure that it is up and running by deploying it in IIS.

Chapter 4: Demonstrates the implementation and deployment of SP application connector.

Chapter 5: Provides a real world application example of how request and its subsequent response is communicated between the SAP and SP generic and application specific connectors. Elaborates the flow of request and response on both SAP and SP side for synchronous and asynchronous mode. Also discusses various issues encountered while handling these requests and response and provides solutions to these issues in the form of recipes.

Chapter 6: Discusses the requests and response handled by SAP and SP connectors in an Inter-Gateway scenario.

Chapter 2 Deployment of SAP Generic Connectors

Before we begin with the deployment of SP generic connector, we need to know, how the SAP generic connector functions and is used. SAP Generic connector being a dynamic link library (DLL) needs to be added in the project in which SAP implements application specific connector. Deployment of application specific connector will be as per requirements of the application in the context of SAP application. The following are the steps for using SAP generic connector.

Note: The Web Application used in deployment snapshots is a sample web application. The Web Application is likely to vary in form, format and specifications depending upon nature of services requested. The user will develop a web application that matches the needs of the services required.

2.1 Steps for deploying SAP Generic Connector

- i) Go to Visual Studio->New Project->Web Application

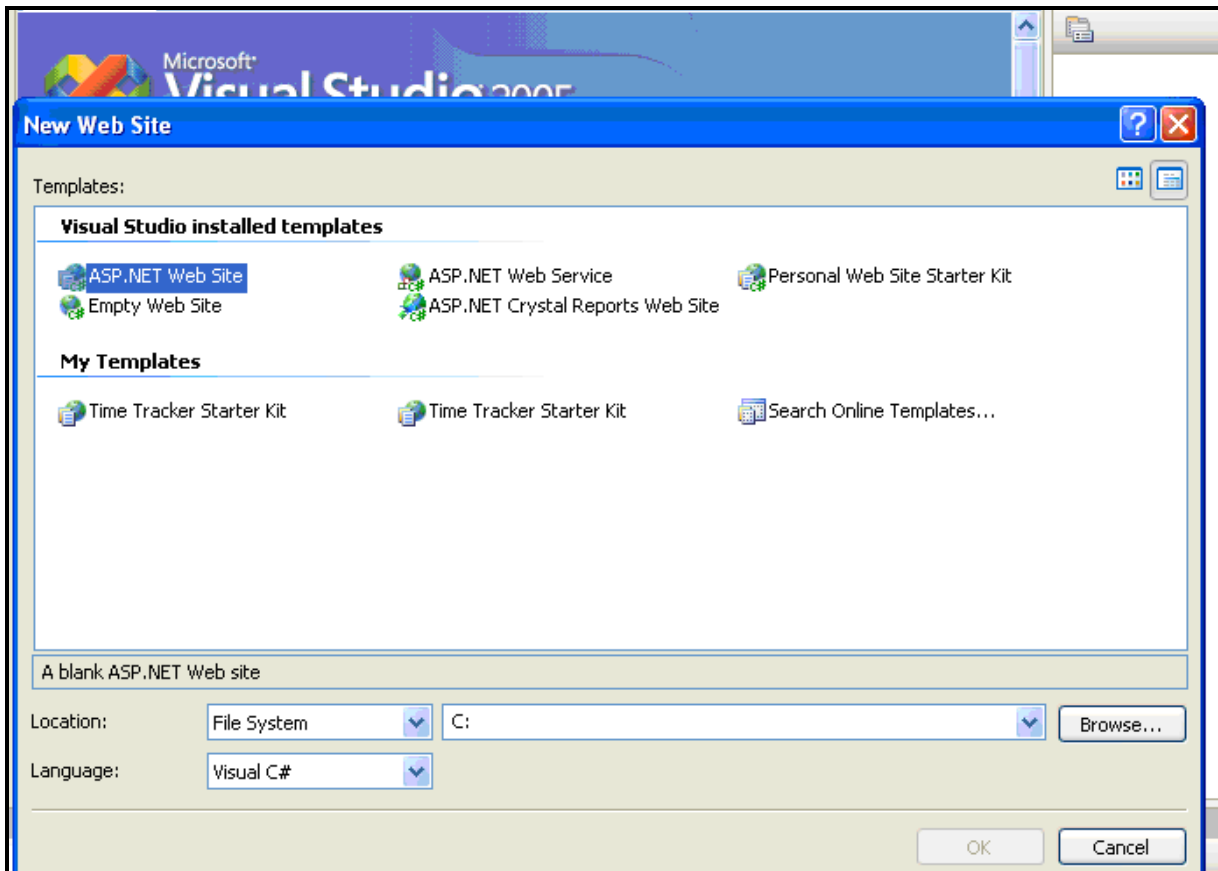


Figure 1 : Create a New Web Application in Visual Studio

- ii) Add reference to SAP Generic DLL

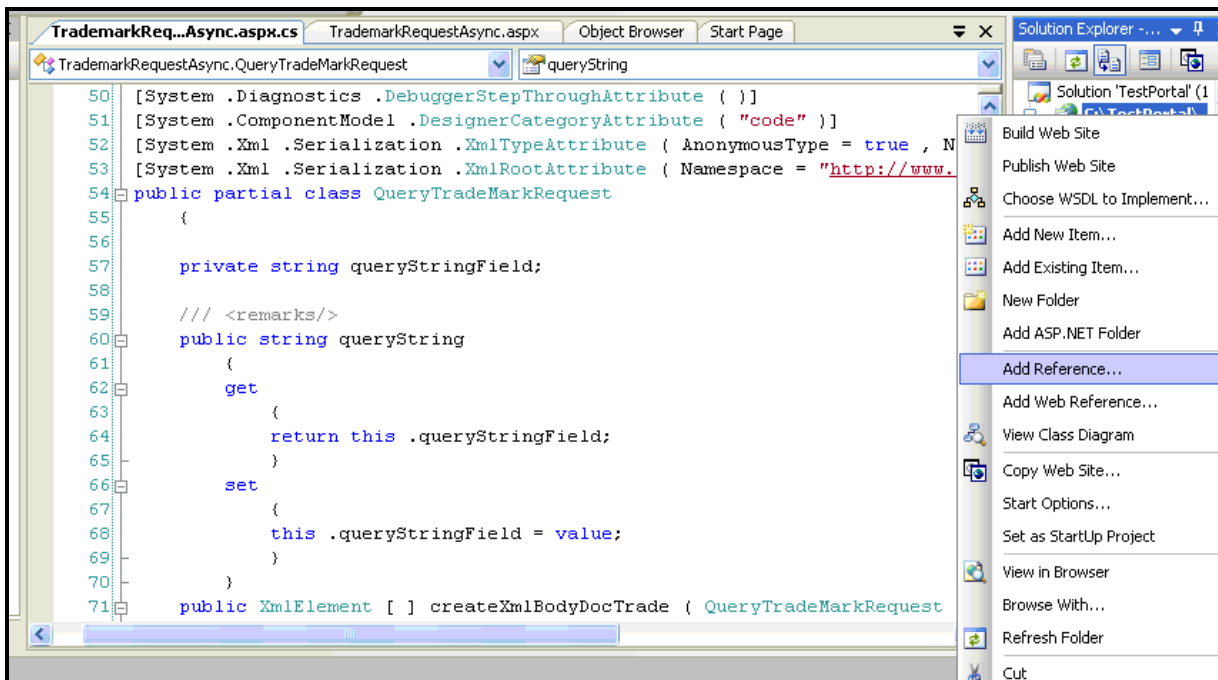


Figure 2 : Adding Reference to SAP Generic DLL

iii) Screenshot given below shows the reference added as SAP generic and Log4Net DLL

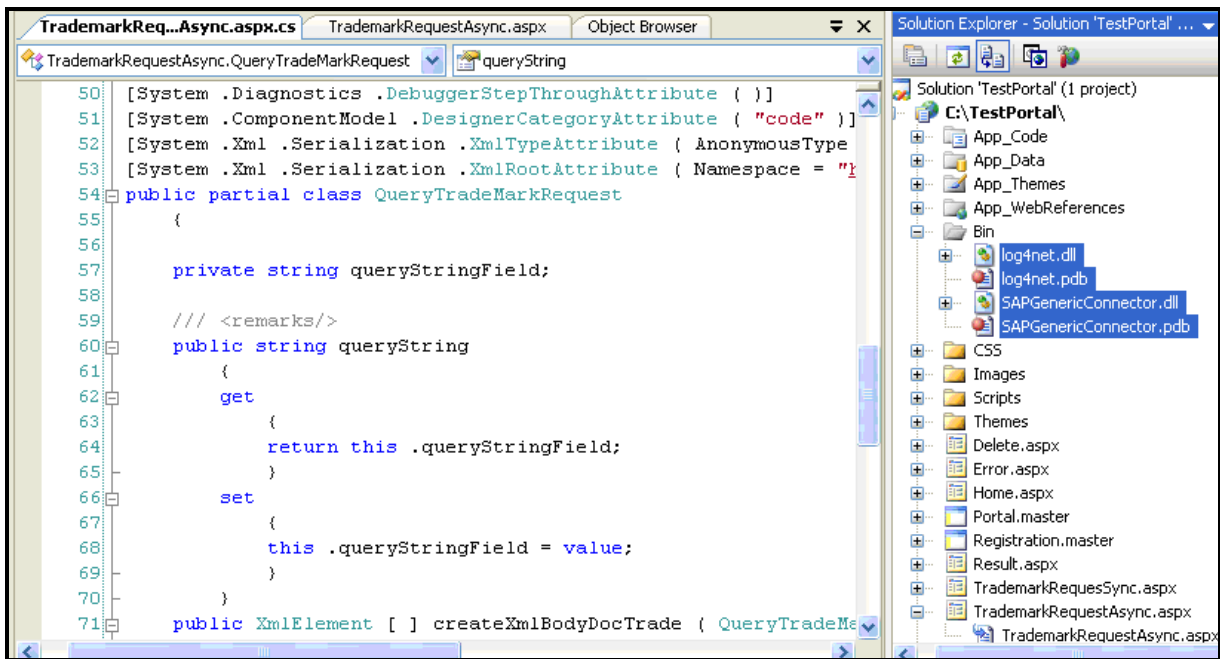
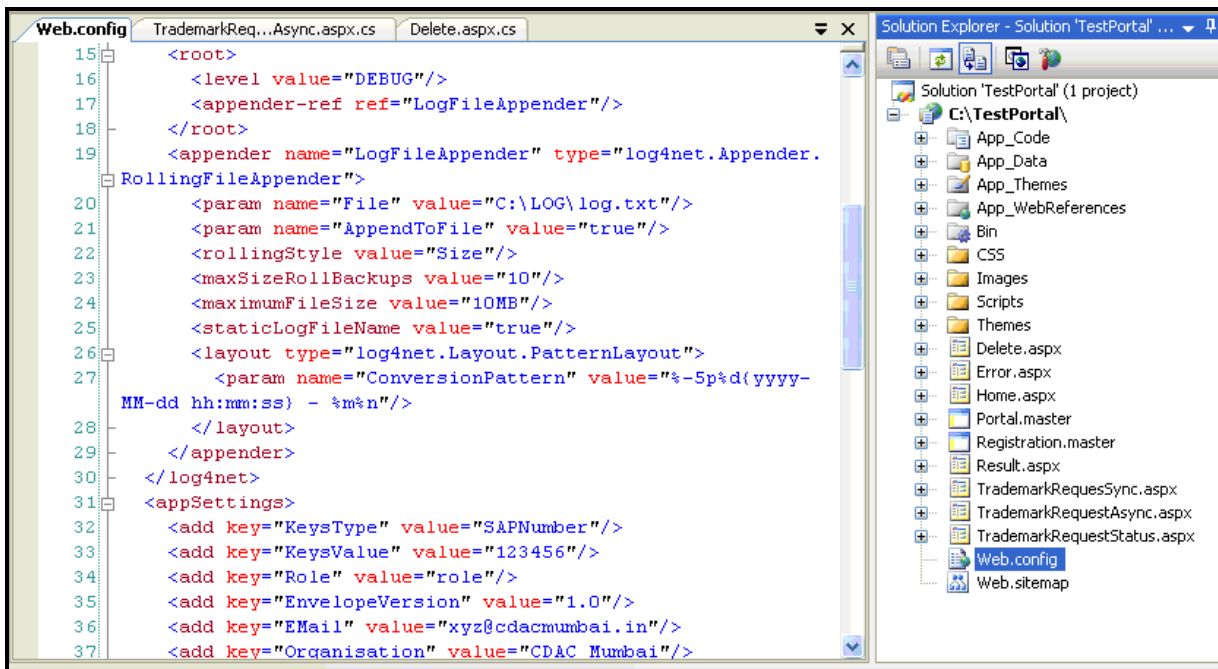


Figure 3 : Reference added as SAP Generic DLL and Log4NetDLL

iv) Include all the required classes to run a particular submission request to gateway.

v) Rename App.config given in DotNet connector manual to Web.config if using in Web based applications and refer it in the project.



The screenshot shows the Visual Studio IDE with the Web.config file open in the editor. The code is as follows:

```
15 <root>
16 <level value="DEBUG"/>
17 <appender-ref ref="LogFileAppender"/>
18 </root>
19 <appender name="LogFileAppender" type="log4net.Appender.
RollingFileAppender">
20 <param name="File" value="C:\LOG\log.txt"/>
21 <param name="AppendToFile" value="true"/>
22 <rollingStyle value="Size"/>
23 <maxSizeRollBackups value="10"/>
24 <maximumFileSize value="10MB"/>
25 <staticLogFileName value="true"/>
26 <layout type="log4net.Layout.PatternLayout">
27 <param name="ConversionPattern" value="%-5p%d(yyyy-
MM-dd hh:mm:ss) - %m%n"/>
28 </layout>
29 </appender>
30 </log4net>
31 <appSettings>
32 <add key="KeysType" value="SAPNumber"/>
33 <add key="KeysValue" value="123456"/>
34 <add key="Role" value="role"/>
35 <add key="EnvelopeVersion" value="1.0"/>
36 <add key="EMail" value="xyz@cdacmumbai.in"/>
37 <add key="Organisation" value="CDAC Mumbai"/>
```

The Solution Explorer on the right shows the project structure for 'TestPortal' (1 project) located at C:\TestPortal\, including folders like App_Code, App_Data, App_Themes, App_WebReferences, Bin, CSS, Images, Scripts, Themes, and various ASPX files.

Figure 4 : Renaming App.config file to Web.config for use in Web Applications

vi) Compile the project

2.2 Deployment of SAP Application Specific project on IIS.

i) Create New Virtual directory in IIS as shown in *Figure 5*.

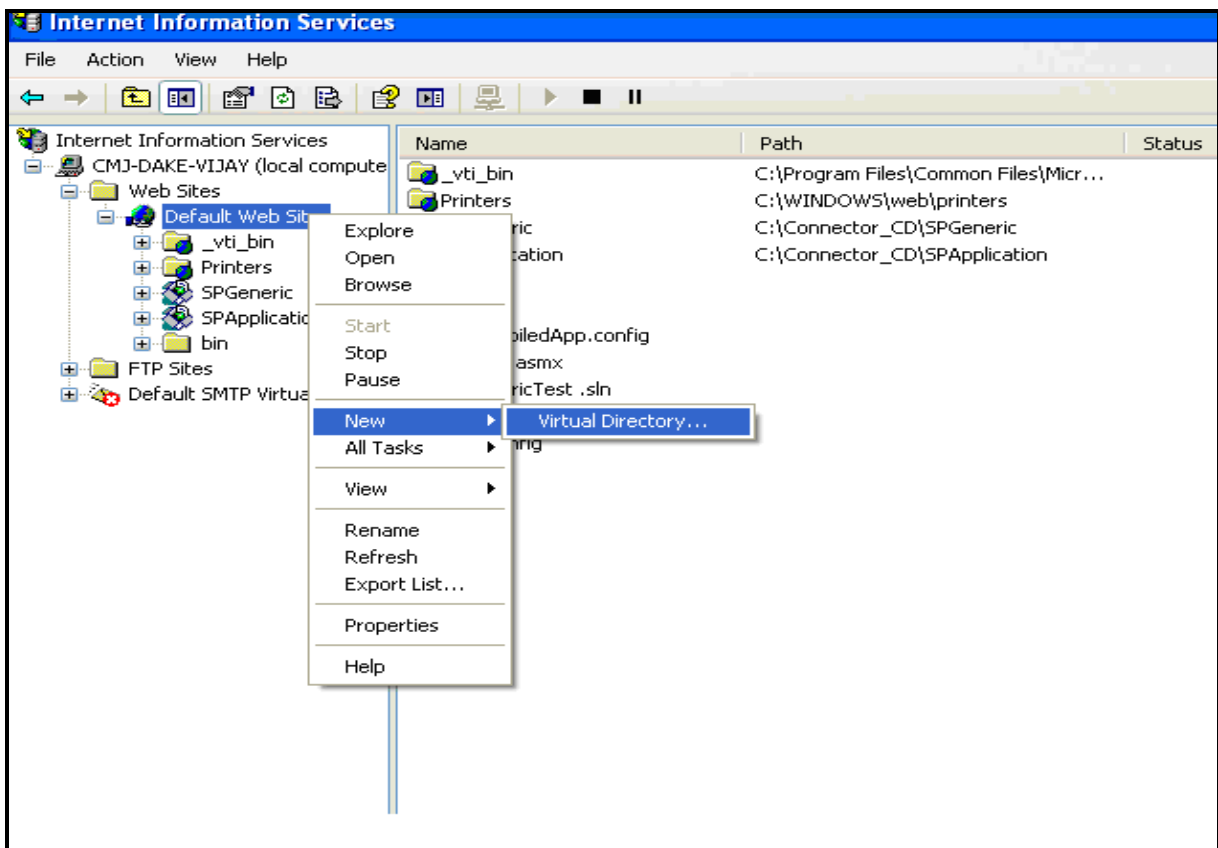


Figure 5 : Creating a new Virtual Directory in IIS

ii) Give name to directory as *SAPApplication* as shown in *Figure 6*.

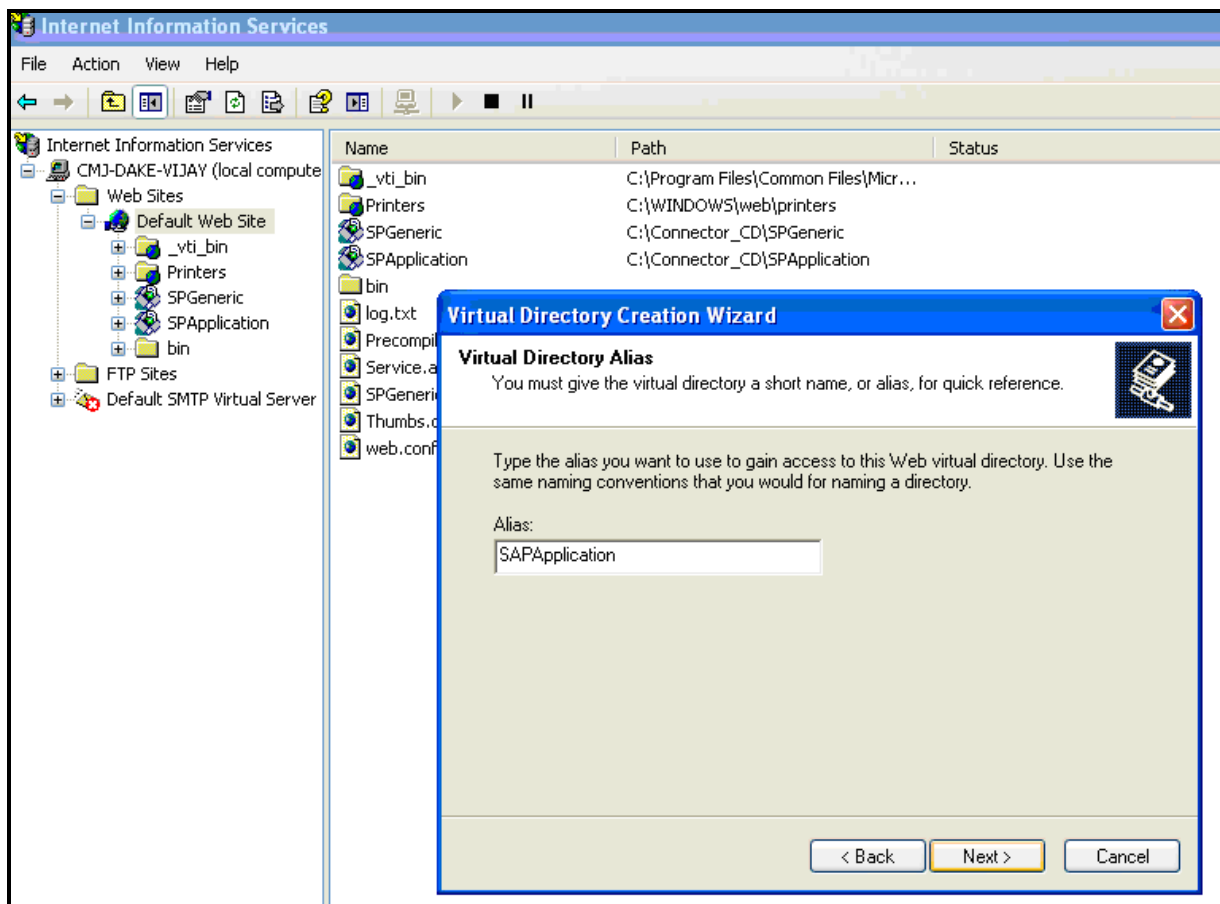


Figure 6 : Naming the directory as *SAPApplication*

- iii) After creating the virtual directory named as *SAP Application* refer it to actual web application on the location where it is actually copied.

Chapter 3 Deployment for SP Generic Connectors

The SP end has the following 2 components that take care of the incoming requests and serve the appropriate response.

SP End Components

- SP Generic connector
- SP Application DLL

Here we will see how the SP generic connector which is a web service is deployed on IIS and what are the steps involved for the same. This section explains the deployment of the SP generic connector in a step by step manner along with the screen shots for every step.

3.1 Steps for deploying SP Generic Connector

- i) Create New Virtual directory in IIS as shown in *Figure 1*.

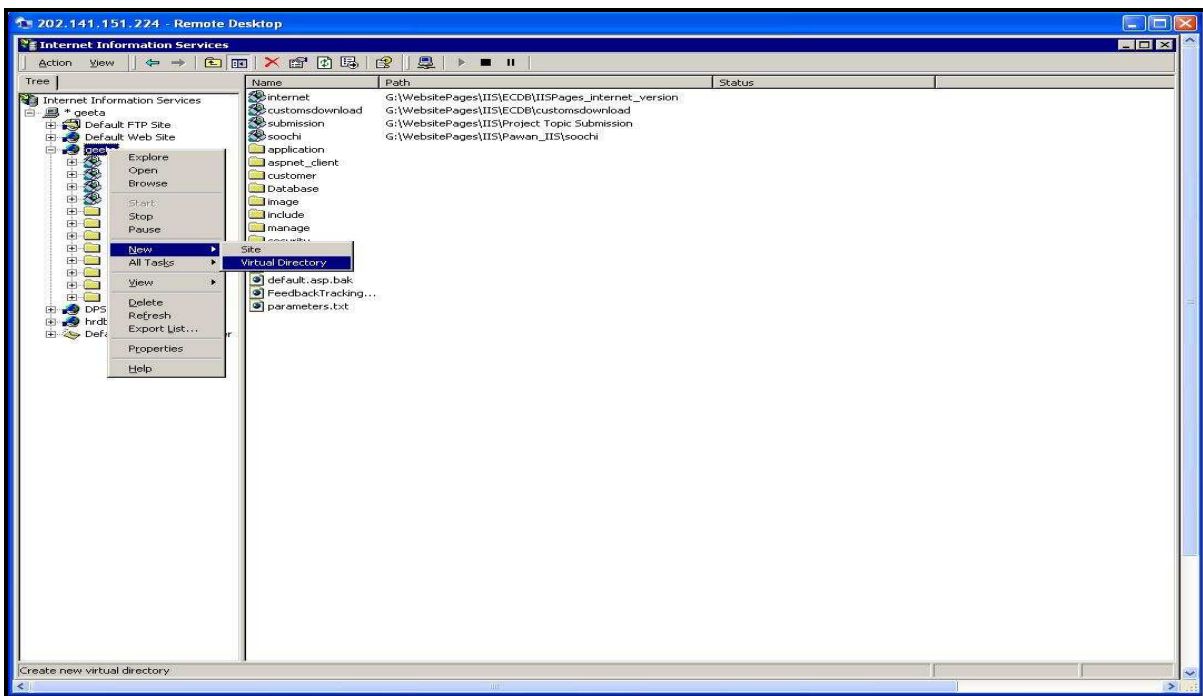


Figure 7 : Creating Virtual Directory in IIS

- ii) Give name to directory as *SPGeneric* as shown in *Figure 2*.

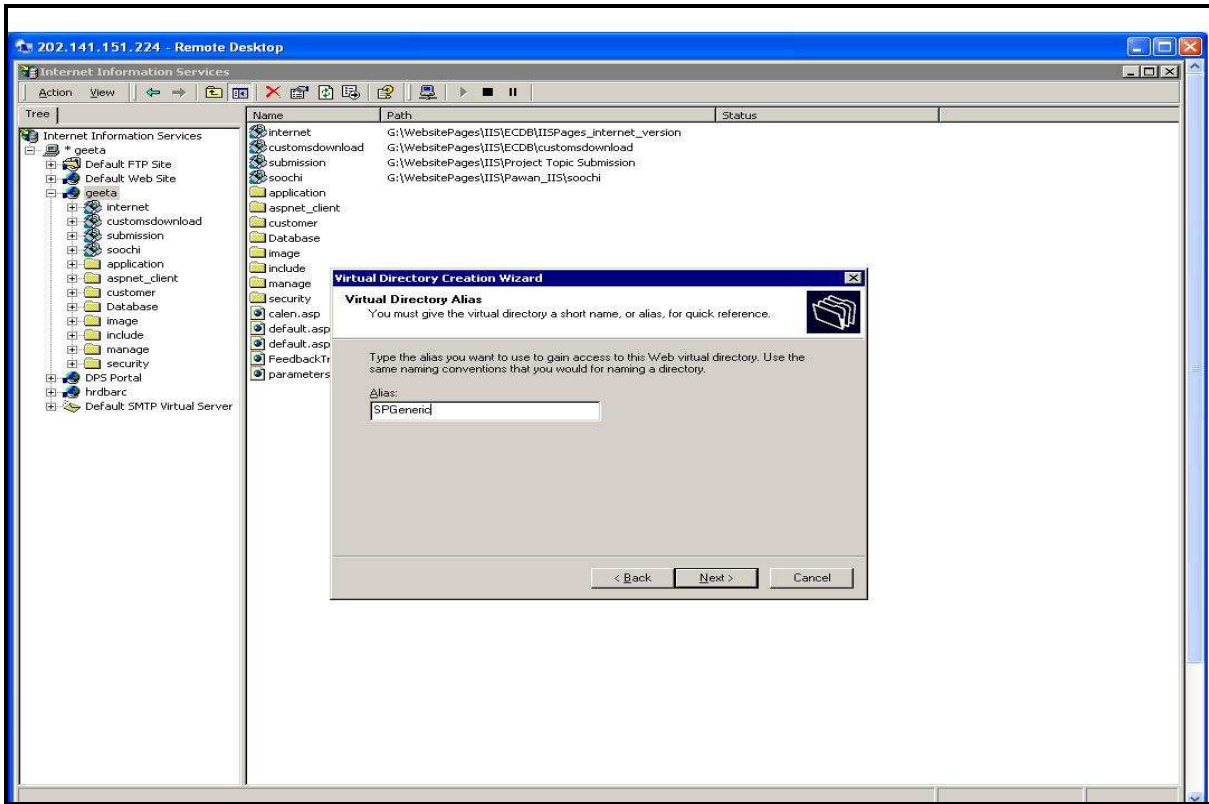


Figure 8 : Naming of created Virtual directory in IIS

- iii) After creating the virtual directory named as *SPGeneric* refer it to SP generic connector on the location where it is actually copied. *Figure 3* shows the directory reference where *SPGeneric* is copied.

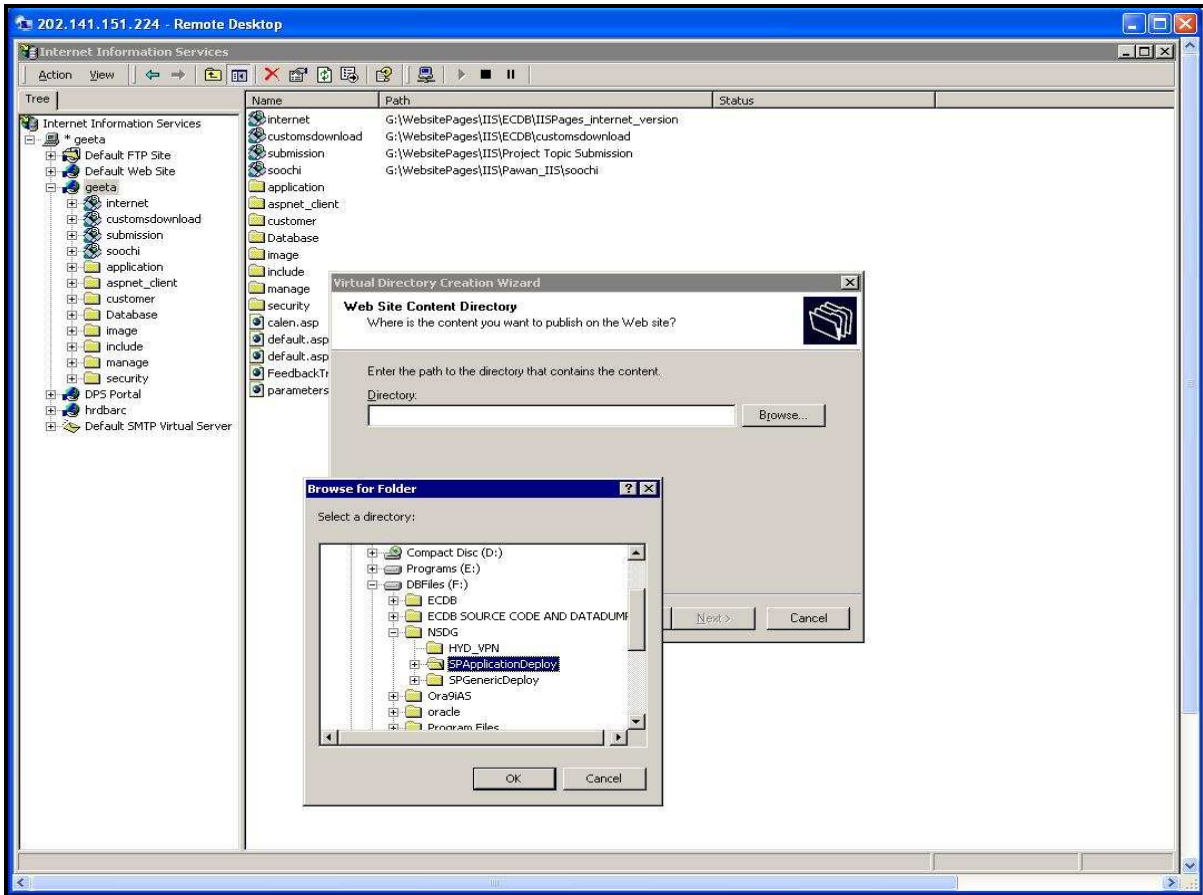


Figure 9 : Referring to folder for SPGeneric Connector

- iv) Once the *SPGeneric* gets deployed make changes in security configuration as shown in the Figure 4.

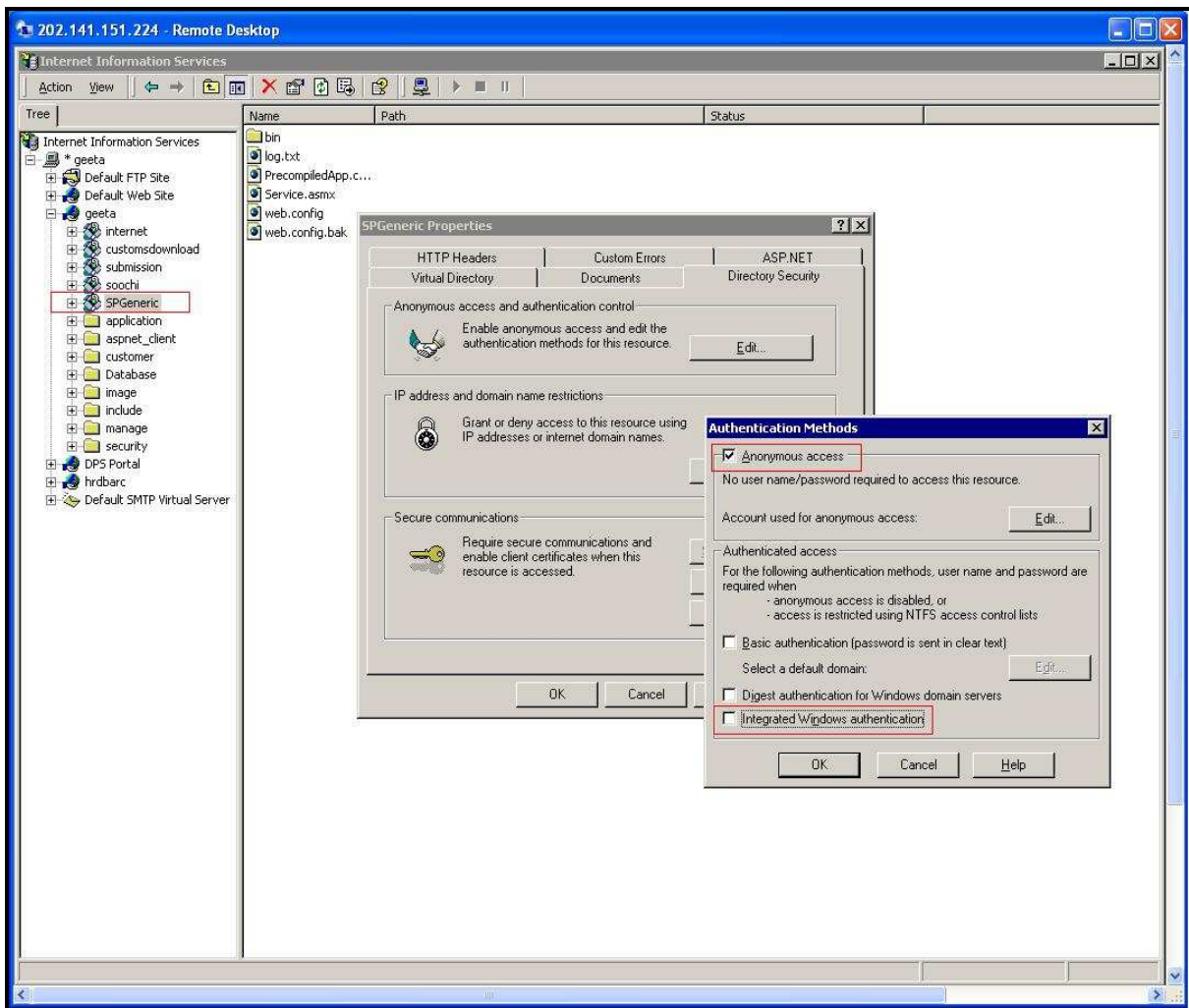


Figure 10 : Making Authentication related changes in IIS

v) Test the deployment in the browser as shown in *Figure 5*.

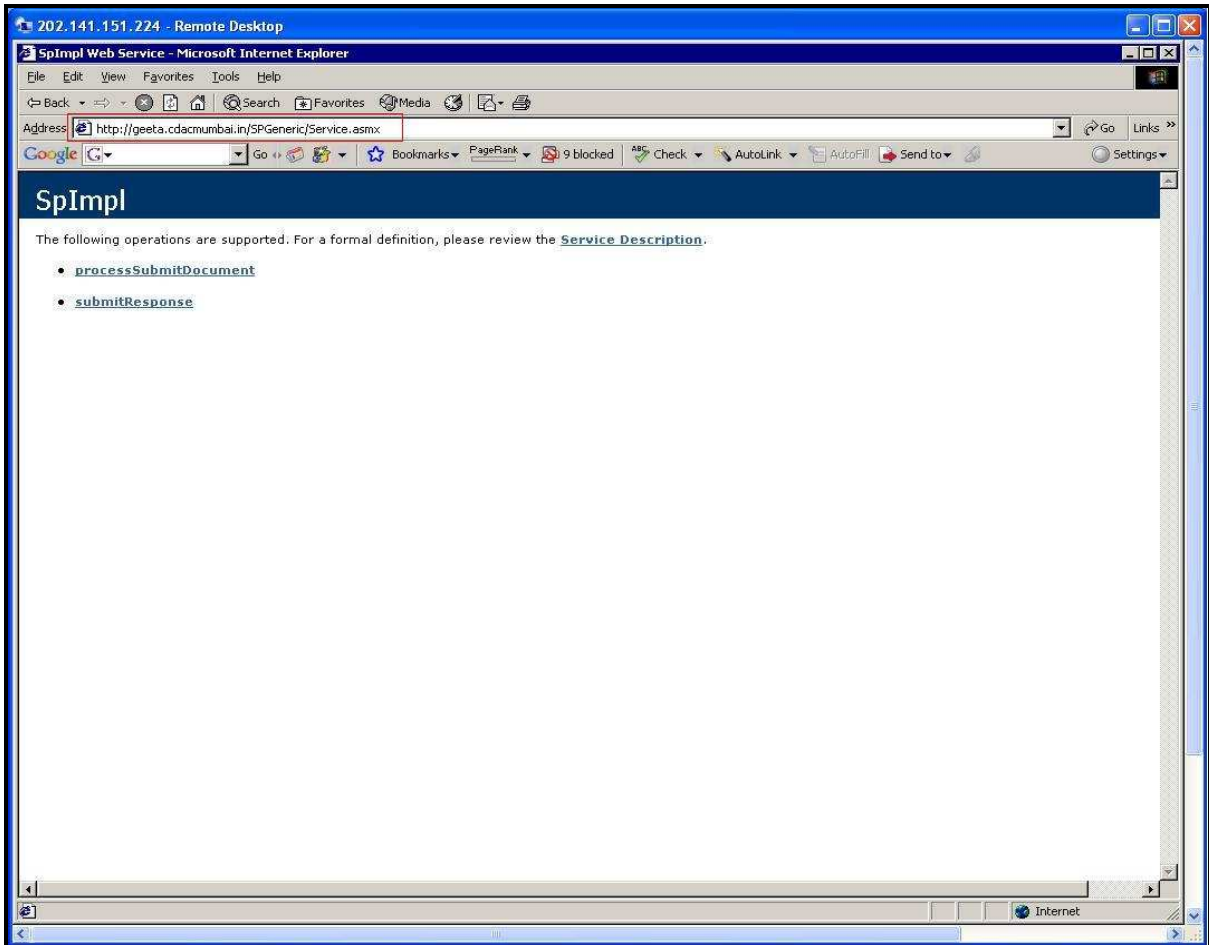


Figure 11 : Testing the deployment of SP Generic Connector

- vi) Mark the configuration details in configuration details so that SP generic connector can access the application specific connector as shown in *Figure 6*. In *deploymentURI*

key mention the path for SP application specific connector i.e. path where SP application specific connector will get deployed.

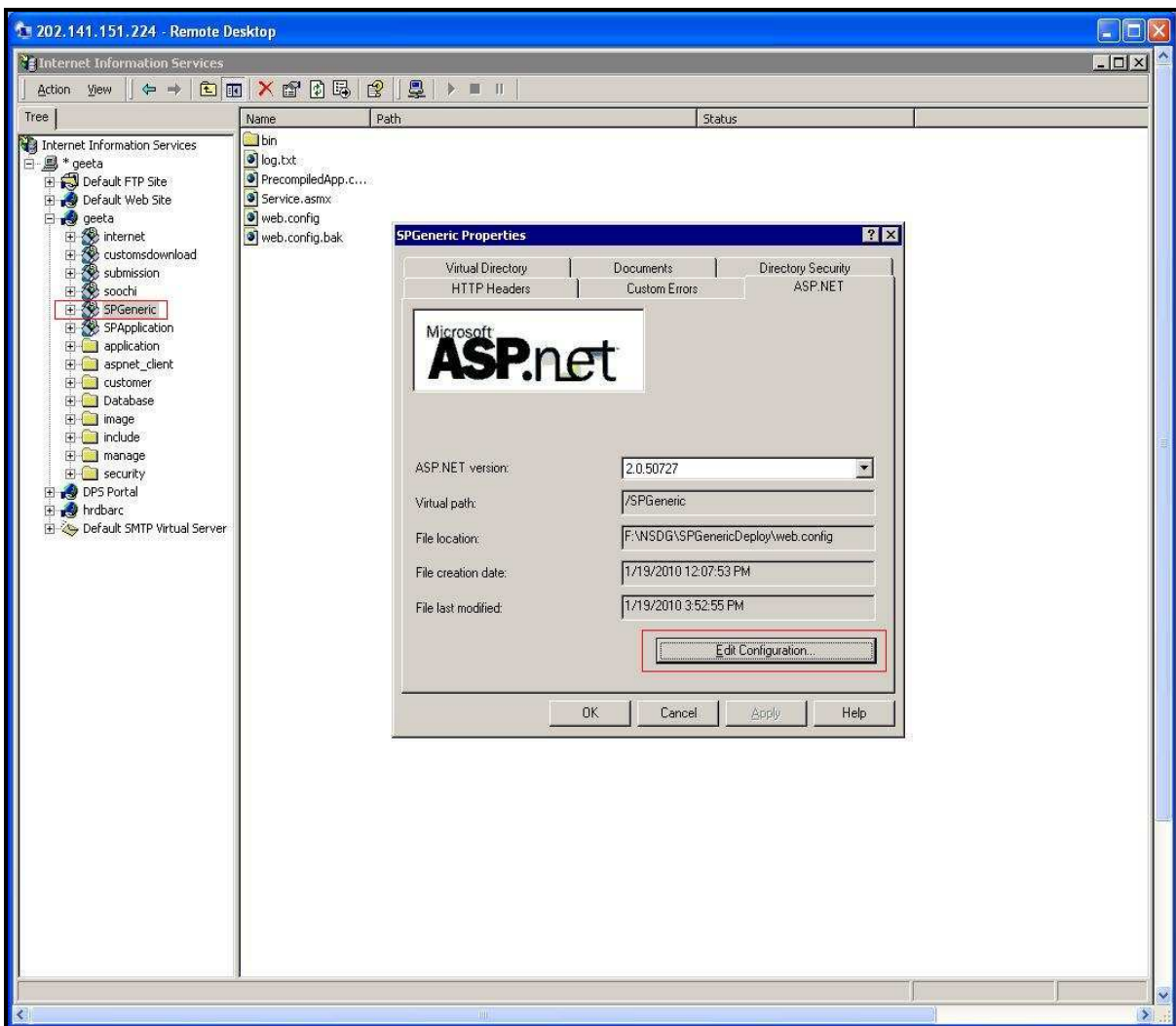


Figure 12 : Editing Configuration Settings

Chapter 4 **Implementation & Deployment of SP Application Specific Connector**

The SP application specific connector will use the DLL provided with SP generic connector.

Implementing SP Application Specific Connectors

- i) Go to Visual Studio->New Project->Web Site->Web Service.
- ii) Add reference to SP Application DLL named *FullClasses.dll*.
- iii) Inherit the interface *ISoapInterface* defined in DLL.
- iv) Implement the methods named as *asynSubmitRequest()* and *synSubmitRequest()* defined in interface *ISoapInterface* and retrieve the values in the web methods from SSDG through SPGeneric connector method call.
- v) For submitting response for asynchronous request received from SSDG, SP application specific will call the *submitResponse()* method of SP Generic Connector using Class *SPImplSoap* defined in DLL.

Deployment of SP Application Specific Connector

Here we will see how the SP application specific connector which is a web service is deployed on IIS and what are the steps involved for the same. Sample Implementation for SP application specific is discussed above. This section explains the deployment of the SP application specific connector in a step by step manner along with the screen shots for every step.

4.1 Steps for deploying SP Application Specific Connector

- i) Create New Virtual directory to deploy the SP application specific connector as per *Figure 7*.

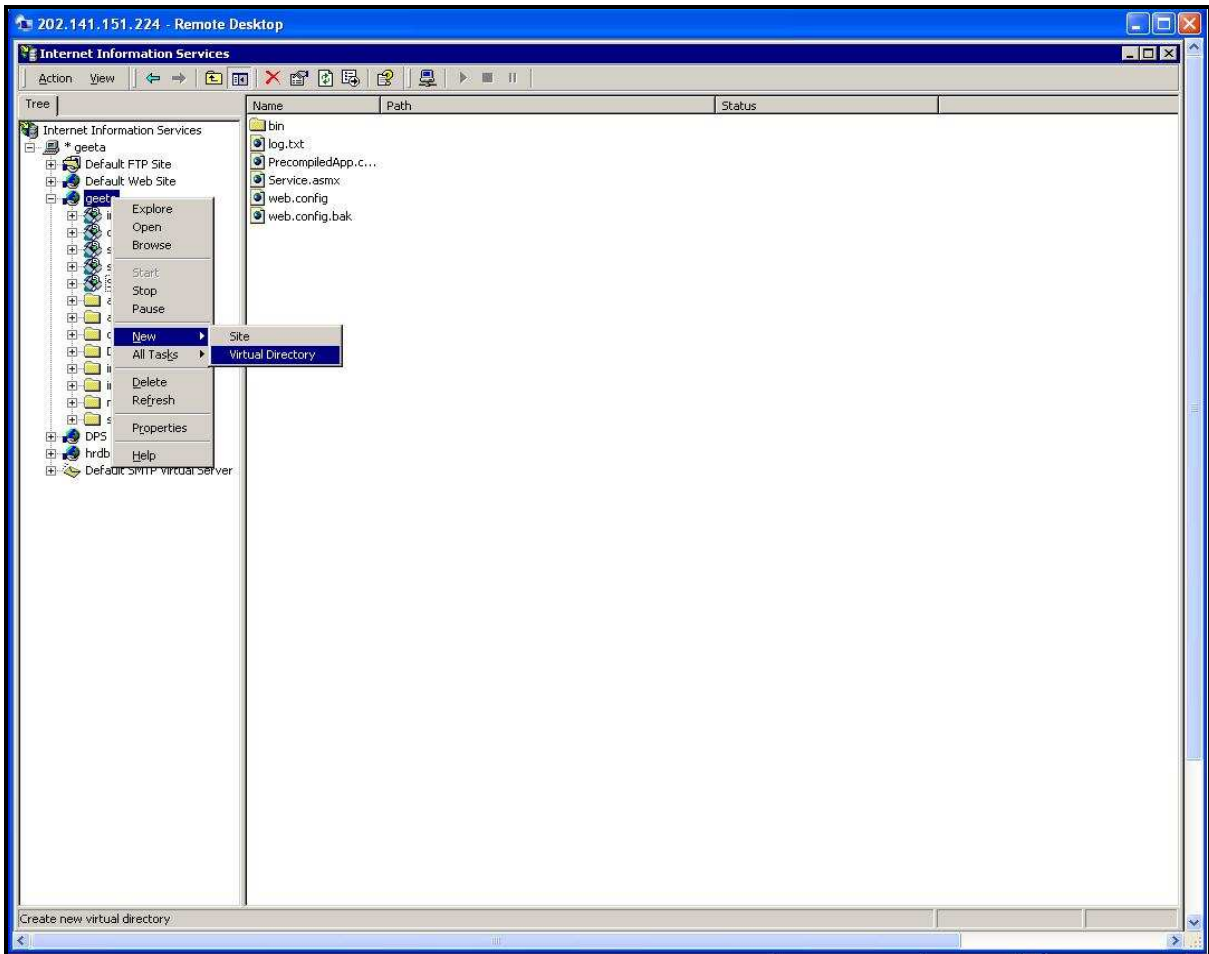


Figure 13 : Creating Virtual Directory for SP Application Specific connector in IIS

ii) Give name to the Virtual directory as *SPApplication* as shown in *Figure 8*.

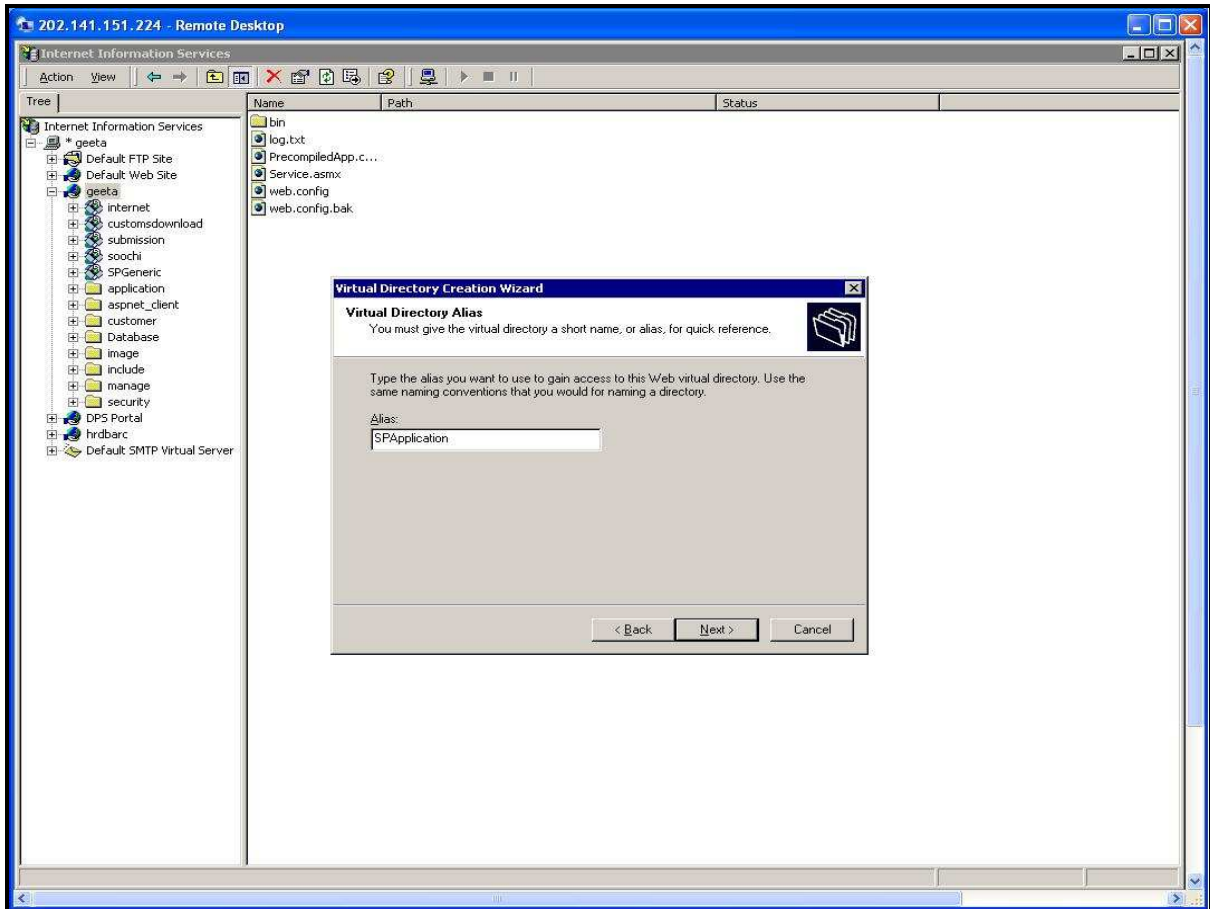


Figure 14 : Naming Virtual Directory

- iii) After creating the virtual directory named as *SPApplication* refer it to SP application connector on the location where it is actually copied. *Figure 9* shows the directory reference where *SPApplication* is copied.

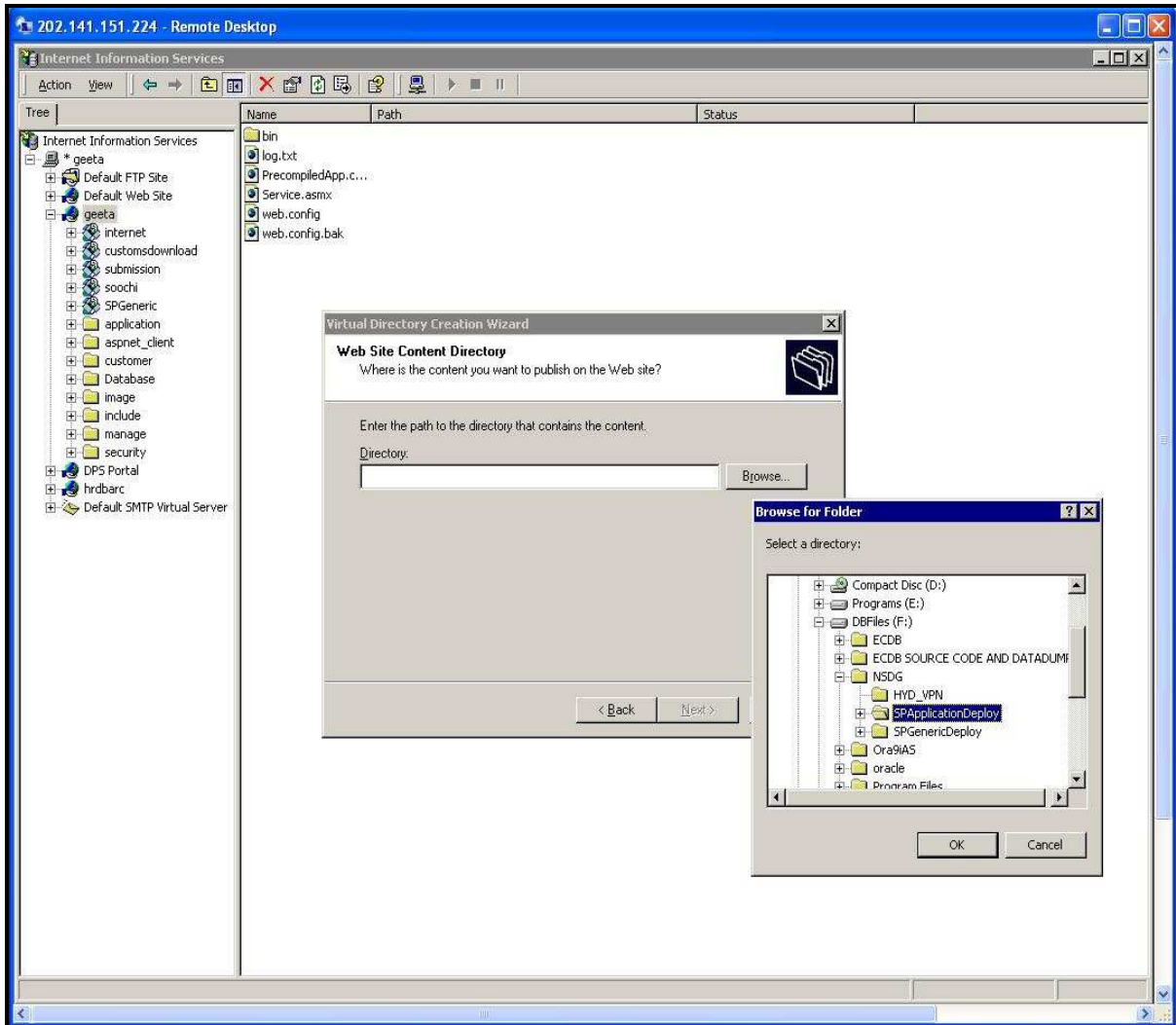


Figure 15 : Referring SP Application folder.

- iv) Once the SP application gets deployed make changes in security configuration as shown in the *Figure 10*.

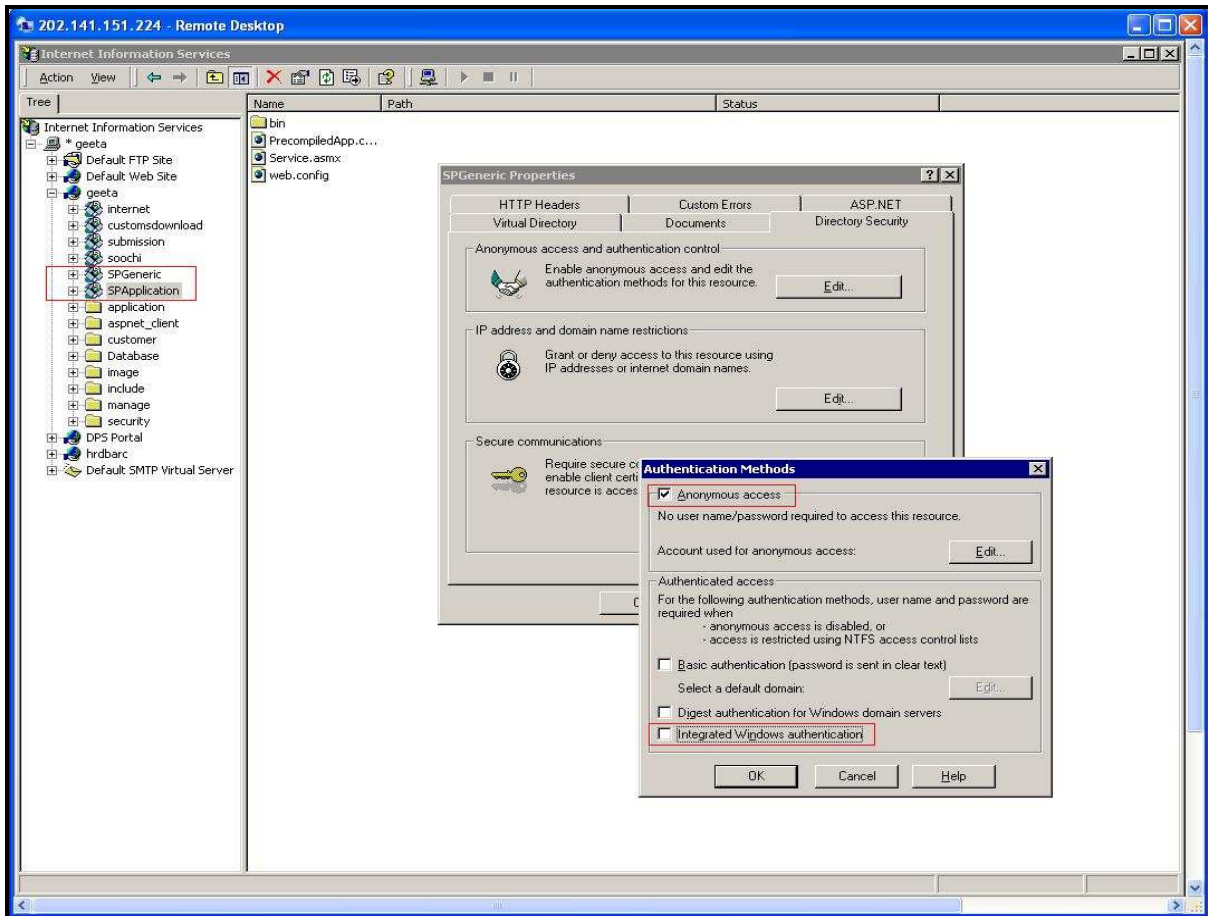


Figure 16 : Making changes for Authentication Setting in IIS

Note: Code used specific to body are prototype based, it can vary from application to application and implementation as per business logic of SP Application.

Chapter 5 Connectors in SSDG Architecture

This section elaborately discusses how the SAP and SP connectors interact with SSDG to exchange requests and relevant response. The classes and methods pertaining to the SAP and SP connectors are extensively explained along with the code. Also, there is a comprehensive discussion as to how the request travels to the SSDG through SAP side connectors finally reaching the SP application, and how response for the same is submitted to SAP through SSDG in case of synchronous and asynchronous requests.

A real world example is discussed in the following part where all scenarios pertaining to the connectors and their functioning are discussed in detail along with code snippets.

5.1 Real World Application-Trademark Request

In this cookbook we will discuss the different scenarios of connectors using Trademark Application.

Trademark Office provides details related to company registration. Request for company details is made by SAP for which the response is served in the form of all company details being provided to SAP by SP. The request given by SAP in body part of SAP generic connector comprises of “*Company Name*” against which TMR (trademark application) SP application gives all the company details under that name. The response contains details like application no., proprietor name etc.

The application specific data i.e. SAP request as per Trademark request schema will be set in the body field of the SAP generic connector. SAP will use this schema to send the request to Trademark department (SP). Similarly the SP response will be set in the body field of SP generic connector as per trademark response schema. SP will use this schema to send the response to SAP. Schema for communication between SAP and SP is named as Trademark Request schema and Trademark Response schema and will be known to both SAP and SP. The schema used for Trademark request and Trademark response is shown in *Snippet 1* and *Snippet 2*.

Note: Other properties for sending the request to SSDG will vary as per the different request scenarios as mentioned in DotNet manual and are required to be set accordingly.

Communication scenarios for Synchronous Request, Asynchronous Request and Inter-Gateway request submission are discussed. Also, how the response for these requests is

received is detailed. Various gateway operations like Poll Request, Delete Request and List Request are also explained. SAP in this case is portal from where user/citizen keys in the data which is converted as per the trademark request schema. Whereas the SP is an backend trademark application processing the request and generating the response.

5.2 Schema for Trademark Request and Response

Schema for Trademark Request

XML Generated for this schema will be set in the Body Element of SAP generic connector.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
version="1.0" targetNamespace="http://www.dipp.nic.in">
<xsd:element name="QueryTradeMarkRequest">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="queryString" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Snippet 1 : Schema for Trademark Request

Schema for Trademark Response

XML Generated for this schema will be set by SP in Body Element of SP generic connector.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="http://www.dipp.in" xmlns:tns=http://www.dipp.in
elementFormDefault="qualified" xmlns:xsd=http://www.w3.org/2001/XMLSchema version="1.0">
<xsd:element name="TradeMarkResponse">
<xsd:complexType>
<xsd:sequence>

<xsd:element name="RecordSet" type="tns:RecordSetType" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType name="RecordType">
<xsd:sequence>
<xsd:element name="TradeMarkName" type="xsd:string"/>
<xsd:element name="ApplicationNo" type="xsd:string"/>
<xsd:element name="ProprietorName" type="xsd:string"/>
<xsd:element name="ProprietorAddress" type="xsd:string"/>
<xsd:element name="Status" type="xsd:string"/>
<xsd:element name="Class" type="xsd:string"/>

<xsd:element name="ApplicationStatus" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RecordSetType">
<xsd:sequence>
<xsd:element name="Record" type="tns:RecordType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Snippet 2 : Schema for Trademark Response

5.3 Executing Synchronous Submit Request at SAP end

Recipe 1 Synchronous Submit Request Execution

SAP wants to send request to SP i.e. Trademark for which immediate response in the form of company registration details is desired from Trademark. Now as the SP i.e. Trademark is registered with SSDG in synchronous mode of communication, it will provide the response as soon as the request hits the Trademark application.

Solution:

The following steps are involved in executing the synchronous submit request. Each of these step are explained in detail along with the code snippets.

- i) *Figure 11* shows the synchronous request from SAP to Trademark application.
- ii) Generate code from Trademark request schema to set the company name against which information is required by SAP from Trademark application. *Snippet 3* shows the code generated from Trademark Request Schema. This generated code is used for setting the values of client specific content i.e. “Company Name” in this case to the Body field of the generic connector after converting into XML.
- iii) Define method in SAP application specific connector to convert the Body specific content into XML Element [] as shown in *Snippet 4*.
- iv) Implement SAP application specific connector as shown in *Snippet 5* using SAP generic connector with the classes provided in *Snippet 3* and *Snippet 4*. The classes should be referred in the project.
- v) application *Snippet 5* also shows how the SAP connector calls the *SubmitRequest()* method by passing the necessary parameters as listed in Table 18 **Input Parameters for Synchronous Submit Request** of the DotNet Connector Development Manual. The *SubmitRequest()* method is defined in SAP generic connector.

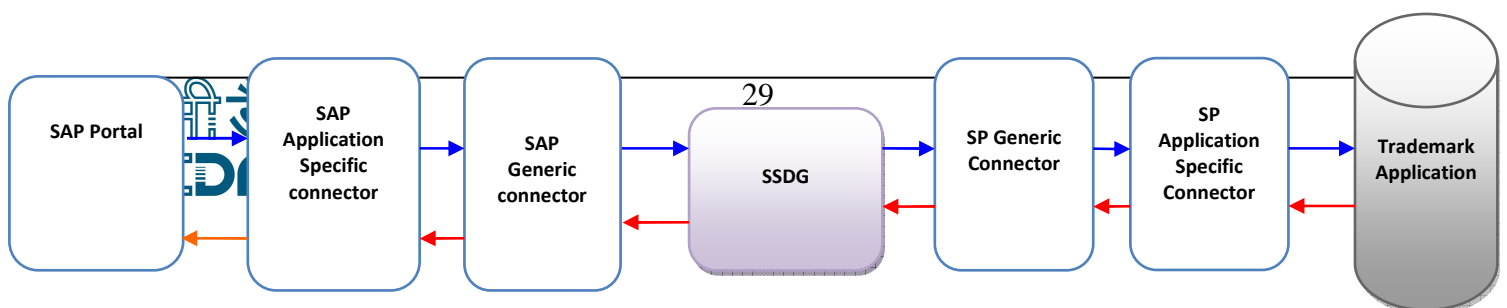


Figure 17 : SAP calling trademark application in synchronous mode

Code Snippets for Recipe 1

For generating code from xsd run the following query;

*Open Visual Studio tool command prompt -> xsd /classes /language:CS
Trademarkrequest.xsd*

```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;

// Summary description for Class1

10 [System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
11 [System.SerializableAttribute()]
12 [System.Diagnostics.DebuggerStepThroughAttribute()]
13 [System.ComponentModel.DesignerCategoryAttribute("code")]
14 [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true, Namespace =
    "http://www.dipp.nic.in")]
15 [System.Xml.Serialization.XmlRootAttribute(Namespace = "http://www.dipp.nic.in", IsNullable
    = false)]

16 public partial class QueryTradeMarkRequest
17 {
18     private string queryStringField;
```

```
19     public string queryString
20     {
21         get
22         {
23             return this.queryStringField;
24         }
25         set
26         {
27             this.queryStringField = value;
28         }
29     }
30 }
```

Snippet 3 : Snippet of code generated from Trademark Request Schema

Discussion:

- i) Code used in the snippet 3 is generated from schema given in *Snippet 1*
- ii) Code in line 22 sets the field “querystring” with the value received from SAP in the request body while submitting the request

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using System.Diagnostics.CodeAnalysis;
8 using System.Diagnostics;
9 using SAPConnector;
10 using System.Security.Cryptography.Xml;
11 using System.Security.Cryptography.X509Certificates;
12 using System.Configuration;
13 using System.Security.Cryptography.Pkcs;

14 public partial class ConvetToXML
15 {
16     //Passing QueryTradeMarkRequest generated from Schema as Object
17     public XmlElement[] createXmlBodyDocTrade ( QueryTradeMarkRequest obj )
18     {
19         XmlSerializer sr = new XmlSerializer ( typeof ( QueryTradeMarkRequest ) );
20         MemoryStream stream = new MemoryStream ( );
21         sr .Serialize ( stream , obj );
22         stream .Position = 0;
23         XmlDocument xd = new XmlDocument ( );
24         xd .Load ( stream );
25         XmlElement [ ] xa=new XmlElement[1];
26         xa [ 0 ] = xd .DocumentElement;
27         return xa;
28     }
29 }
```

Snippet 4 : Snippet to define method for XML conversion

Discussion:

- i) Snippet 4 is used to generate the `XmlElement []` for the request which will be sent by SAP to SSDG.
- ii) Code in line 16 declares method `createXmlBodyDocTrade()` which takes `QueryTrademarkRequest` object as parameter for converting it into XML.
- iii) Code in line 18 shows the `XMLSerializer` being called for this object type.

- iv) Code in line 20 and 21 puts the serialized object into temporary storage i.e. memory stream.
- v) Code in line 22 through 25 loads the data into *XmlDocument* and finally assigns it to *XMLElement []*.
- vi) Code in line 26 returns the *XMLElement []* for the converted *QueryTrademarkRequest* object.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;

8 public class SAPInformation
9 {
10     static void Main ( string [ ] args )
11     {
12         //Specific Client Information as per Trademark request schema
13         QueryTradeMarkRequest content = new QueryTradeMarkRequest ();
14         //Values can also be taken from Web form.
15         content.queryString = "AMWAY";

16         //Method call to create XMLElement[]for above provided request
17         ConverterToXML converter=new ConverterToXML();
18         XmlElement[] body = converter.createXmlBodyDocTrade (content);

19         //Generic Connector API Starts here
20         ConnectorInterface inter = new ConnectorInterface ();

21         //Optional Field set by Client
22         inter.TransactionId = "12345678912345678912345678912345";
23         inter.CorrelationID = "";
24         //Type of Service Requested i.e. Synchronous
25         inter.ResponseMode = ResponseMode.Synchronous;
26         //Valid End point Address of SP for synchronous request
27         inter.ClassValue = "http://127.0.0.1/22";
28         //Address of SSDG
29         inter.TargetEndPoint="http://localhost:3960/SPService/Service.asmx";
30         //Authentication Details recived after registration of SAP
31         inter.AuthMode = AuthMode.Clear;
32         inter.SenderID = "SAP130720081454";
33         inter.Password = "[B@13a87a0";

34         //Setting Converted XMLElement array SAP Specific content
35         inter.BodyContent = body;
```

```
//Calling of SSDG through Generic Connector API
```

```
26     ConnectorInterface.SubmitRequest(ref inter);
27     }
28     }
29 }
```

Snippet 5 : Snippet to Implement SAP Application Connector

Discussion:

- i) Code in lines 12 and 13 is used to define the input for *QueryTrademarkRequest* class by creating an object *content* of this class and assigning user input to the same.
- ii) Code in line 15 uses method *createXmlBodyDocTrade()* to convert the client specific content to XML object. Data set in line 13 is then converted to XML object by calling the method *createXmlBodyDocTrade()* using the *converter* object of *ConvertToXML* class and passing the object *content* to this method. The method returns the XML object after appropriate conversion
- iii) Next, an object name *inter* of *ConnectorInterface* class (defined in Generic Connector) is created to set properties of that class and call the synchronous *SubmitRequest()* method. The object creation code is given in line 16. The code from line no. 17 to 26 is used to set properties of the *ConnectorInterface* class which includes setting the *TransactionId*, *CorrelationID*, *ClassValue*, *ResponseMode*, *TargetEndPoint*, *AuthMod* , *SenderID*, *Password* and *BodyContent*.
- iv) Line 28 demonstrates how the synchronous *SubmitRequest()* is called using the object *inter* of *ConnectorInterface* class. A reference of the object *inter* is passed while invoking *SubmitRequest()* method.

5.4 Processing of Trademark Request at SP end sent in Synchronous Mode.

The request sent by SAP is retrieved by SP generic connector from SSDG and is sent to SP application specific connector by invoking the web method *synSubmitRequest()* of SP application connector. This method will finally call the service method (which will process the request and generate response) or implement the logic in *synSubmitRequest()* method for processing the data sent by SAP.

The steps given below discuss the request processing at SP end. Deployment steps for *SPGeneric* and *SPApplication* specific is already discussed in section 2.1 and section 3.1.

Recipe 2 Retrieve Request from SSDG in case of Synchronous Submit Request.

Trademark Application needs to retrieve the request sent by SAP as described in Recipe 1.

Solution:

- i) SP application specific connector implements the methods defined in the interface *IServiceSoap* provided in the form of **DLL**.
- ii) SP application specific connector implements *synSubmitRequest()* as a web method in a new Web Service.
- iii) *synSubmitRequest()* method is called by SP generic connector for submitting synchronous requests received from SSDG for targeted SP Application i.e. Trademark.

Note: *The developer needs to build SP application specific connector in the form of Web Service and will use the properties defined in the provided DLL. Description of these properties is given in Table 24 of Dot Net Connector Manual.*

5.4.1 Steps involved in the implementing the Solution

- i) Code generated from trademark response schema is mentioned in *Snippet 6*. For generating code from xsd run the following query;

Open Visual Studio tool command prompt -> xsd /classes /language:CS Trademarkresponse.xsd

- ii) *Snippet 7* shows the method that is defined in SP application specific connector for conversion and reversion of XML Element []. Conversion from XML Element[] is needed when the request from SAP comes in the form of XML and reversion is required by SP for sending back the response to SAP in XMLElement[].
- iii) *Snippet 8* gives the implementation of Web method *synSubmitRequest()* in SP application specific connector.
- iv) The Trademark request schema (*Snippet 3*) is used for retrieving the input and trademark response schema for setting the response (*Snippet 6*) from SP Application.

```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;

/// Summary description for TradeMarkResponse
/// </summary>

10 [System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
11 [System.SerializableAttribute()]
12 [System.Diagnostics.DebuggerStepThroughAttribute()]
13 [System.ComponentModel.DesignerCategoryAttribute("code")]
14 [System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true,
    Namespace="http://www.dipp.in")]
15 [System.Xml.Serialization.XmlRootAttribute(Namespace="http://www.dipp.in", IsNullable=false)]
```

```
16 public partial class TradeMarkResponse
17 {
18     private string statusFlagField;
19     private RecordType[] recordSetField;
20     /// <remarks/>
21     public string StatusFlag {
22         get {
23             return this.statusFlagField;
24         }
25         set {
26             this.statusFlagField = value;
27         }
28     }
29     /// <remarks/>
30     [System.Xml.Serialization.XmlArrayItemAttribute("Record", IsNullable=false)]
31     public RecordType[] RecordSet {
32         get {
33             return this.recordSetField;
34         }
35         set {
36             this.recordSetField = value;
37         }
38     }
39     /// <remarks/>
40     [System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
41     [System.SerializableAttribute()]
42     [System.Diagnostics.DebuggerStepThroughAttribute()]
43     [System.ComponentModel.DesignerCategoryAttribute("code")]
44     [System.Xml.Serialization.XmlTypeAttribute(Namespace = "http://www.dipp.in")]
45     public partial class RecordType
46     {
47         private string tradeMarkNameField;
48         private string applicationNoField;
49         private string proprietorNameField;
50         private string proprietorAddressField;
51     }
52     /// <remarks/>
```

```
49 public string TradeMarkName
50 {
51     get
52     {
53         return this.tradeMarkNameField;
54     }
55     set
56     {
57         this.tradeMarkNameField = value;
58     }
59 }
    /// <remarks/>

60 public string ApplicationNo
61 {
62     get
63     {
64         return this.applicationNoField;
65     }
66     set
67     {
68         this.applicationNoField = value;
69     }
70 }
    /// <remarks/>

71 public string ProprietorName
72 {
73     get
74     {
75         return this.proprietorNameField;
76     }
77     set
78     {
79         this.proprietorNameField = value;
80     }
81 }
    /// <remarks/>

82 public string ProprietorAddress
83 {
```

```
84  get
85  {
86      return this.proprietorAddressField;
87  }
88  set
89  {
90      this.proprietorAddressField = value;
91  }
92  }
```

Snippet 6 : Snippet of code generated form Trademark Response Schema

Discussion:

- i) *TradeMarkResponse* class in snippet 6 will be used by SP to send the response to SAP.

```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;
10 using System.Xml;
11 using System.Xml.Serialization;
12 using System.IO;

    /// Summary description for SPApplication
    /// </summary>

13 public class SPApplication
14 {
15     public SPApplication()
16     {
17         // TODO: Add constructor logic here
18     }

    ///This class will be used to retrieve the values sent by SAP in Request Object sent by
    ///SAP

18     public QueryTradeMarkRequest XMLBody(XmlElement body)
19     {
20         QueryTradeMarkRequest request = new QueryTradeMarkRequest();
21         string bodyContent = body.OuterXml;
22         StringReader read = new StringReader(bodyContent);
23         XmlSerializer sr = new XmlSerializer(request.GetType());
24         XmlReader reader = new XmlTextReader(read);
25         request = (QueryTradeMarkRequest)sr.Deserialize(reader);
26         return request;
27     }

    ///This class will be used to convert the result in XMLElement[] as per trademark
    ///response schema

28     public XmlElement createXmlBodyDocTrade(TradeMarkResponse obj)
29     {
30         XmlSerializer sr = new XmlSerializer(typeof(TradeMarkResponse));
31         MemoryStream stream = new MemoryStream();
```

```
32     sr.Serialize(stream, obj);
33     stream.Position = 0;
34     XmlDocument xd = new XmlDocument();
35     xd.Load(stream);
36     XmlElement xe = xd.DocumentElement;
37     return xe;
38 }
39 }
```

Snippet 7 : Snippet defining de-conversion method of XML element and XML converter for XML element

Discussion:

- i) Code in lines 18 through 24 defines method *XMLBody()* to convert the client specific object named as *request* as per *QueryTradeMarkRequest*.
- ii) Code in lines 28 through 38 defines method *createXmlBodyDocTrade()* to convert the result returned by SP into XML.

```
1 using System;
2 using System.Web;
3 using System.Web.Services;
4 using System.Web.Services.Protocols;
5 using System .Data .Sql;
6 using System .Data .SqlClient;
7 using System .Xml;
8 using System .Xml .Serialization;
9 using System .IO;

10 [WebService(Namespace = "connector.cdac.com")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

12 public class Service :System.Web.Services.WebService,IServiceSoap
13 {
14     public Service ()
15     {
16     }

17     [WebMethod]
18     public void synSubmitRequest ( ref SPInterface interfaceSP )
19     {
20         //Retrieving the Values recieved from SAP through SSDG
21         XmlElement[] bodyArr = new XmlElement[1];
22         SPApplication obj = new SPApplication();
23         bodyArr = interfaceSP .Body;
24         QueryTradeMarkRequest requestObj = obj.XMLBody(bodyArr[0]);

25         //showed if else for demonstartion purpose
26         if (requestObj.queryString == "Amway")
27         {
28             SPResponse response = new SPResponse();
29             TradeMarkResponse responseRecieved = new TradeMarkResponse();
30             responseRecieved = response.resultReturn();
31             bodyArr[0] = response. createXmlBodyDocTrade(responseRecieved);
32             interfaceSP.Body[0] = bodyArr[0];
33             interfaceSP.SubmissionStatus = "response";
34         }
35         else
36         {
37             //If the content recieved in the body is incorrect.Also set the
38             //body as per schema of Error giving the reason for the error.
39             setProperties .SubmissionStatus = "error";
40         }
41     }
42 }
```

```
37 }
38 }
    /// <summary>
    /// SP Application business logic class
    /// </summary>
39 public class SPResponse
40 {
41     public TradeMarkResponse resultReturn()
42     {
43         TradeMarkResponse responseObj = new TradeMarkResponse();
44         RecordType[] recordArray = new RecordType[1]
45         recordArray[0] = new RecordType();
46         recordArray[0].ApplicationNo = "123456";
47         recordArray[0].ProprietorName = "UK";
48         recordArray[0].ProprietorAddress = "MUMBAI";
49         recordArray[0].TradeMarkName = "Adidas";
50         responseObj.RecordSet = recordArray;
51         return responseObj;
52     }
53 }
```

Snippet 8 : Snippet for implementing synSubmitRequest() method

Discussion:

- i) Code in line 12 shows the web service class *Service* extending the interface *IServiceSoap*.
- ii) Code in lines 17 and 18 defines the implementation of web method *synSubmitRequest()* which takes the class *SPInterface objects* as parameter.
- iii) Code in line 22 retrieves the request sent by SAP in the form of XMLElement [].
- iv) Code in line 23 converts the request received in XML form to an object named *requestObj* of a class *QueryTradeMarkRequest*.

5.5 Retrieving the result sent by SP Application to SAP

Recipe 3 Response from Trademark application

SAP will retrieve the values received from trademark application as per trademark response schema described in *Snippet 6* against the trademark request sent by SAP.

Solution:

- i) SAP will use the *ConnectorInterface* class object passed in *SubmitRequest()* of generic connector to retrieve the values sent by Trademark application.
- ii) *Snippet 9* shows how the response from the Trademark application.
- iii) If the response received is an error, the error details from the accompanying error list are read along with the respective error codes and necessary rectification is made in the SAP application connector.
- iv) If response received is the desired result, then the response is read into the fields of *Response* class generated from trademark response schema.
- v) Before setting the response fields in *Response* class, SAP deserializes the response which is in the form of XML Element[]set in the Body field.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;
8
9 public class SAPInformation
10 {
11     static void Main ( string [ ] args )
12     {
13         //Method Call initiated by SAP
14         ConnectorInterface.SubmitRequest(ref inter);
```

```
14      /Receiving Result
14      string status= inter.SubmissionStatus;

15      //if status is response
16      string corrID=inter.CorrelationID;
16      string classID = inter.ClassValue;

17      //If Status is error
17      string errorCode = inter .ErrorCode;
18      string[] errorList=new string[1];
19      XmlElement[] bodyArr = new XmlElement[1];
20      errorList = inter .ErrorList;
21      string error=null;
22      if ( errorList != null )
23      {
24          error = errorList [ 0 ];
25      }
26      if ( inter .BodyContent != null )
27      {
28          bodyArr = inter .BodyContent;
29      }
30  }
31 }
```

Snippet 9 : Snippet for retrieving Response received from Trademark Application

Discussion:

- i) Code in lines 13 shows the calling of **SubmitRequest()** method by SAP. The result be returned in the parameter **ref** which is passed while invoking the **SubmitRequest()**.
- ii) Code in lines 14 through 18 retrieves properties Submission status, Correlation ID and Class Value which are set in the returned **ref** object.
- iii) Based on submission status, if the status is an error, then code in lines 17 to 25 retrieves the error.
- iv) Based on submission status if status is response, code in lines 26 to 28 retrieves the response sent by SAP in **XMLElement[]**.

5.6 Executing Asynchronous Submit Request

In this section we will discuss the Trademark scenario in asynchronous communication mode.

Recipe 4 Asynchronous Submit Request Execution

SAP needs to send request to SP Application i.e. trademark for which response from trademark is desired related to company registration details. The SP is registered with SSDG in asynchronous mode of communication so it provides the response to SSDG after processing at some later point of time.

Solution:

Figure 12 demonstrates the flow of message from SAP to SP Application in asynchronous mode

- i) Use Trademark request Schema generated in *Snippet 3* to set the company name against which SAP needs information from Trademark. This class is used for setting the values in Body field of the SAP generic connector.
- ii) Snippet 10 shows how SAP generic connector uses the already generated classes in *Snippet 3* and *Snippet 4* for Trademark request and conversion of class in XMLElement [].
- iii) Call method **SubmitRequest()** defined in SAP generic connector and pass the necessary parameters listed *Table 27* as **Input Parameters for Asynchronous Submit Request** in the DotNet Connector Development Manual.

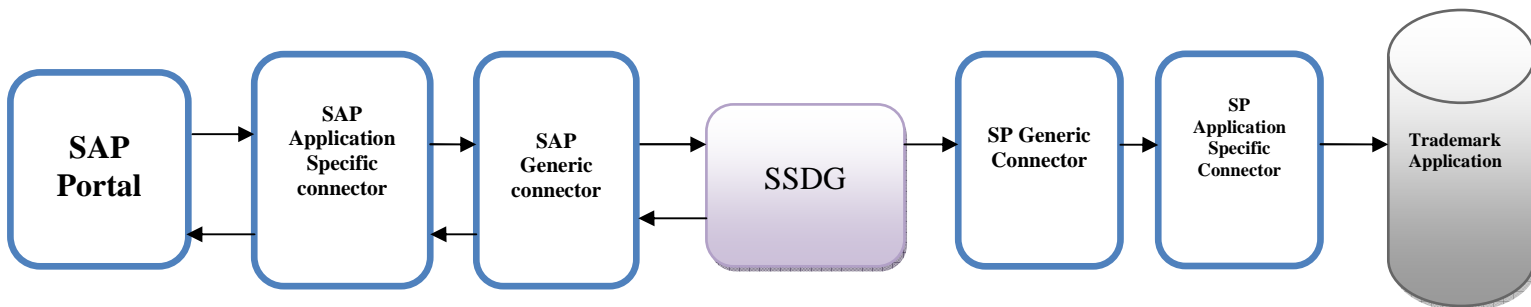


Figure 18 : SAP asynchronously submits request to trademark application through SSDG.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;

8 public class SAPInformation
9 {
10     static void Main ( string [ ] args )
11     {
12         //Specific Client Information as per trademark schema
13         QueryTradeMarkRequest content = new QueryTradeMarkRequest ();

14         //Values can also be taken from Web form.
15         content.queryString = "AMWAY";

16         //Method call to create XElement[]for above provided request
17         ConvetToXML converter=new ConvetToXML();
18         XElement[] body = converter.createXmlBodyDocTrade (content);

19         //Generic Connector API Starts here
20         ConnectorInterface inter = new ConnectorInterface ();

21         //Optional Field set by Client
22         inter.TransactionId = "12345678912345678912345678912345";
23         inter.CorrelationID = "";

24         //Type of Service Requested i.e. Synchronous
25         inter.ResponseMode = ResponseMode.Asynchronous;

26         //Valid End point Address of SP for synchronous request
27         inter.ClassValue = "http://127.0.0.1/22";

28         //Address of SSDG
29         inter.TargetEndPoint="http://localhost:3960/SPService/Service.asmx";

30         //Authentication Details recived after registration of SAP
31         inter.AuthMode = AuthMode.Clear;
32         inter.SenderID = "SAP130720081454";
33         inter.Password = "[B@13a87a0";

34         //Setting Converted XMLElement array SAP Specific content.
35         inter.BodyContent = body;
```

```
//Calling of SSDG through Generic Connector API  
28 ConnectorInterface.SubmitRequest(ref inter);  
29 }  
30 }
```

Snippet 10 : Snippet for using Generic Connector to submit Asynchronous Request to SSDG

Discussion:

- i) Code in lines 12 and 13 is used to define the input for *QueryTrademarkRequest* class.
- ii) Code in line 14 creates an object of *ConvetoXML* class named as *converter* which is used to call the method *createXmlBodyDocTrade()* to convert the client specific content to XML object.
- iii) Data set in line 13 is then converted to XML object by calling the method *createXmlBodyDocTrade()* using the *converter* object and passing the object *content* to this method. The method returns the XML object after appropriate conversion.
- iv) Next, in line number 16, an object name *inter* of *ConnectorInterface* class (defined in SAP generic connector) is created to set properties of that class and call the asynchronous *SubmitRequest()* method. The code from line no. 17 to 26 is used to set properties of the *ConnectorInterface* class which includes setting the *TransactionId*, *CorrelationID*, *ClassValue*, *ResponseMode* *TargetEndPoint*, *AuthMod*, *SenderID*, *Password* and *BodyContent*.
- v) Line 28 demonstrates how the asynchronous *SubmitRequest()* method is called using the object *inter* of *ConnectorInterface* class. A reference of the object *inter* is passed while invoking the asynchronous *SubmitRequest()* method.

5.7 Retrieving Acknowledgement from SSDG

Recipe 5 Retrieve the Acknowledgement from SSDG

The asynchronous request sent by SAP is routed to Trademark application through SSDG. Now, as the request is asynchronous, the response will not be returned instantaneously to SAP. Meanwhile when the request is submitted to SSDG, the SSDG validates the correctness of request message and if the request is found to be valid, SSDG issues submit acknowledgement to SAP. This recipe shows how this acknowledgement is retrieved by SAP which is submitted by SSDG.

Solution:

As the mode of communication is asynchronous the response from trademark application will be submitted to SSDG. SAP will retrieve the values received in the submitted acknowledgement and use the retrieved values to poll SSDG for retrieving the response submitted by SP to SSDG.

- i) *Snippet 11* shows the retrieval of values received in Submit Acknowledgement from SSDG.
- ii) SAP will use the *connector Interface class* object passed in Generic connector method to read the values sent in acknowledgement from SSDG.
- iii) If there is error, read the error details from errorlist along with error code and make correction in SAP application specific as per error details.
- iv) If acknowledgement is there then read all the values necessary for *DocumentPoll()*.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;
```

```
8 public class SAPInformation
9 {
10     static void Main ( string [ ] args )
11     {
12         //Method Call initiated by SAP
13         ConnectorInterface.SubmitRequest(ref inter);
14
15         //Receiving Result
16         string status= inter.SubmissionStatus;
17
18         //if status is acknowledgement
19         string corrID=inter.CorrelationID;
20         string classID = inter.ClassValue;
21         string endPoint = inter .TargetEndPoint;
22
23         //if Status is error
24         string errorCode = inter .ErrorCode;
25         string[] errorList=new string[1];
26         errorList = inter .ErrorList;
27         string error=null;
28         if ( errorList != null )
29         {
30             error = errorList [ 0 ];
31         }
32     }
33 }
```

Snippet 11 : Snippet for Retrieving Acknowledgement from SSDG

Discussion:

- i) Code in line 12 shows the call of SubmitRequest method by SAP. As the parameter is passed by *ref*, the acknowledgment will be obtained in same object which is sent as parameter.
- ii) Code in lines 13 through 16 retrieves the properties Submission status, Correlation ID and Class Value and endpoint from the returned object.
- iii) Based on submission status if status is error, code in lines 17 to 23 retrieves the error list for the error generated while sending request form SAP to SSDG.

5.8 Processing of Trademark Request at SP end sent in Asynchronous Mode

In the above procedures request is sent to SSDG through SAP Generic connector which will be forwarded to SP Generic connector through SSDG. When request from SAP is received by SSDG, SSDG will determine if the request is valid. If valid, SSDG sends Acknowledgement to SAP. The request is then retrieved by SP generic connector from SSDG and is sent to SP application specific connector by invoking the web method *asynSubmitRequest()* of SP application connector. This method will finally call the service method (which will process the request and generate response) or implement the logic in *asynSubmitRequest()* method for processing the data sent by SAP

Recipe 6 Retrieve Request from SSDG in case of Asynchronous Submit Request

Trademark Application needs to retrieve the request sent by SAP as described in Recipe 6.

Solution:

- i) SP application specific connector implements the methods defined in the interface *IServiceSoap* provided in the form of **DLL**.
- ii) SP Application specific connector implements *asynSubmitRequest()* as a web method in a Web Service.
- iii) *asynSubmitRequest()* method is called by SP generic connector for submitting asynchronous requests received from SSDG for targeted SP Application i.e. Trademark.

Note : The developer needs to build SP application specific connector in the form of Web Service and will use the properties defined in the provided DLL. Description of these properties is given in Table 24 of Dot Net Connector Manual.

- iv) Use already generated classes for Trademark response schema and XML serialization and Deserialization from *Snippet 6* and *Snippet 7* respectively.
- v) *Snippet 12* shows the implementation of Web method *asynSubmitRequest()* and processing of input received from SAP through SSDG.

```
1 using System;
2 using System.Web;
3 using System.Web.Services;
4 using System.Web.Services.Protocols;
5 using System .Data .Sql;
6 using System .Data .SqlClient;
7 using System .Xml;
8 using System .Xml .Serialization;
9 using System .IO;
10 [WebService(Namespace = "connector.cdac.com")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

12 public class Service :System.Web.Services.WebService,IServiceSoap
13 {
14     public Service ()
15     {
16     }

17     [WebMethod]
18     public void asynSubmitRequest ( ref SPInterface interfaceSP )
19     {
20         //Retrieving the Values recieved from SAP through SSDG
21         XmlElement[] bodyArr = new XmlElement[1];
22         SPApplication obj = new SPApplication();
23         bodyArr = interfaceSP .Body;

24         QueryTradeMarkRequest requestObj = obj.XMLBody(bodyArr[0]);
25         string CorrelationID = interfaceSP.CorrelationID;
26         string classValue = interfaceSP.ClassValue;
27         string endPointSSDG = interfaceSP.TargetEndPoint;

28         //Store Values in the Database
29     }
30 }
```

Snippet 12 : Snippet for implementing asynSubmitRequest() method

Discussion:

- i) Code in line 12 shows the web service class *Service* extending the interface *IServiceSoap*.

- ii) Code in lines 17 and 18 shows the web method **asynSubmitRequest()** implementation taking class **SPInterface** object as parameter.
- iii) Code in line 22 retrieves the request sent by SAP in Body in the form of XMLElement [].
- iv) Code in line 23 converts the request received in XML form to object of class **QueryTradeMarkRequest**.
- v) Code in lines 24 through 26 retrieves the other properties sent by SSDG for successful submission of response by SP in asynchronous mode.
- vi) This data is stored by SP in a database and will be later processed for generating the response. **SubmitResponse()** method of SP generic connector will then be invoked for submitting the response.

5.9 Submitting the Response from SP Application to SSDG

In asynchronous request scenario, the SP application after processing the data at some later point from the time of submission submits result to SSDG. SSDG stores the result and forwards it to SAP when SAP polls SSDG for result.

Recipe 7 Submit Response from SP Application to SSDG for Asynchronous Request

As the application sent by SAP is in asynchronous mode, SP after processing the request sends the result to SSDG. SAP need to Poll SSDG for receiving the result submitted by SP Application.

Solution:

SP submits the response in the Body field of SPGeneric connector as an XMLElement [] to SSDG along with the values received in the request from SSDG.

- i) *Figure 13* demonstrates the flow for submitting the response from SP application and *Snippet 13* shows the response submission to SSDG from SP Application.
- ii) SP will use the DLL referenced in implementing SP application specific connector and set the property values as described in *Table 25* of dot net manual for the method `submitResponse()`.
- iii) SP will call SP generic connector method named as `submitResponse()` using class `SPImplSoap` defined in DLL.

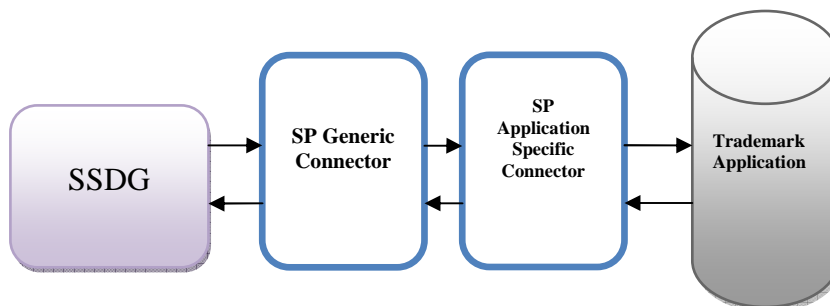


Figure 19 : Trademark application response to SSDG for Asynchronous Request

```

1 using System;
2 using System.Web;
3 using System.Web.Services;
4 using System.Web.Services.Protocols;
5 using System .Data .Sql;
6 using System .Data .SqlClient;
7 using System .Xml;
8 using System .Xml .Serialization;
9 using System .IO;

10 [WebService(Namespace = "connector.cdac.com")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
12 public class Service : System.Web.Services.WebService, IServiceSoap
13 {
14     public Service ()
15     {
16         //Uncomment the following line if using designed components
17         //InitializeComponent();
18     }

19     public void SPResponse ( ref SPInterface interfaceSP )
20     {
21         XmlElement[] bodyArr = new XmlElement[1];
22         interfaceSP.CorrelationID="Value stored in DB from asynchronous request";
23         interfaceSP.ClassValue=" Value stored in DB from asynchronous request ";
24         interfaceSP.TargetEndPoint="Address of SSDG";
25         TradeMarkResponse responseRecieved = new TradeMarkResponse();

26         //Shown here for demonstartion purpose,else need to retrieve response from DB based
27         //upon input recieved and saved in asynchronous communication.

28         SPResponse response = new SPResponse();
29         responseRecieved = response.resultReturn();
30         bodyArr[0] = response.createXmlBodyDocTrade(responseRecieved);
31         interfaceSP.Body[0] = bodyArr[0];
32         interfaceSP.SubmissionStatus = "response";
33         SPSOAP callResponse = new SPSOAP ("path of Generic Connector");
34         callResponse .submitResponse ( ref interfaceSP );
35     }

36     /// SP Application business logic class
37     /// </summary>

```

```
32 public class SPResponse
33 {
34     public TradeMarkResponse resultReturn()
35     {
36         TradeMarkResponse responseObj = new TradeMarkResponse();
37         RecordType[] recordArray = new RecordType[1]
38         recordArray[0] = new RecordType();
39         recordArray[0].ApplicationNo = "123456";
40         recordArray[0].ProprietorName = "UK";
41         recordArray[0].ProprietorAddress = "MUMBAI";
42         recordArray[0].TradeMarkName = "Amway";
43         rresponseObj.RecordSet = recordArray;
44         return responseObj;
45     }
46 }
47 }
```

Snippet 13 : Snippet for Submitting Response to SSDG

Discussion:

- i) Code in line 17 defines the *SPResponse()* method for submitting the response from SP.
- ii) Code in lines 20 through 22 shows the retrieval of stored values received from asynchronous request and setting them in the field of *SPInterface* which will be sent to *submitResponse()* method of SP generic connector for submission to SSDG.
- iii) Code in Line 24 shows the response setting in specific response object.
- iv) Code in Line 26 creates the XMLElement [] for the response which is to be sent.
- v) Code in Line 27 sets the response in the property field of *SPInterface* class that will be passed as parameter to *submitResponse()* method of SP generic connector.
- vi) Code in line 29 creates the object for calling the *SPGeneric* method and parameter for this method will be path of SP generic connector.
- vii) Code in line 30 calls the *submitResponse()* method.
- viii) Code in lines 35 through 44 shows the response sent by SP, i.e. response for the request submitted by SAP through SSDG in asynchronous mode.

5.10 Sending the Acknowledgement back from SSDG to SP

Recipe 8 Submit Acknowledgement from SSDG to SP

When response from SP application is submitted to SSDG , SSDG validates the message and if the message received is valid, an acknowledgement will be issued to SP Application via SP generic connector.

Solution:

SP will retrieve the values received in submit acknowledgement to verify the correct delivery of message to SSDG.

- i) SP application will retrieve the values sent in acknowledgement from SSDG. The code for retrieving acknowledgement is given in *Snippet 14*.
- ii) If the values retrieved indicate an error status, the error list and error code are retrieved, the application specific code is rectified and the response is resent to SSDG.

```
1 using System;
2 using System.Web;
3 using System.Web.Services;
4 using System.Web.Services.Protocols;
5 using System .Data .Sql;
6 using System .Data .SqlClient;
7 using System .Xml;
8 using System .Xml .Serialization;
9 using System .IO;

10 [WebService(Namespace = "connector.cdac.com")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
12 public class Service : System.Web.Services.WebService, IServiceSoap
13 {
14     public Service ()
15     {
16         //Uncomment the following line if using designed components
17         //InitializeComponent();
18     }
19 }
```

```
17 public void submitResponse ( ref SPInterface interfaceSP )
18 {
19     SPSImplSoapSoap callResponse = new SPSImplSoapSoap ( "path of
        Generic Connector Web Service" );

        //Method Call Initiated by SP Application i.e trademark
20     callResponse .submitResponse ( ref interfaceSP );

        //Receiving Result
21     string status= inter.SubmissionStatus

        //if status is acknowledgement
22     string corrID=inter.CorrelationID;
23     string classID = inter.ClassValue;
24     string endPoint = inter .TargetEndPoint;

        //If Status is error
25     string errorCode = inter .ErrorCode;
26     string[] errorList=new string[1];
27     errorList = inter .ErrorList;
28     string error=null;
29     if ( errorList != null )
30     {
31         error = errorList [ 0 ];
32     }
33 }
34 }
```

Snippet 14 : Snippet for retrieving acknowledgement against submitResponse () from SP

Discussion:

- i) Code in lines 19 and 20 shows the call of **submitResponse()** method of SP Generic connector by SP application specific connector and as the parameter is passed by **ref** the acknowledgement will be obtained in same object.
- ii) Code in lines 21 through 24 receives the properties Submission status, Correlation ID, Class Value and Target End Point.
- iii) Based on submission status, if the status is error, code from lines 25 to 31 retrieves the error that occurred while sending the response from SP Application to SSDG.

5.11 Polling the SSDG to retrieve the Result.

Recipe 9 Poll the SSDG to retrieve the result submitted by SP Application

SAP will retrieve the values received from Trademark application as per trademark response schema mentioned in *Snippet 6*.

Solution:

- i) *Figure 14* gives the flow for polling the SSDG.
- ii) *Snippet 15* shows the polling by SAP application.
- iii) SAP submits the request through asynchronous mode and gets the submit acknowledgement from SSDG. SAP will use the details given in submit acknowledgement to Poll SSDG to receive the response submitted by SP application .
- iv) On polling the SSDG, if SAP receives acknowledgement, it indicates that the response has not yet been submitted by SP Application to SSDG whereas if SAP receives an error, SAP retrieves the error details from error list along with error code and rectifies Trademark request values with respect to SAP application connector.
- v) If SAP receives a response on polling SSDG, SAP reads the response class **TradeMarkResponse** files. The class **TradeMarkResponse** generated from trademark response schema.
- viii) For retrieving all fields, SAP first needs to deserialize the XML Element[] sent by SP in the Body field of SAP generic connector.

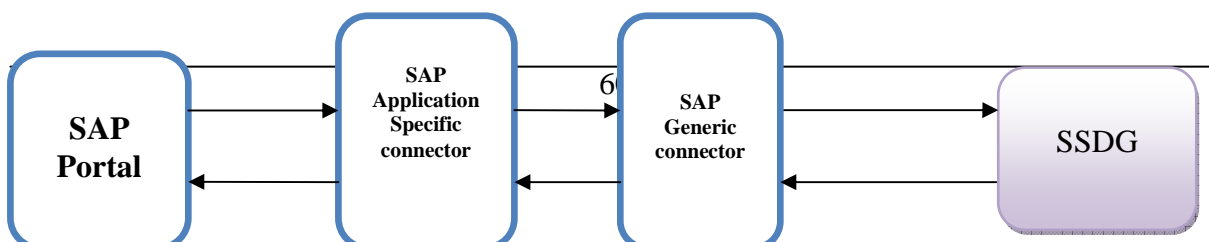


Figure 20 : SAP Polling SSDG for response

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;

8 namespace SAPClient
9 {
10 public class SAPInformation
11 {
12     static void Main ( string [ ] args )
13     {
14         //Generic Connector API Starts here
15         ConnectorInterface inter = new ConnectorInterface ();
16         inter.TransactionId = "12345678912345678912345678912345";

17         //or
18         //Will give response as it is optional field
19         inter.TransactionId = "";

20         //It should contain the value as received in Asynchronous
21         //SubmitRequest
22         inter.CorrelationID = "34E17E40952F4EAEB37E7B1FD7BA67EF";

23         //End Point URL for the service which is used in
24         //Asynchronous SubmitRequest
25         inter.ClassValue = "http://127.0.0.1/22";
```

```
    //End Point Url of the Gateway where DocumentPoll can be
    done
19     inter.TargetEndPoint="http://localhost/SPService/Service.asmx";
20     inter.DocumentPoll ( ref inter )

    //Receiving Result
21     string status= inter.SubmissionStatus;

    //if status is acknowledgement,it means response is still awaited from SSDG.
22     string corrID=inter.CorrelationID

    //Target End point for the Gateway
23     string responseEndPoint=inter.TargetEndPoint;
24     string classId = inter .ClassValue

    //If Status is error
25     string errorCode = inter .ErrorCode;
26     string[] errorList=new string[1];
27     errorList = inter .ErrorList;
28     string error=null;
29     if ( errorList != null )
30     {
31         error = errorList [ 0 ];
32     }

    //Can be retrived for Business Error and response.
33     if ( inter.BodyContent != null )
34     {
35         element = inter .BodyContent
36     }
37     }
38     }
39 }
```

Snippet 15 : Snippet for Poll Operation on SSDG

Discussion:

- i) Code in lines 15 to 19 shows the setting of generic connector properties received in acknowledgement when asynchronous submit request is submitted on SSDG.

- ii) Code in line 20 shows the calling of *DocumentPoll()* method by SAP application specific connector.
- iii) Code in lines 21 through 24 shows the retrieval of response sent by SSDG.
- iv) Based on submission status, if status is error, code in line 25 to 32 retrieves the error list
- v) Based on submission status, if status is response, code in lines 33 to 37 retrieves the response sent by SP in XMLElement[].

5.12 Deleting Request submitted to SSDG

SAP can request the deletion of requests for which SAP has already received the response through document poll.

Recipe 10 Executing Delete Request

Requests for which response has been received may be deleted using the *DocumentDelete()* operation.

Solution:

Call method *DocumentDelete()* defined in SAP generic connector and pass the necessary parameters listed in *Table 10 as Document Delete Input Parameters in* the DotNet Connector Development Manual.

- i) *Figure 15* demonstrates the flow for sending the delete request to SSDG.
- ii) If there is response the record is successfully deleted from the SSDG.
- iii) If there is error received from gateway, read error details in error specific properties and corrects the document for resending document to gateway for deletion request.
- iv) *Snippet 16* shows the calling of *DocumentDelete()* method on SAP generic connector.

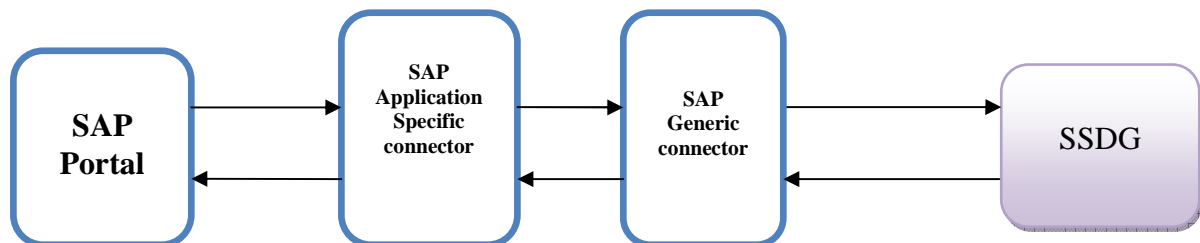


Figure 21 : SAP calling delete request for previously submitted requests.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;
8 namespace SAPClient
9 {
10  public class SAPInformation
11  {
12      static void Main ( string [ ] args )
13      {
14          //Generic Connector API Starts here
15          ConnectorInterface inter = new ConnectorInterface ( );
16          inter.TransactionId = "12345678912345678912345678912345";
17          //or
18          //Will give response as it is optional field
19          inter.TransactionId = "";
20
21          //It should contain the value as received in Asynchronous SubmitRequest
22          inter.CorrelationID = "34E17E40952F4EAEB37E7B1FD7BA67EF";
23
24          //End Point URL for the service which is used in Asynchronous SubmitRequest
25          inter.ClassValue = "http://127.0.0.1/22";
26
27          //End Point Url of the Gateway where DocumentPoll can be done
28          inter.TargetEndPoint = "http://localhost/SPService/Service.aspx";
29          inter.DocumentDelete ( ref inter );
30
31          //Receiving Result
32          string status= inter.SubmissionStatus;
33          //if status is response
34          string corrID=inter.CorrelationID;
35          //Target End point for the Gateway
36          string responseEndPoint=inter.TargetEndPoint;
37          string classId = inter .ClassValue;
38          //If Status is error
39          string errorCode = inter .ErrorCode;
40          string[] errorList=new string[1];
41          errorList = inter .ErrorList;
42          string error=null
```

```
29     if ( errorList != null )
30     {
31         error = errorList [ 0 ];
32     }
33
34     }
35 }
36 }
```

Snippet 16 : Snippet for Delete Operation on SSDG

Discussion:

- i) Code in lines 15 to 19 shows the setting of generic connector properties as received in document poll response.
- ii) Code in line 20 shows the calling of **DocumentDelete()** method by SAP application specific connector.
- iii) Code in lines 21 through 24 shows the retrieval of response sent by SSDG which indicates the successful deletion of the request from SSDG.
- iv) Based on submission status, if status is error, code in lines 25 to 32 retrieves the error list sent by SSDG.

5.13 Listing Requests submitted to SSDG.

Recipe 11 Executing List Request

For enquiring about the previous requests which were sent to SP during a specific time interval, SAP uses List Request operation by specifying the schema as specified in *Appendix D* of Dot Net Connector Manual which will produce a list of requests that were sent to SP and also the status of all such requests

Solution:

Call method *DocumentList()* defined in SAP generic connector and pass the necessary parameters listed in *Table 12, 14 and 16* as *Document List Input Parameters* in the DotNet Connector Development Manual. *Snippet 17* shows how the application specific connector calls the *DocumentList()* method on generic connector for Asynchronous Request.

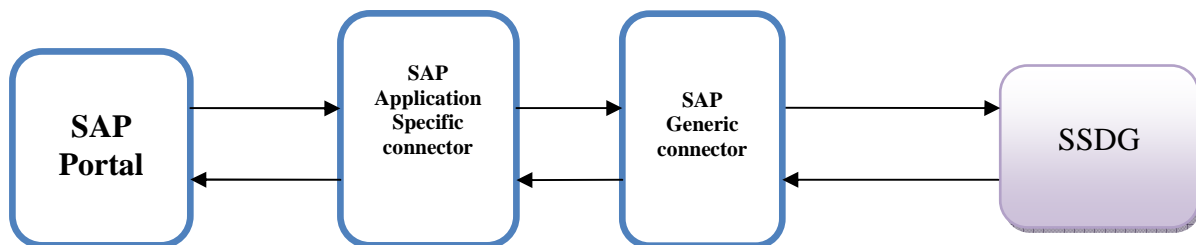


Figure 22 : List Request by SAP to track status of all requests and response from SSDG

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.IO;
5 using System.Xml.Serialization;
6 using System.Xml;
7 using SAPConnector;
8 namespace SAPClient
9 {
  
```

```

10 public class SAPInformation
  
```

```

11     {
12         static void Main ( string [ ] args )
13         {
14             //Generic Connector API Starts here
15             ConnectorInterface inter = new ConnectorInterface ( );
16             inter .CorrelationID = "";
17             //Type of Details Requested i.e Asynchronous or Synchronous
18             inter .ResponseMode = ResponseMode.Asynchronous;
19             inter.ClassValue = "http://192.168.0.140/133";
20             //Address of SSDG
21             inter .TargetEndPoint =
22                 "http://nsdgstaging.cdacmumbai.in/gateway/services/NSDGService";
23             //Authentication Details recived after registration of SAP
24             inter .AuthMode = AuthMode .Clear;
25             inter .SenderID = "SAP1238072453839";
26             inter .Password = "[B@1f19353";
27             //Prepare Body as List request Schema provided in Manual Appendix
28             inter.BodyContent = body;
29
30             //Calling of SSDG through Generic Connector API
31             ConnectorInterface .DocumentList (ref inter );
32
33             //Recieving Result
34             string status = inter .SubmissionStatus;
35
36             //if status is response
37             string corrID=inter.CorrelationID;
38             string classID = inter .ClassValue;
39             string errorCode = inter .ErrorCode;
40             string[] errorList=new string[1];
41             errorList = inter .ErrorList;
42             string error=null;
43
44             if ( errorList != null )
45             {
46                 error = errorList [ 0 ];
47             }
48             XmlElement [ ] element=new XmlElement[1] ;
49             if ( inter .BodyContent != null )
50             {

```

```
//Recieve result as per schema mentioned in Manual
```

```
39         element = inter .BodyContent;
40     }
41 }
42 }
43 }
```

Snippet 17 : Snippet for List Request Operation on SSDG

Discussion:

- i) Code in lines 15 to 21 shows the setting of generic connector properties as received in acknowledgement asynchronous submit request is submitted.
- ii) Code in line 22 shows the setting of Body content as per status schema mentioned in *Appendix D* of Dot Net Connector Manual.
- iii) Code in line 23 is calling the ***DocumentList()*** method on SSDG to know the status for earlier submitted requests.
- iv) Based on submission status. if status is error, code in lines 28 to 35 retrieves the error.
- v) Based on submission status, if status is response, code in lines 36 to 40 retrieves the response send by SAP in XMLElement[].

Chapter 6 Inter-Gateway Communication

This scenario is applicable when SP Application is registered on some other SSDG i.e. SSDG B and SAP is registered with SSDG A. SAP which is registered on SSDG A wants to avail the service of Trademark Application registered with SSDG B. Inter-Gateway communication is always an asynchronous mode of communication and *Figure 17* shows the scenario applicability.

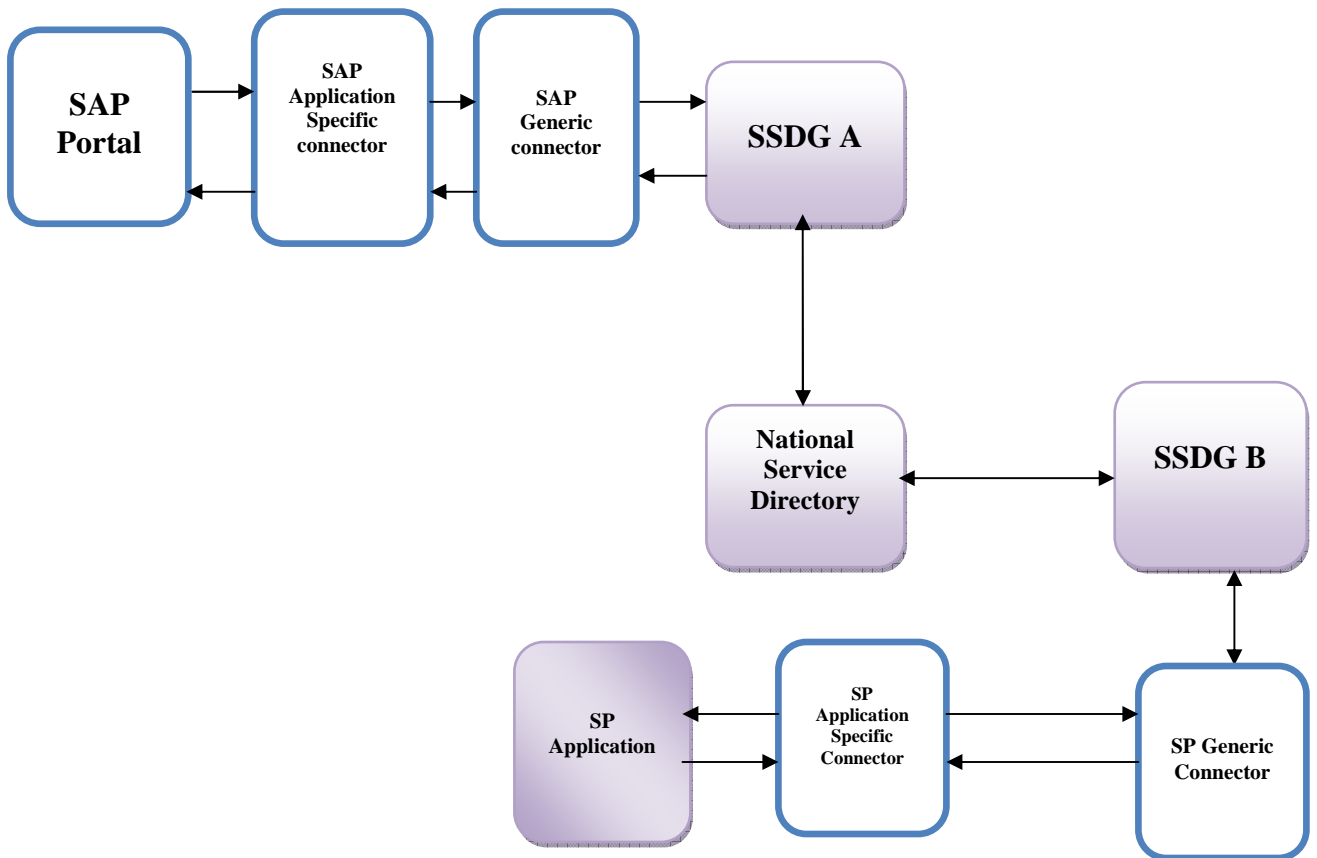


Figure 23 : Inter-gateway communication between SAP and SP

6.1 Executing Asynchronous Submit Request at SAP end

Recipe 12 Asynchronous Submit Request Execution

SAP wants to send request to SP i.e. Trademark where Trademark application is registered on SSDG B and SAP is registered on SSDG A.

Solution:

The following steps are involved in executing the asynchronous submit request in Inter-Gateway scenario. Each of these step are explained in detail along with the code snippets.

- i) *Figure 17* shows the request by SAP to Trademark application.
- ii) Use the steps for sending submit request and retrieving result as mentioned in Recipe 4.
- iii) Request received by SSDG A will be forwarded to SSDG B after resolving the address of SSDG B from National Service Directory (NSD). *Figure 18* shows the scenario where SSDG A forwards the request to SSDG B for specific application i.e. trademark requested by SAP registered with SSDG A.

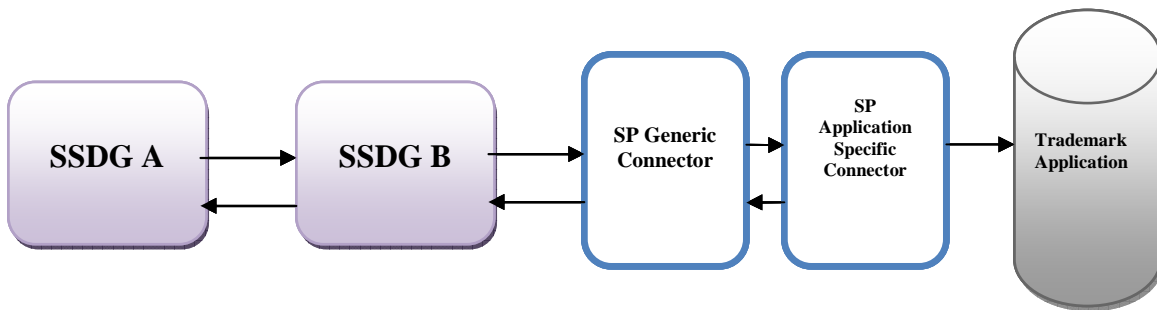


Figure 24 : Request Forwarded by SSDG A to SSDG B after resolving the address

6.2 Polling the SSDG to retrieve the Result

Recipe 13 Poll the SSDG to retrieve the result submitted by SP Application

SAP which is unaware of Inter-Gateway scenario will poll SSDG A. SSDG A will return the address of SSDG B where the Trademark application is actually registered.

Solution:

- i) *Figure 19* gives the flow for polling the SSDG A.
- ii) Polling operation will be same as defined in Recipe 9
- iii) SAP will use the details given in submit acknowledgement by SSDG A.
- iv) SAP will poll SSDG B in order to receive the response submitted by SP i.e. trademark application to SSDG B.

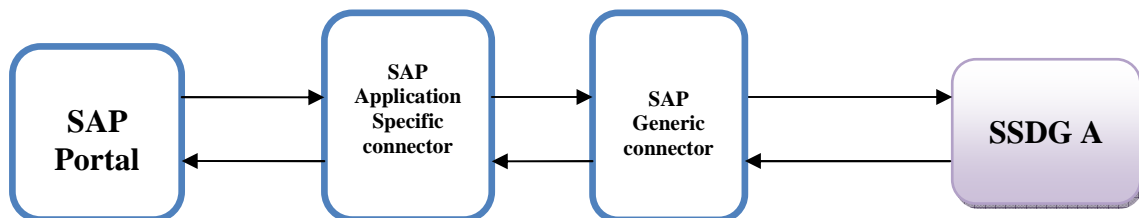


Figure 25 : Submit Poll by SAP on SSDG A to get address of SSDG B

6.3 Submitting the Response from SP Application to SSDG

Recipe 14 Submit Response from SP Application to SSDG for Asynchronous Request

As the request sent by SAP is in asynchronous mode. SP after processing the request sends the result to SSDG B. SAP needs to Poll SSDG B for receiving the result submitted by SP Application.

Solution:

SP will submit the response in Body field as an XMLElement [] to SSDG B along with the values received in request from SSDG B.

- i) *Figure 20* demonstrates the flow for submitting the response from SP application.
- ii) Submit response operation will be same as described in Recipe 7.

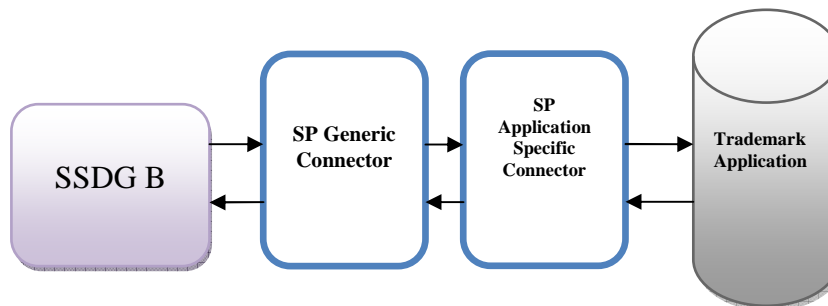


Figure 26 : Response Submitted by SP Application to SSDG B

6.4 Deleting Requests submitted to SSDG.

SAP can request for deletion of documents for which SAP has received the response through document poll from SSDG B. SAP needs to call delete request on both gateways i.e. SSDG A as well as on SSDG B.

Recipe 15 Executing Delete Request

Requests for which response has been received may be deleted using the Document Delete operation.

Solution:

Call method *DocumentDelete* () defined in SAP generic connector and pass the necessary parameters listed in *Table 10* as *Document Delete Input Parameters* in the DotNet Connector Development Manual.

- i) *Figure 21* demonstrates the flow for sending the delete request to SSDG A.
- ii) Call *DocumentDelete()* method on generic connector as described in Recipe 10.
- iii) If there is response the record is successfully deleted from SSDG A.
- iv) Call *DocumentDelete()* on SSDG B as shown in *Figure 22*. The method call will be same as described in Recipe 10.
- v) If there is response the record is successfully deleted from the SSDG B.

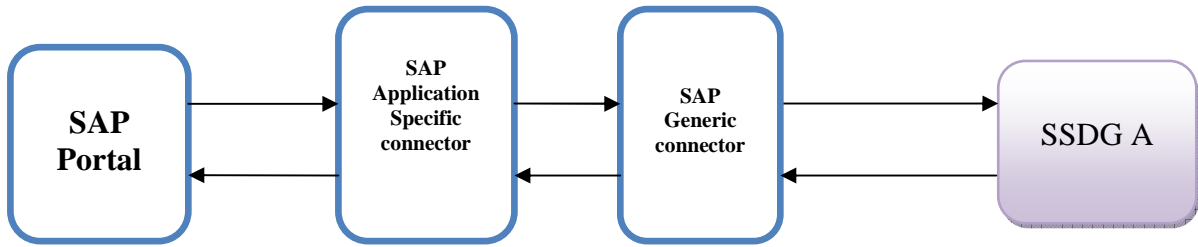


Figure 27 : Delete request on SSDG A

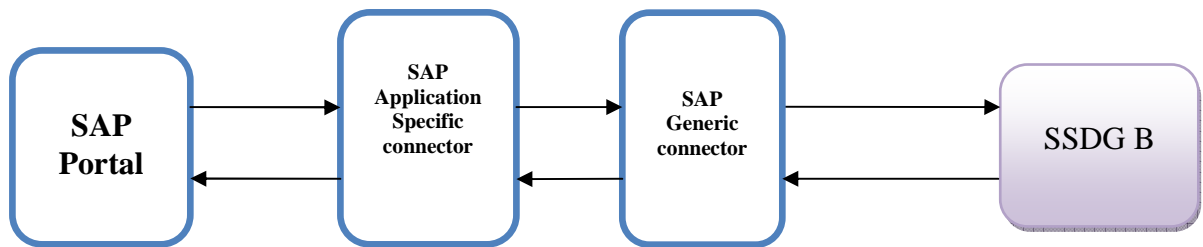


Figure 28 : Delete request on SSDG B

Additional Resources

This cookbook has been developed to help the developers in getting their job done. The examples and scenarios used are as self explanatory and lucid as can be reasonable along with a fair portion of description, explanation and screen shots to drive home the point to the developers. A fair number of scenarios have been covered in this cookbook. If in the process of reading the recipes in this cookbook, the user is left with some unanswered questions, he may refer the DotNet Connector Development Manual to find more information.

Index

SAP Application Connector, 5, 24, 30, 41, 59, 62, 65
SP Application Connector, 2, 7, 15, 16, 17, 19, 32, 33, 50, 53, 58
SAP Generic Connector, 2, 8, 21, 22, 24, 25, 44, 47, 59, 63, 66, 74
SP Generic Connector, 6, 7, 8, 9, 11, 14, 16, 21, 23, 32, 50, 52, 53, 55, 57
SubmitRequest, 25, 30, 31, 41, 42, 44, 47, 49, 60, 61, 64
DocumentPoll, 48, 61, 62, 64
DocumentList, 66, 67, 68
DocumentDelete, 63, 64, 65, 74
synSubmitRequest, 5, 16, 32, 33, 39, 40
asyncSubmitRequest, 5, 16, 50, 51, 52
submitResponse, 5, 16, 53, 54, 55, 56, 58