

# Java Connector Development Manual

# Table of Contents

<b>ACRONYMS AND ABBREVIATIONS</b> .....	<b>7</b>
<b>PART I: INTRODUCTION</b> .....	<b>8</b>
<b>CHAPTER 1 INTRODUCTION</b> .....	<b>9</b>
1.1 ROLES AND RESPONSIBILITIES OF STAKEHOLDERS.....	9
1.2 PRE-REQUISITES .....	9
1.3 HOW TO USE THIS MANUAL? .....	10
1.4 MOTIVATION FOR CONNECTORS.....	11
<b>CHAPTER 2 ALL ABOUT CONNECTORS</b> .....	<b>13</b>
2.1 REGISTRATION.....	13
2.1.1 SAP Registration.....	13
2.1.2 SAP Register Service.....	14
2.1.3 SP Registration .....	15
2.1.4 SP Service Enrolment.....	16
2.2 TYPES OF CONNECTORS.....	18
2.2.1 WHY TWO TYPES OF CONNECTORS? .....	18
2.2.2 Generic Connector.....	18
2.2.2.1 RESPONSIBILITIES OF SAP GENERIC CONNECTOR .....	19
2.2.2.2 RESPONSIBILITIES OF SP GENERIC CONNECTOR .....	19
2.2.3 Application Specific Connector.....	20
2.2.3.1 RESPONSIBILITIES OF SAP APPLICATION SPECIFIC CONNECTOR .....	20
2.2.3.2 RESPONSIBILITIES OF SP APPLICATION SPECIFIC CONNECTOR .....	20
2.2.4 Synchronous and Asynchronous Request.....	20
2.3 WORKING WITH CONNECTORS.....	22
2.3.1 Connectors in Simple Gateway Structure .....	22
2.3.2 Connectors in a Constellation of Gateways.....	25
<b>PART II: TECHNICAL DETAILS</b> .....	<b>28</b>
<b>CHAPTER 3 GENERIC CONNECTOR</b> .....	<b>29</b>
3.1 Authentication in Connectors.....	29
3.2 SAP Generic Connector.....	30
3.2.1 Responses that SAP receives.....	30
3.2.2 SAP Generic Connector Properties .....	31
3.2.3 SAP Generic Connector Functions .....	32
3.2.3.1 SEQUENCE DIAGRAM FOR ASYNCHRONOUS SUBMIT REQUEST.....	32
a) Asynchronous Submit Request.....	35
b) Submit Poll.....	38
c) Delete Request .....	40
d) List Request.....	42
3.2.3.2 SEQUENCE DIAGRAM FOR SYNCHRONOUS SUBMIT REQUEST .....	44
a) Synchronous Submit Request .....	46
3.3 SP Generic Connector.....	48
3.3.1 SP Generic Connector Functions.....	49
<b>CHAPTER 4 APPLICATION CONNECTOR</b> .....	<b>51</b>

4.1	SAP Application Connector .....	52
4.2	SP Application Connector.....	52
4.2.1	Synchronous Submit Request .....	52
4.2.2	Asynchronous Submit Request .....	54
<b>PART III:</b>	<b>CONNECTOR TROUBLESHOOTING.....</b>	<b>55</b>
<b>CHAPTER 5</b>	<b>TESTING AND TROUBLESHOOTING .....</b>	<b>55</b>
5.1	SAP Connector.....	56
5.2	SP Connector .....	72
5.3	Error Logs.....	78
<b>APPENDIX A:</b>	<b>ERROR CODES.....</b>	<b>79</b>
<b>APPENDIX B:</b>	<b>CONFIGURATION FILES FOR SAP GENERIC CONNECTOR.....</b>	<b>87</b>
<b>APPENDIX C:</b>	<b>CONFIGURATION FILES FOR SP GENERIC CONNECTOR.....</b>	<b>89</b>
<b>APPENDIX D:</b>	<b>ERROR SCHEMA .....</b>	<b>90</b>
	<i>Business Error Response Schema .....</i>	<i>90</i>
	<i>List Response Schema .....</i>	<i>91</i>
	<i>List Request Schema.....</i>	<i>92</i>

## Table of Figures

<i>Figure 1 : Block Diagram of Connector Architecture .....</i>	<i>11</i>
<i>Figure 2 : SAP Registration.....</i>	<i>14</i>
<i>Figure 3 : SAP Register Service.....</i>	<i>15</i>
<i>Figure 4 : SP Registration .....</i>	<i>16</i>
<i>Figure 5 : SP Service Registration.....</i>	<i>17</i>
<i>Figure 6 : Connectors in Simple Gateway Structure .....</i>	<i>23</i>
<i>Figure 7 : Connectors in a Constellation of Gateways.....</i>	<i>25</i>
<i>Figure 8 : Sequence Diagram for Asynchronous Submit Request.....</i>	<i>33</i>
<i>Figure 9 : Sequence Diagram for Synchronous Submit Request.....</i>	<i>45</i>

## List of Tables

<i>Table 1 : SAP Generic Connector Properties</i> .....	31
<i>Table 2 : Input Parameters for makeAsynchronousSubmitRequest</i> .....	36
<i>Table 3 : Input Parameters for makeAsynchronousSubmitRequest with SHA1 encoding</i> .....	36
<i>Table 4 : Input Parameters for makeAsynchronousSubmitRequest with digital signature</i> .....	36
<i>Table 5 : makeAsynchronousSubmitRequest O/p Parameters in case of Acknowledgement</i> ..	37
<i>Table 6 : makeAsynchronousSubmitRequest Output Parameters in case of Error</i> .....	37
<i>Table 7 : makeAsynchronousSubmitRequest Output Parameters in case of Resubmit</i> .....	38
<i>Table 8 : makeSubmitPoll Input Parameters</i> .....	38
<i>Table 9 : makeSubmitPoll Output Parameters in case of Response</i> .....	38
<i>Table 10 : makeSubmitPoll Output Parameters in case of Acknowledgement</i> .....	39
<i>Table 11 : makeSubmitPoll Output Parameters in case of Error</i> .....	39
<i>Table 12 : makeSubmitPoll Output Parameters in case of Resubmit</i> .....	39
<i>Table 13 : makeDeleteRequest Input Parameters</i> .....	40
<i>Table 14 : makeDeleteRequest Output Parameters in case of Response</i> .....	40
<i>Table 15 : makeDeleteRequest Output Parameters in case of Acknowledgement</i> .....	41
<i>Table 16 : makeDeleteRequest Output Parameters in case of Error</i> .....	41
<i>Table 17 : makeDeleteRequest Output Parameters in case of Resubmit</i> .....	41
<i>Table 18 : makeListRequest Input Parameters</i> .....	42
<i>Table 19 : makeListRequest Input Parameters with SHA1 encoding</i> .....	42
<i>Table 20 : makeListRequest Input Parameters with Digital Signature</i> .....	43
<i>Table 21 : makeListRequest Output Parameters in case of Response</i> .....	43
<i>Table 22 : makeListRequest Output Parameters in case of Error</i> .....	44
<i>Table 23 : makeListRequest Output Parameters in case of Resubmit</i> .....	44
<i>Table 24 : makeSynchronousSubmitRequest Input Parameters</i> .....	46
<i>Table 25 : makeSynchronousSubmitRequest Input parameters with SHA1 encoding</i> .....	46
<i>Table 26 : makeSynchronousSubmitRequest Input parameters with Digital Signature</i> .....	47
<i>Table 27 : makeSynchronousSubmitRequest Output Parameters in case of Response</i> .....	47
<i>Table 28 : makeSynchronousSubmitRequest Output Parameters in case of Error</i> .....	48
<i>Table 29 : makeSynchronousSubmitRequest Output Parameters in case of Resubmit</i> .....	48
<i>Table 30 : makeSubmitResponse Parameters</i> .....	49
<i>Table 31 : makeSubmitResponse Output Parameters in case of Response</i> .....	51
<i>Table 32 : makeSubmitResponse Output Parameters in case of Error</i> .....	51
<i>Table 33 : Synchronous Submit Request Input Parameters</i> .....	52
<i>Table 34 : Synchronous Submit Request Output Parameters in case of Response</i> .....	53
<i>Table 35 : Synchronous Submit Request Output Parameters in case of Error</i> .....	53
<i>Table 36 : Asynchronous Submit Request Input Parameters</i> .....	54
<i>Table 37 : Asynchronous Submit Request Output Parameters in case of Acknowledgement</i> ..	54

<i>Table 38 : Asynchronous Submit Request Output Parameters in case of Error.....</i>	<i>54</i>
<i>Table 39 : Incorrect values for makeSubmitRequest and the resultant errors .....</i>	<i>59</i>
<i>Table 40 : Incorrect values for makeSubmitPoll and the resultant errors .....</i>	<i>63</i>
<i>Table 41 : Incorrect values for makeDeleteRequest and the resultant errors .....</i>	<i>67</i>
<i>Table 42 : Incorrect values for makeListRequest and the resultant errors .....</i>	<i>71</i>
<i>Table 43 : Description of Input Parameters for callAsyncSubmitRequest .....</i>	<i>72</i>
<i>Table 44 : Description of Input Parameters for callSyncSubmitRequest .....</i>	<i>75</i>

## Acronyms and Abbreviations

**Service Access Providers (SAP):** A SAP provides easy access to services provided by the government to service seekers. Examples of SAP are front-end infrastructure (web-portal), a kiosk in rural area or a Citizen Service Centre (CSC).

**Citizen Service Centre (CSC)** : Service access point for the citizens.

**Service Provider (SP):** SP is a back-end government department or any other third-party agency offering e-services to citizens, businesses and to other government departments.

**State e-governance Service Delivery Gateway (SSDG):** SSDG is a service exchange infrastructure that routes the services desired by the end user to the appropriate service provider (state department). It thus acts as a hub for e-governance services to be functional at state level.

**Interoperability Interface Protocol/Specification (IIP/S):** The protocol used to communicate with the Gateway by the client application such as SAP and SP.

**Implementation Agency (IA):** Agencies which are appointed by the States to deploy SSDG and develop application specific connector to connect the applications to SSDG. The implementation agencies are to be chosen from the list of the DIT empanelled agencies for eforms on State Portal and SSDG project.

## **Part I: Introduction**

## Chapter 1 Introduction

This manual is intended for the developers who want to build an application specific connector to communicate with SSDG using the Java framework.

### 1.1 Roles and Responsibilities of Stakeholders

The stakeholders in this project are C-DAC, empanelled implementation agencies and the state governments.

#### 1.1.1 State government

- The state government is the one who initiates the deployment of SSDG and owns it.
- It acquires the gateway from C-DAC through DIT.

#### 1.1.2 Implementation Agency

- Will be deploying the SSDG.
- Will be developing the connectors for applications given by state government departments to communicate with SSDG.
- Will be providing the maintenance for the same for three years.

#### 1.1.3 C-DAC

- Will provide the SSDG software product and the SSDG stack
- Will provide the Bill of Material required for SSDG
- Will provide the patches for the SSDG software product
- Will provide the centralized support for the SSDG software product
- Will provide the manuals
- Consultancy for developing the application specific connectors to the implementation agencies will be provided by C-DAC.
- Generic connectors will be provided by C-DAC.

### 1.2 Pre-requisites

- Basic knowledge about Java 1.6 / J2EE Technology
- Knowledge of XML Serialization and De-serialization.

- Know how of communicating with Web services developed using J2EE Technology

### 1.3 How to use this Manual?

This manual is divided into three sections each explaining different aspect of the connector development.

**Note:** Users who are well acquainted with the basic concepts and working of connectors, may skip Part 1 and directly refer Part 2 for technical details.

#### *Part 1 (Introduction)*

Part One provides an introduction to connectors. It gives an overview about connectors and their role in communication between the SAP and the SP. It also lists the types of connectors and their positioning in the SSDG architecture.

**Chapter 1:** Covers the prerequisites for connector development. Provides details about the types of connectors and their significance in SSDG. Also covers roles and responsibilities of various stakeholders.

**Chapter 2:** Extensively describes the significance, functionality and working of the connectors in SSDG architecture.

#### *Part 2 (Technical Details)*

It covers the technical aspect of the connectors. Explanation of various features and functionality of the connectors is provided in this section.

**Chapter 3:** Describes the authentication mechanism for connectors as well as the details of generic connectors.

**Chapter 4:** Covers development of application specific connectors.

#### *Part 3(Troubleshooting)*

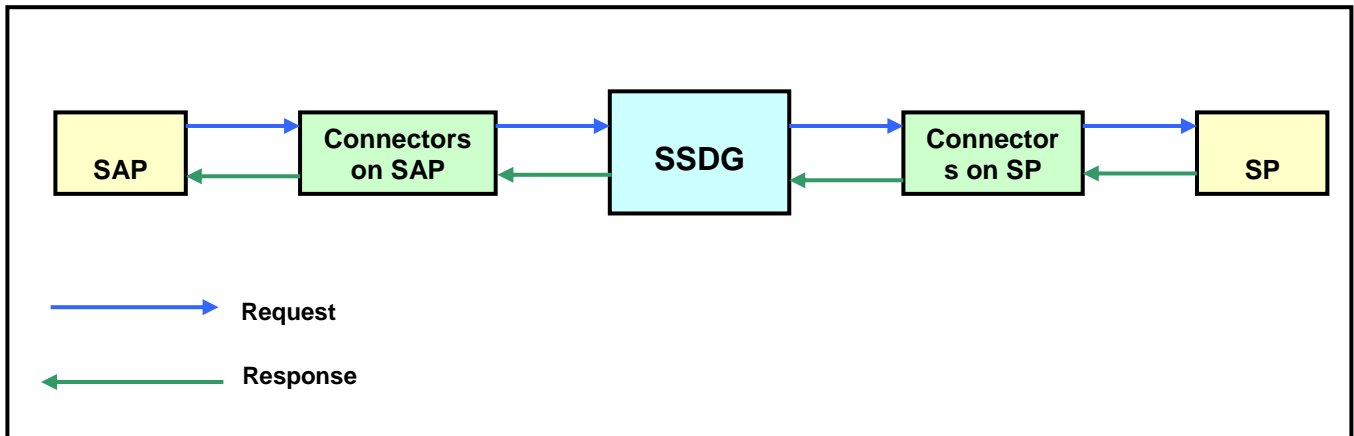
This part deals with troubleshooting related to connector development. It lists out the various error codes and logs.

**Chapter 5:** Lists out the various error codes that may be faced during the development of application specific connectors. It also explains the significance of maintaining error log for efficient trouble shooting and problem fixing.

## 1.4 Motivation for Connectors

The core functionality of the SSDG is to provide messaging communication between the SAP and the SP and vice versa for a citizen to avail the various services offered by SP. The citizen makes use of SAP to avail services offered by SP. The SAP and SP in turn communicate with each other through SSDG for rendering services to the citizen. In order to accomplish this task, citizen’s request should be in the format understandable to SSDG. Here the connectors come into picture. Connectors are nothing but a piece of software that channels the data between SAP and SP through SSDG by manipulating the data as per standards and specifications of SSDG and later of SP. Connectors may also be looked upon as an integration mechanism between SAP and SP and SSDG

Connectors, as the name suggests, serve the basic purpose of connecting Service Access Provider (SAP) and Service Provider (SP) to State e-governance Service Delivery Gateway (SSDG).



**Figure 1 : Block Diagram of Connector Architecture**

A connector is essentially a software (a piece of code in the form of DLL or Web-Service) acting as an adapter to communicate with SSDG. The messages that are exchanged between the SAP and SP applications through SSDG pass through the connector first i.e. SAP and SP send and receive messages through SSDG by using the services of the connector.

On SAP as well as SP side, there is a requirement for two units of connectors to be operational viz., *Generic connector* and *Application Specific Connector*. The application

specific connector uses the APIs provided by the generic connector for passing the data provided by SAP and SP on either side. Whereas, the generic connector which is provided by C-DAC for both SAP and SP side, interfaces with the gateway and forwards/receives this data to/from the gateway by complying to Gateway Interoperability standards (IIP/IIS).

## Chapter 2 All About Connectors

This chapter covers the details about the communication between the SAP and SP that occurs for accessing and providing a service through SSDG. The role of connectors in this exchange is covered in this chapter.

### 2.1 Registration

Before getting into the details of the connector, one needs to be familiar with the functioning of SAP and SP. It is important to note that SAP needs to be registered with SSDG before availing the services of the SP. Similarly, SP needs to register with SSDG to provide its services to SAP.

Also, it is equally mandatory for the SP to register each of its services individually with SSDG. Similarly, the SAP has to register for each service individually for accessing those services.

Note : SAP needs to be registered to SSDG to avail any services. SP and the services offered should be separately registered to SSDG for providing these services on SSDG

#### 2.1.1 SAP Registration

The SAP has to register with the SSDG for availing services of any SP. For this, the SAP needs to fill up the *SAP Registration* form on the relevant State Portal giving details like Organisation Name, Description, Valid Upto, Contact Person, Address, City, State, Pin Code, Phone No and E-mail address. Once the SAP is registered, an SAP ID along with a Password shall be sent through e-mail to SAP. SAP can now avail of various services using this SAP ID and Password.

**Figure 2 : SAP Registration**

### 2.1.2 SAP Register Service

As mentioned earlier, along with registering itself with SSDG, SAP also needs to register the services it wants to avail. For this, SAP has to fill up **SAP Register Service Form** on the relevant State Portal enlisting details like Service ID (ID of the service SAP desires to avail), SAP ID and Validation date (date upto which the services can be accessed by SAP) for accessing that service.



Figure 3 : SAP Register Service

### 2.1.3 SP Registration

Similarly, SP also has to register with SSDG for providing its services. For this, SP needs to fill *SP Enrolment Form* on the relevant State Portal providing following details:

Organisation Name, Description, Contact Person, Address, City, State, Pin Code, Phone No and E-mail address. Once SP has registered, an SP ID along with a Password shall be sent through e-mail to SP.

The screenshot displays the 'SP Enrolment Form' on the NSDG portal. The form includes the following fields:

- Organization Name \*
- Description \*
- Contact Person \*
- Address \*
- City \*
- State \* (dropdown menu showing 'STATE')
- Pin code \*
- Phone No \*
- Email Address \*

Navigation buttons at the bottom of the form are 'Submit', 'Reset', and 'Home'. The footer text reads: 'Terms of Use | Disclaimer | Designed managed and maintained by www.cdacmumbai.in @ DIT'.

**Figure 4 : SP Registration**

On completion of all the above mentioned steps, SP will be designated as registered service provider.

### 2.1.4 SP Service Enrolment

SP needs to register its services with SSDG for making them available to SAP. Each and every service of SP has to be enrolled separately by providing details of the service. To do so, SP needs to fill up **Service Enrolment Form** on the relevant State Portal. SP has to give the details such as Service Name, Service URL, Description, Method URL, Method Name, Domain, Authentication Type, Response Type, Poll Interval, SP ID. Upon registering with SSDG, a unique Class ID is assigned to the service registered by SP. This Class ID is critical in resolving services when a request for a particular service is raised.

**Note:** *Authentication Type* determines how SAP will be authenticated by SP. Note that when SAP makes a request to SP, the SAP application connector should ensure that the property pertaining to the Authentication Mode is set as per the requirement of service registered by SP.

*Response Type* determines how SP will respond to the request by SAP. While making request, the SAP application connector sets this property as per the requirement of service registered by SP. There are 2 types of response based on the 2 types of requests. In case of synchronous request, the response is served in a single cycle whereas in case of asynchronous request the response submitted using the API of SP generic connector once SP application processes the request.

The screenshot displays the NSDG website interface. At the top, there is a header with the logo of the Department of IT, Government of India, and the text 'National e-Governance Service Delivery Gateway'. Below the header, there are navigation links: HOME, CONTACT US, SITE MAP, and LOGOUT. The main content area is titled 'Service Enrolment Form' and contains a form with the following fields:

- Service Name \*
- Service URL \*
- Description \*
- Method URL \*
- Method Name \*
- Domain \*
- Authentication Type \* (Dropdown menu with '-Authentication Type-')
- Response Type \* (Radio buttons for Synchronous and Asynchronous, with Asynchronous selected)
- Poll Interval
- SP ID \*

At the bottom of the form, there are three buttons: Submit, Reset, and Home. A callout box with a red border and a yellow background points to the 'Reset' button, containing the text: 'Click on the Reset button to clear the field'.

**Figure 5 : SP Service Registration**

## 2.2 Types of Connectors

There are two types of connectors operational on the SP and SAP side,

- i. Generic Connector.
- ii. Application Specific Connector.

### 2.2.1 Why Two Types of Connectors?

Connectors need to cater two types of functionalities (i) common tasks that are required for communicating with the SSDG and (ii) functionalities to communicate with SP. While generic connector provides functionalities for some common tasks associated with Gateway communication, the application specific connector is responsible for the functionalities related to application needs.

It also provides with separation of concern between Implementation Agency and C-DAC. C-DAC will own up responsibilities towards Generic connector whereas provisioning and maintenance of Application Specific connector will be Implementation Agency's responsibility.

There are some common or general requirements that are to be fulfilled for communicating with Gateway to comply with the gateway specifications which are taken care in the generic connectors. The application level requirements vary depending on factors like application vendors, the application development platform, hence there is a need for application specific connector to fulfil these requirements at application level leading to two separate connectors, generic and application specific, both at SAP and SP side.

### 2.2.2 Generic Connector

Generic connector consists of a set of APIs (*ConnectorInterface class*) that are exposed to the application specific connector. Application specific connectors should use these APIs to communicate with generic connectors. Generic connector packages the request in a format required by SSDG.

### 2.2.2.1 Responsibilities of SAP Generic Connector

- i.) Request packaging: When a request is sent from SAP to SP for availing the services of SP, SAP generic connector converts this request message for compliance with the Interoperability protocol (IIP) used by Gateway.
- ii.) Web service communication: in IIS compliant format.
- iii.) 3 modes of Authentication support: SAP generic connector also provides authentication features for validating SAP to SP through several authentication mechanisms like *SHA1 encoding*, *Digital Certificates* or *Clear Text*. The type of authentication depends upon the services availed by SAP and decided a priori by SAP and SP.
  
- iv.) Provision for data signing for integrity check: SAP generic connector will aid the integrity check of the request message by signing the request message with digital signature. This shall be validated at the SP generic connector end to ensure that the request message has not been modified or tampered with during transit.

### 2.2.2.2 Responsibilities of SP Generic Connector

- i) Web service interface: SP Generic connector is a webservice interface (compliant to IIS) and receives the request message from SSDG in synchronous /asynchronous mode and responds accordingly.
- ii) Request extraction: It extracts the payload, the *CorrelationId*, *Transaction Id* and forwards the same to SP application by invoking the appropriate function of the SP application specific connector.
- iii) Provision for integrity check: SP generic connector carries out end-to-end integrity checks to ensure that the request message has not been modified or tampered with during transit.
- iv) Web service client: It also provides the API for SP application connector to submit the response generated by SP application. The SP application connector uses this API to submit response to SP generic connector which is then forwarded to SSDG.

### 2.2.3 Application Specific Connector

An application specific connector should use the APIs exposed by the generic connector to communicate with SSDG. Application specific connector should convert the message that is understandable to SP.

#### 2.2.3.1 Responsibilities of SAP Application Specific Connector

- i) Object to XML serialization: When a request is sent from SAP to SP for availing the services of SP, SAP application connector converts this request message in XML format for compliance with format specified by SP for availing its services.
- ii) Service setup: It is also the responsibility of the SAP application connector to search for the *Class ID* (*Class Id* is the Id of the service that SAP wishes to avail. This is assigned to the service by the Gateway when the service is registered with the gateway) pertaining to a specific service for availing this service.
- iii) Generic Connector API invocation: The SAP application connector invokes appropriate functions of the SAP generic connector API for availing services of SP.

#### 2.2.3.2 Responsibilities of SP Application Specific Connector

- i) Preparation of call handler: It implements the interfaces defined by SP generic connector.
- ii) Mapping of ClassID to method implementation: The SP application connector is also responsible for looking up the actual methods that SP uses to process the request and generate the response. These methods are identified based on their class ids.
- iii) Generic connector API invocation: Once identified, the SP application specific connector calls these methods for processing the request and sends back the response to SP generic connector to complete the cycle for synchronous / asynchronous message.

### 2.2.4 Synchronous and Asynchronous Request

Before we proceed with the details about how connectors work, what their significance is, when their services are used etc, we should be familiar with the types of request s that are

involved in the communication between SAP and SP. There are two types of requests possible (i) Synchronous and (ii) Asynchronous.

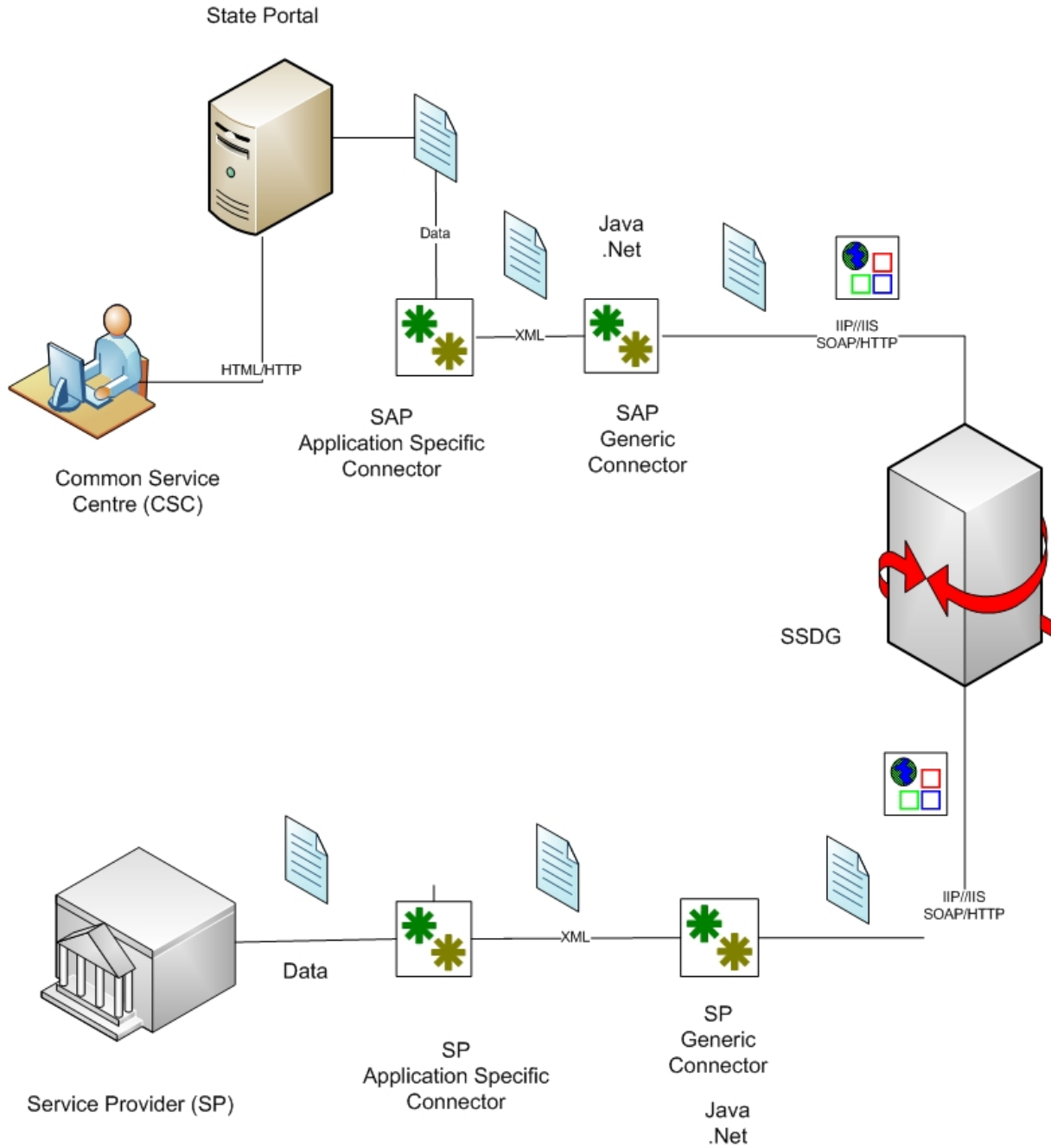
*Synchronous Request* – If a request made by SAP is served by SP without breaking the connection, then it is a Synchronous Request.

*Asynchronous Request* – If a request made by SAP is served by SP at some later point of time, then such request is termed as Asynchronous Request. The connection breaks down after the request is made and response to the request is sent at a later stage. Request and response are separate execution cycles in the case of asynchronous communication.

## **2.3 Working with Connectors**

The SSDG architecture is explained below. Here we see how SAP and SP communicate with each other through SSDG and what role the connectors have to play in facilitating this communication.

### **2.3.1 Connectors in Simple Gateway Structure**



*Figure 6 : Connectors in Simple Gateway Structure*

**Conceptual Working of Connectors on SAP side**

Request from SAP to avail SP service is sent through SSDG. Following steps explain the actions taking place at SAP side in a sequential order.

- a. SAP application sends the request to SAP application specific connector.
- b. The message request sent by SAP application specific connector should be in the XML format as per the XSD (schema) defined by SP.
- c. Application specific connector uses the API i.e. class *SAPConnector* of the SAP generic connector to pass the request in XML format to SP side through SSDG.
- d. The generic connector converts the payload as per the IIP/IIS standards before sending it to the SSDG.
- e. If the request is synchronous, the following steps are executed to serve the request.
  - Suppose a SAP requests for the PAN details of a citizen by entering his/her PAN card number.
  - This request will be served by SSDG instantaneously by fetching the PAN details of the citizen from SP based on his/her Pan Card number.
  - Thus details are available to SAP in real time basis in a single continuous cycle. This is how a synchronous request is executed on SSDG.
- f. If the request is asynchronous, SSDG returns an acknowledgement for the same to SAP. The acknowledgement contains an ID (*correlationId* generated by SSDG) for the request. Using this ID, the SAP can track the status of citizen's request by using poll operation on SSDG. The following steps are executed to serve the request but response from SP is not received immediately unlike in synchronous request.

### Conceptual Working of Connectors on SP side

The following business process is catered on SP side

- a. The SP generic connector checks whether the request message sent by SSDG is valid and takes appropriate action in case of valid/invalid request messages.
- b. If the message sent from SSDG is invalid then the SP generic connector sends the message back to SSDG along with the error message conveying appropriate reason for the error.
- c. If the message sent by SSDG is valid; the SP generic connector passes the message to the SP application specific connector by extracting the payload and other necessary details.
- d. Based on the payload and details provided by SP generic connector, SP application specific connector sends the request to SP application whose service is requested.
- e. SP application specific connector then formats and sends back the response returned by SP application to SP generic connector using the SP generic connector API.(for

asynchronous request, the response is returned using generic connector API whereas in case of synchronous request the response is sent in a single call to SP).

- f. The generic connector converts the payload as per the IIP/IIS standards before sending it to the SSDG which in turn sends it to SAP generic connector.

### 2.3.2 Connectors in a Constellation of Gateways

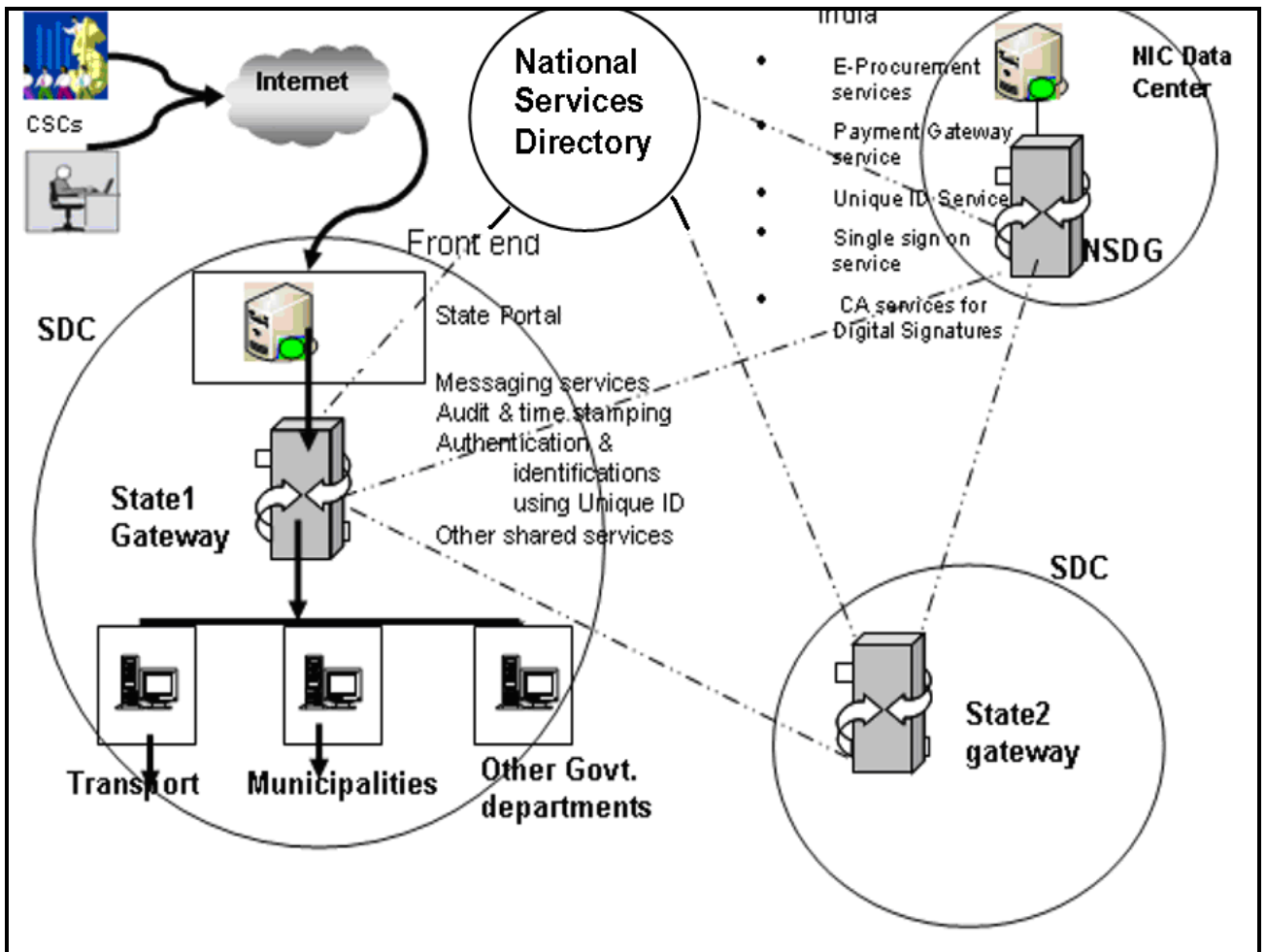


Figure 7 : Connectors in a Constellation of Gateways

In the above scenario for a constellation of Gateways, SAP and SP communicate in a similar pattern as discussed above. However, there is a possibility that the requested SP may not be registered on the same gateway as that of the SAP. In this case, when request from SAP reaches the gateway to which it is registered, the gateway looks up the relevant gateway in

National Services Directory (NSD) and communicates the SAP request to the respective Gateway.

The working of connectors in a constellation of Gateways is explained below.

### **Conceptual Working of Connectors on SAP side**

Request from SAP to avail SP service is sent through SSDG.

- a. SAP sends the request to SAP application specific connector.
- b. The message request sent by SAP application specific connector should be in the XML format as per the XSD (schema) defined by SP.
- c. Application specific connector uses the API i.e. class *SAPConnector* of the SAP generic connector to pass the request in XML format to SP side through SSDG.
- d. The generic connector converts the payload as per the IIP/IIS standards before sending it to the SSDG.
- e. If the request is synchronous, the following steps are executed to serve the request. If the request is asynchronous, SSDG returns an acknowledgement for the same to SAP. The acknowledgement contains an ID for the request. Using this ID, the citizen can track the status of his request by using poll operation on SSDG. The following steps are then executed to serve the request but response from SP is not received immediately unlike in synchronous request.
- f. SSDG (Gateway A in this case) looks up for the relevant SP registered with it to forward the request. If the SP is not registered with Gateway A, it will look up in the National Services Directory (NSD) to locate a gateway where the desired SP is registered. If found, the NSD returns the location of the gateway (Gateway B) where SP is registered.
- g. Gateway A will now pass the request to the Gateway B where SP is registered.
- h. Once the request reaches the Gateway B, it is channelized to SP as described below.

### **Conceptual Working of Connectors on SP side**

The following business process is catered on SP side.

- a. The SP generic connector checks whether the request message sent by SSDG is valid and takes appropriate action in case of valid/invalid request messages.
- b. If the message sent from SSDG is invalid then the SP generic connector sends the message back to the Gateway B along with the error message conveying appropriate reason for the error.
- c. If the message sent is valid, the SP generic connector passes the message to the SP application specific connector by extracting the payload and other necessary details.

- d. Based on the payload and details provided by SP generic connector, SP application specific connector sends the request to SP whose service is requested.
- e. SP application specific connector then formats and sends back the response returned by SP application to SP generic connector using the SP generic connector API.(for asynchronous request, the response is returned using generic connector API whereas in case of synchronous request the response is sent in a single call to SP)
- f. SP generic connector then sends back the necessary response to SAP through Gateway B.

It is noteworthy that in the above scenario where a constellation of Gateways is involved, when the `makeSubmitPoll()` is invoked for the first time, the request shall be submitted to the original Gateway. The Gateway will then look up in the NSD where the service pertaining to the request is registered. The Gateway will then route the request to the respective Gateway where the service is registered and if the response for the request has been submitted, the same will be delivered. If not, then again, when the `makeSubmitPoll()` is invoked, this time the request will be directly routed to the corresponding Gateway without involving the original Gateway.

Also, when the `makeDeleteRequest()` is invoked, the request needs to be deleted from both the original Gateway as well as the Gateway where the services are actually registered.

# Part II: Technical Details

## Chapter 3      Generic Connector

This Chapter covers all aspects of Generic Connector including SAP Generic Connector, SP Generic Connector, their properties and functionalities in detail. SAP connector communicates with SAP application specific connector and SSDG. It provides APIs for SAP application specific connector to communicate through SSDG.

SP generic connector takes a request from SSDG and forwards it to the targeted SP by invoking appropriate functions on the SP application connector. The SP application connector finally calls the desired service of SP to serve the request. The SP generic connector provides interface to the SP application connector using which SP application connector invokes a function *makeSubmitResponse()* on SP generic connector and forwards the response for an asynchronous request to SSDG.

Before we go into the details about the types of connectors and their working, we will take a look at the authentication features provided by connectors.

Apart from assuring delivery and acknowledgement of requests between the SAP and SP, connectors also perform the indispensable task of authenticating the SAP to the SP so as to ensure valid and authentic SAP interact with valid and authentic SP services.

### 3.1 Authentication in Connectors

When SAP makes a request to SP, SAP application connector ensures that the property pertaining to the Authentication Mode is set as per the requirement of service registered by SP. This Authentication Mode may be either of the following.

- i) **Clear Text** - If the desired Authentication Mode requested by SP is Clear Text in a request, then SAP application connector will set the property pertaining to Authentication Mode as Clear Text. SAP generic Connector will not encrypt the password set as Clear Text and pass the request to SSDG. As there is no encryption in Clear Text, the security ensured by this type of authentication mode is much lesser than SHA1 and Digital Signature.
- ii) **SHA1** - If the desired Authentication Mode by SP is SHA1, then SAP application connector will set the property pertaining to Authentication Mode as SHA1. SAP generic connector will perform SHA1 based hashing and base64 Encoding. The SHA1 encryption offers much more security and lesser vulnerability than Clear Text authentication mode.

- iii) **W3C Signed** – If the Authentication Mode requested by the SP is a digitally signed certificate in a request, then application connector will set the property pertaining to Authentication Mode as W3C Signed. It will also set the path of the file which is to be used for digital signature. SAP generic connector will then execute the algorithm for digital signing of the file before passing the request to SSDG. Digital signatures are the most secure way of ensuring information security than Clear Text or SHA1.

**Note:** *Authentication Mode* indicates how the SAP will be authenticated to SP. It is imperative that only authentic SAP interact with the SP. Hence every SP service dictates that any SAP interacting with it should validate itself through one of the above mentioned modes. The Authentication Mode is specified by the SP service.

## 3.2 SAP Generic Connector

The SAP generic connector provides interface depending on the type (asynchronous/synchronous) of SAP request. This interface is exposed to the SAP application connector.

### 3.2.1 Responses that SAP receives

Following are the three types of responses that can be received when services of SAP generic connector are invoked.

**Error-** The Submit Request method will return appropriate error if any of the inputs set by SAP application specific connector are invalid or gateway is unreachable.

**Acknowledgement-** A *correlationId* with status acknowledging the receipt of request is returned in case of successful submission of request. Acknowledgements are returned only in case of Asynchronous Requests.

**Response** – The response for the submitted request.

**SAP Generic Connector Functions** - There are 5 methods defined in SAP generic connector API.

- a) *makeSynchronousSubmitRequest()* – This method is used for making initial request to SSDG for availing a service in synchronous mode.
- b) *makeASynchronousSubmitRequest()* - This method is used for making initial request to SSDG for availing a service in asynchronous mode.

- c) *makeSubmitPoll()* – This method is used to check the status of the request which has been submitted through asynchronous mode.
- d) *makeListRequest()* – This method helps in getting the status of all the requests submitted to SSDG during a particular period.
- e) *makeDeleteRequest()* - This method is used to delete the request and response which have already been served successfully.

Each of these functions is explained in detail below in Section 3.2.3.

**Note:** Only *makeSynchronousSubmitRequest()* is used for Synchronous Requests while the remaining functions can be used for Asynchronous requests.

### 3.2.2 SAP Generic Connector Properties

Various SAP Generic Connector properties that need to be configured are given in Table 1. Terms related to SAP.

To send any type of request, SAP should know the following terms.

*Table 1 : SAP Generic Connector Properties*

Sr.No	Elements	Type	Mandatory	Description
1	<b>ClassId</b>	<b>String</b>	<b>YES</b>	It is the Service ID of the service provided by SP. The service ID may be the URL of the service which is hosted on the Service Provider (SP).
2	<b>TransactionId</b>	<b>String</b>	<b>NO</b>	Transaction ID can be used by the SAP application to club multiple Submit Request into single transaction.
3	<b>Body</b>	<b>Object</b>	<b>YES</b>	Payload for SP from SAP in the format understood by SP.
4	<b>CorrelationId</b>	<b>String</b>	<b>YES</b>	CorrelationId is used by SAP application to make <i>Submit Poll</i> and <i>Delete Request</i> . It is used to correlate <i>Submit Request</i> and subsequent <i>Submit Poll</i> and <i>Delete</i>

Sr.No	Elements	Type	Mandatory	Description
				<i>Request.</i>
5	<b>ResponseMode</b>	<b>Enum</b>	<b>YES</b>	The ResponseMode element specifies the type of response requested by the originator. (Synchronous / Asynchronous)
6	<b>NsdgTest</b>	<b>Enum</b>	<b>YES</b>	GatewayTest element indicates whether a message submitted to the Gateway is a test message or a real message. Absence of this element or value of “0” indicates that a message is a live, rather than a test, message whereas a value of “1” indicates that the message is a test message.
7	<b>StartTime</b>	<b>Calendar</b>	<b>NO</b>	This element should be present for making <i>list request</i> calls
8	<b>EndTime</b>	<b>Calendar</b>	<b>NO</b>	This element should be present for making <i>list request</i> calls
9	<b>TargetEndPointUrl</b>	<b>String</b>	<b>YES</b>	URL of the Gateway service

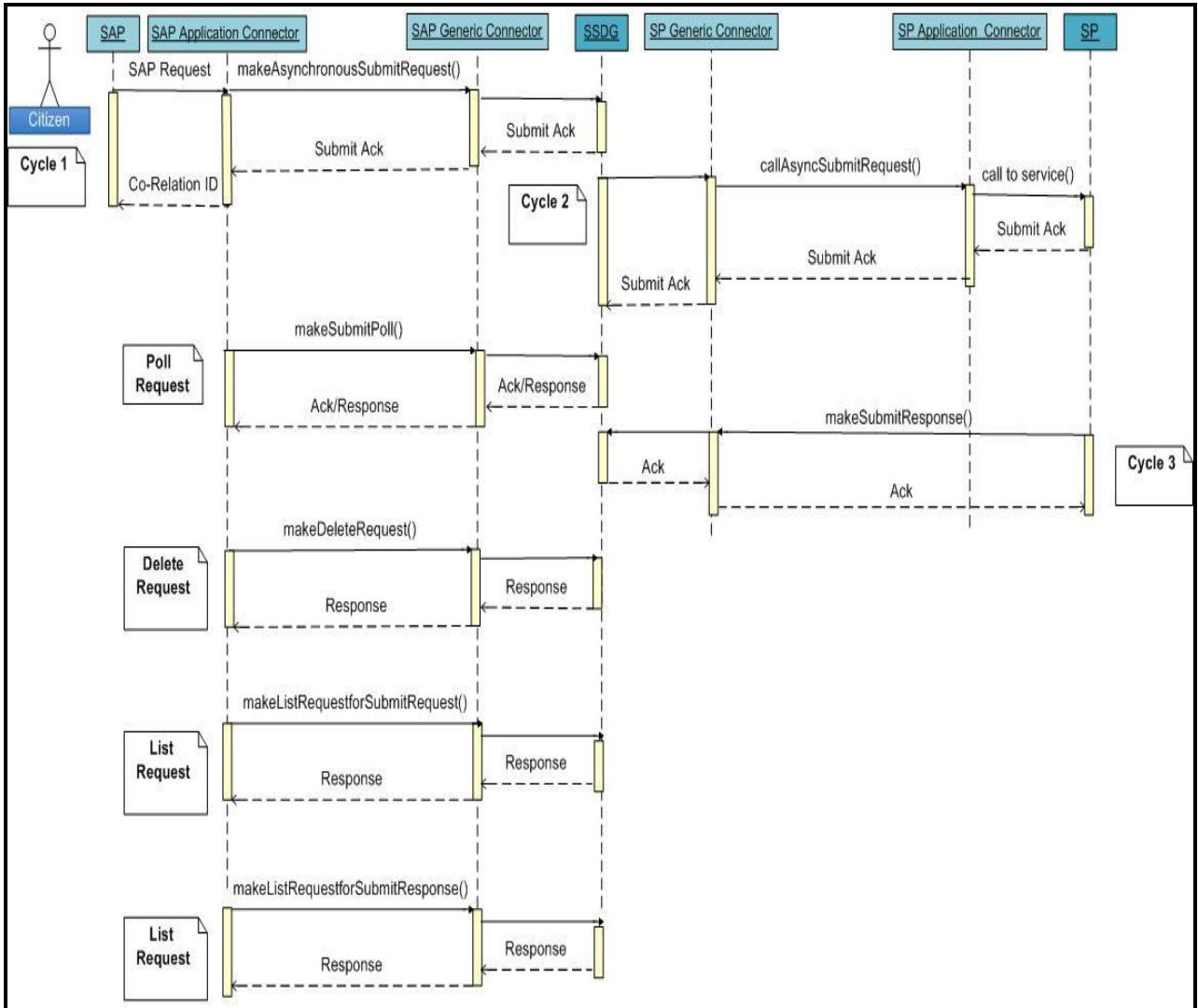
### 3.2.3 SAP Generic Connector Functions

The SAP Generic Connector Functions defined in the Generic Connector API are explained in detail below with their sequence diagrams. The inputs and outputs for the various request modes and authentication types are also provided with explanation.

#### 3.2.3.1 Sequence Diagram for Asynchronous Submit Request

The sequence diagram in Figure 3 shows the illustrative flow of **Asynchronous Request** from SAP to SP. It also depicts how SP routes the response back to SAP. During this exchange of request and response, several method calls are made between SAP and SP connectors. This handshake between various components involved is elaborated ahead.

It is worthwhile noting here that since the request call is *Asynchronous*, the citizen does not receive the response immediately. Alternatively, the citizen will receive an acknowledgement implying the receipt of the request. The response is generated and delivered at a later stage.



**Figure 8 : Sequence Diagram for Asynchronous Submit Request**

The following steps describe how SAP and SP interact with each other through the SSDG by invoking various method calls.

- 1) Suppose a citizen wants to apply for Birth Certificate through relevant Govt. department. The citizen initiates a request for the same using SAP application (Cycle 1)
- 2) SAP converts the request into the format that is understood by SP and routes it to *SAP Application Specific Connector*.

- 3) This request is then passed to *SAP Generic Connector* by invoking a method *makeAsynchronousSubmitRequest()* on *SAP Generic Connector*.
- 4) *SAP Generic Connector* delivers this request to SSDG. SSDG will then send an acknowledgement along with a *unique correlationId* to *SAP Generic Connector*. *Generic Connector* passes it back to *SAP Application Specific Connector* which delivers it to SAP Application from where it finally reaches the citizen. This *correlationId* helps the citizen in tracking the status of the request submitted. Thus Cycle 1 as depicted in Figure 3 is completed and the connection with SSDG is broken.
- 5) Now, Cycle 2 as shown in Figure 3 begins and SSDG delivers the request to *SP Generic Connector*.
- 6) This request is further forwarded to *SP Application Connector*.
- 7) Here, *SP Application Specific Connector* calls the appropriate service on SP and passes the request. The acknowledgement for the same is conveyed to SSDG through *SP Specific* and *Generic* connectors respectively. Cycle 2 completes.
- 8) Now, as Cycle 3 in Figure 3 depicts, SP generates the appropriate response once the request has been successfully processed and sends response to *SP Application Specific Connector*.
- 9) *SP Application Specific Connector* invokes the *makeSubmitResponse()* function on *SP Generic Connector* and pass the response to it.
- 10) *SP Generic Connector* then forwards this response to SSDG. SSDG acknowledges the receipt of the response to SP. Cycle 3 gets completed on submission of response to SSDG.

### **Poll Request**

After submitting the request and receiving the *correlationId*, the citizen can check the status of his request. SAP caters to this request by calling *makeSubmitPoll()* method.

**Poll Request** in Figure 3 depicts this execution.

- 1) The citizen checks the status of his/her request by submitting the *correlationId* to SAP application.
- 2) This request is picked up by *SAP Application Specific Connector* which invokes the *makeSubmitPoll()* method on *SAP Generic Connector*.

- 3) *SAP Generic Connector* gets status of the request from SSDG. If the response has already been generated, the same is returned in the form of response to the citizen else an acknowledgement is sent until response is awaited from SP end.

### **Delete Request**

Additionally, if the user's request has been fulfilled and he wishes to remove the request, he can do so by submitting the corresponding *correlationId*.

*Delete Request* in Figure 3 depicts this execution.

- 1) The citizen submits the *correlationId* pertaining to the request to be deleted to SAP application.
- 2) This request is picked up by *SAP Application Specific Connector* which invokes the *makeDeleteRequest()* function on *SAP Generic Connector*.
- 3) *SAP Generic Connector* passes the *Delete* request to SSDG which in turn will delete the corresponding request. It then generates a response acknowledging the delete action and returns the same to the Citizen.

### **List Request**

The citizen can also view all requests posted for a particular time interval.

*List Request* in Figure 3 depicts this execution.

- 1) The citizen specifies a particular interval for which he/she wishes to lists the requests in the SAP Application.
- 2) This *Request* is picked up by *SAP Application Specific Connector* which invokes the *makeListRequest()* function on *SAP Generic Connector*.
- 3) *SAP Generic Connector* passes the *List Request* to SSDG which will respond by listing all requests pertaining to the period specified by the citizen.

## **a) Asynchronous Submit Request**

- i) Type of Response Mode- **Asynchronous**

**Authentication Type- Clear**  
**Table 2 : Input Parameters for makeAsynchronousSubmitRequest**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.ASYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType.CLEAR	java.lang.Enum	Y
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y

**ii) Type of Response Mode- Asynchronous**  
**Authentication Type- SHA1**

**Table 3 : Input Parameters for makeAsynchronousSubmitRequest with SHA1 encoding**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.ASYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType. ENCRYPTED	java.lang.Enum	Y
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y

**iii) Type of Response Mode- Asynchronous**  
**Authentication Type- W3CSigned**

**Table 4 : Input Parameters for makeAsynchronousSubmitRequest with digital signature**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.ASYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType.	java.lang.Enum	Y

		DIGITAL_SIGNATURE		
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y

The response that SAP receives from SSDG on submission of *makeAsynchronousSubmitRequest()* operation will be either of the following;

- “SUCESS” - Indicates that the request is successfully submitted and response is awaited from SP. correlationId from SSDG is sent to SAP to track its pending requests.
- “FAILED” - Indicates that there was some error in the request that was sent due to insufficient information sent by the citizen.
- “RESUBMIT” - Indicates that there was some problem while processing the request hence citizen may be asked to resend his/her request.

### Output

**Received Data Type:** connector.messages.SubmitResponse  
**Received Type:** connector.messages.parameters.RequestStatus

*Table 5 : makeAsynchronousSubmitRequest O/p Parameters in case of Success*

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	getStatus()	RequestStatus.SUCCESS	java.lang.Enum
connector.messages.SubmitResponse	getCorrelationId ()	coorelationid	java.lang.String
connector.messages.SubmitResponse	getResponseEndPointURL ()	responseendpointurl	java.lang.String
connector.messages.SubmitResponse	getPollInterval ()	pollinterval	java.math.BigInteger
connector.messages.SubmitResponse	getResponse ()	Xmlobject	org.apache.xmlbeans.XmlObject

If Status = SUCCESS, look for ‘correlationId’, ‘responseEndPoint’ and ‘pollInterval’ using getCorrelationId(),getResponseEndPointURL(),getPollInterval().

*Table 6 : makeAsynchronusSubmitRequest Output Parameters in case of Error*

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	getStatus()	RequestStatus.FAILED	java.lang.Enum
connector.messages.SubmitResponse	getErrors()	Error array String	java.lang.String

If Status = FAILED then Array of String containing errors will be returned using *getErrors()*.

**Table 7 : makeAsynchronousSubmitRequest Output Parameters in case of Resubmit**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	<b>getStatus()</b>	<b>RequestStatus.RESUBMIT</b>	java.lang.Enum
connector.messages.SubmitResponse	<b>getReason ()</b>	reason	java.lang.String

If Status = RESUBMIT then reason will be returned using *getReason()*.

## b) Submit Poll

To get the response of previously submitted request SAP polls SSDG Gateway for the requests which SAP submitted in an asynchronous mode.

Class	Method	Argument to be passed	Data Type of Required Argument	Required
connector.messages.SubmitPoll	<b>setTargetEndPointURL()</b>	<b>targetEndPoint</b>	<b>java.lang.String</b>	Y
connector.messages.SubmitPoll	<b>setClassId()</b>	<b>classId</b>	<b>java.lang.String</b>	Y
connector.messages.SubmitPoll	<b>setTransactionId ()</b>	<b>transactionid</b>	<b>java.lang.String</b>	Y
connector.messages.SubmitPoll	<b>setNsdgTest()</b>	<b>NSDGTTest.NO</b>	<b>java.lang.Enum</b>	Y
connector.messages.SubmitPoll	<b>setCorrelationId ()</b>	<b>correlationid</b>	<b>java.lang.String</b>	Y

**Table 8 : makeSubmitPoll Input Parameters**

### Output

**Received Data Type:** connector.messages.SubmitPollResponse

**Received Type:** connector.messages.parameters.RequestStatus

**Table 9 : makeSubmitPoll Output Parameters in case of Response**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitPollResponse	<b>getStatus()</b>	<b>RequestStatus.SUCCESS</b>	java.lang.Enum
connector.messages.SubmitPollResponse	<b>getResponseDocumentType ()</b>	<b>ResponseDocumentType.RESPONSE</b>	java.lang.Enum

connector.messages. SubmitPollResponse	getResponse ()	Xmlobject	org.apache.xmlbeans.Xml Object
---	----------------	-----------	-----------------------------------

If Status = SUCCESS and getResponseDocumentType returns ‘response’, then ‘getResponse()’ will return ‘XmlObject’.

**Table 10 : makeSubmitPoll Output Parameters in case of Acknowledgement**

Class	Method	Returns	Date type of Return Value
connector.messages. SubmitPollResponse	getStatus()	RequestStatus.SUCCESS	java.lang.Enum
connector.messages. SubmitPollResponse	getResponseDocumentType ()	ResponseDocumentType.ACK NOWLEDGEMENT	java.lang.Enum
connector.messages. SubmitPollResponse	getCorrelationId ()	coorelationid	java.lang.String
connector.messages. SubmitPollResponse	getResponseEndPointURL ()	responseendpointurl	java.lang.String
connector.messages. SubmitPollResponse	getPollInterval ()	pollinterval	java.math.BigInteger

If Status = SUCCESS and getResponseDocumentType() returns ‘acknowledgement’ then retrieve ‘correlationId’, ‘responseEndPoint’ and ‘pollInterval’.

**Table 11 : makeSubmitPoll Output Parameters in case of Error**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitPollResponse	getStatus()	RequestStatus.FAILED	java.lang.Enum
connector.messages.SubmitPollResponse	getErrors()	Error array String	java.lang.String

If Status = FAILED then Array of String containing errors will be returned using *getErrors()*.

**Table 12 : makeSubmitPoll Output Parameters in case of Resubmit**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitPollResponse	getStatus()	RequestStatus.RESUBMIT	java.lang.Enum
connector.messages.SubmitPollResponse	getReason ()	reason	java.lang.String

If Status = RESUBMIT then reason will be returned using *getReason()*.

### c) Delete Request

Delete previously submitted request and response from the SSDG Gateway for asynchronous type of requests.

*Table 13 : makeDeleteRequest Input Parameters*

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.DeleteRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.DeleteRequest	setClassId()	classId	java.lang.String	Y
connector.messages.DeleteRequest	setTransactionId ()	transactionid	java.lang.String	Y
connector.messages.DeleteRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.DeleteRequest	setCorrelationId ()	correlationid	java.lang.String	Y

### Output

**Received Data Type:** connector.messages.DeleteResponse  
**Received Type:** connector.messages.parameters.RequestStatus

*Table 14 : makeDeleteRequest Output Parameters in case of Response*

Class	Method	Returns	Date type of Return Value
connector.messages.DeleteResponse	getStatus()	RequestStatus.SUCCESS	java.lang.Enum
connector.messages.DeleteResponse	getResponseDocumentType ()	ResponseDocumentType.RESPONSE	java.lang.Enum
connector.messages.DeleteResponse	getResponse ()	Xmlobject	org.apache.xmlbeans.Xml Object

If Status = SUCCESS and getResponseDocumentType returns ‘response’, then ‘getResponse()’ will return ‘XmlObject’.

**Table 15 : makeDeleteRequest Output Parameters in case of Acknowledgement**

Class	Method	Returns	Date type of Return Value
connector.messages.DeleteResponse	getStatus()	RequestStatus.SUCCESS	java.lang.Enum
connector.messages.DeleteResponse	getResponseDocumentType ()	ResponseDocumentType.ACKNOWLEDGEMENT	java.lang.Enum
connector.messages.DeleteResponse	getCorrelationId ()	coorelationid	java.lang.String
connector.messages.DeleteResponse	getResponseEndPointURL ()	responseendpointurl	java.lang.String
connector.messages.DeleteResponse	getPollInterval ()	pollinterval	java.math.BigInteger

If Status = SUCCESS and getResponseDocumentType() returns ‘acknowledgement’ then retrieve ‘correlationId’, ‘responseEndPoint’ and ‘pollInterval’.

**Table 16 : makeDeleteRequest Output Parameters in case of Error**

Class	Method	Returns	Date type of Return Value
connector.messages.DeleteResponse	getStatus()	RequestStatus.FAILED	java.lang.Enum
connector.messages.DeleteResponse	getErrors()	Error array String	java.lang.String

If Status = FAILED then Array of String containing errors will be returned using *getErrors()*.

**Table 17 : makeDeleteRequest Output Parameters in case of Resubmit**

Class	Method	Returns	Date type of Return Value
connector.messages.DeleteResponse	getStatus()	RequestStatus.RESUBMIT	java.lang.Enum
connector.messages.DeleteResponse	getReason ()	reason	java.lang.String

If Status = RESUBMIT then reason will be returned using *getReason()*.

## d) List Request

Lists the status of all the requests/responses submitted to SSDG for a particular service and mode with relevant authentication details.

It is used to determine the status of previous requests to gateway and responses received from SP. SAP generic connector provided two interfaces for List request, *makeListRequestforSubmitRequest()* and *makeListRequestforSubmitResponse()*. The former results in determining the status of submissions/requests to gateway and the later yields the responses received from service provider that have not been deleted from Gateway between start time and end time

Input and Output parameters for both types of requests are the same.

- i) Type of Response Mode – **Asynchronous / Synchronous**  
Authentication Type- **Clear**

**Table 18 : makeListRequest Input Parameters**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages. ListRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages. ListRequest	setClassId()	classId	java.lang.String	Y
connector.messages. ListRequest	setTransactionId ()	transactionid can be null	java.lang.String	Y
connector.messages. ListRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages. ListRequest	setListQualifier()	ListQualifier.REQUEST or ListQualifier.RESPONSE	java.lang.Enum	Y
connector.messages. ListRequest	setEndTime()	Start time	java.util.Calendar	Y
connector.messages. ListRequest	setStartTime()	End time	java.util.Calendar	Y
connector.messages. ListRequest	setAuthenticationType ()	AuthenticationType. CLEAR	java.lang.Enum	Y

- ii) Type of Response Mode – **Asynchronous / Synchronous**  
Authentication Type- **SHA-1**

**Table 19 : makeListRequest Input Parameters with SHAI encoding**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.ListRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.ListRequest	setClassId()	classId	java.lang.String	Y
connector.messages.ListRequest	setTransactionId ()	transactionid can be null	java.lang.String	Y
connector.messages.ListRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.ListRequest	setEndTime()	Start time	java.util.Calendar	Y
connector.messages.ListRequest	setStartTime()	End time	java.util.Calendar	Y
connector.messages.ListRequest	setAuthenticationType ()	AuthenticationType. ENCRYPTED	java.lang.Enum	Y

*Table 20 : makeListRequest Input Parameters with Digital Signature*

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.ListRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.ListRequest	setClassId()	classId	java.lang.String	Y
connector.messages.ListRequest	setTransactionId ()	transactionid can be null	java.lang.String	Y
connector.messages.ListRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.ListRequest	setEndTime()	Start time	java.util.Calendar	Y
connector.messages.ListRequest	setStartTime()	End time	java.util.Calendar	Y
connector.messages.ListRequest	setAuthenticationType ()	AuthenticationType. DIGITAL_SIGNATURE	java.lang.Enum	Y

**Output**

**Received Data Type:** connector.messages.ListResponse

**Received Type:** connector.messages.parameters.RequestStatus

*Table 21 : makeListRequest Output Parameters in case of Response*

Class	Method	Returns	Date type of Return Value
connector.messages.ListResponse	getStatus()	RequestStatus.SUCCESS	java.lang.Enum
connector.messages.ListResponse	getResponseDocumentType ()	ResponseDocumentType. RESPONSE	java.lang.Enum

connector.messages. ListResponse	getResponse ()	Xmlobject	org.apache.xmlbeans.Xml Object
-------------------------------------	----------------	-----------	-----------------------------------

If Status = SUCCESS and getResponseDocumentType returns ‘response’, then ‘*getResponse()*’ will return ‘XmlObject’.

**Table 22 : makeListRequest Output Parameters in case of Error**

Class	Method	Returns	Date type of Return Value
connector.messages. ListResponse	<b>getStatus()</b>	<b>RequestStatus.FAILED</b>	java.lang.Enum
connector.messages. ListResponse	getErrors()	Error array String	java.lang.String

If Status = FAILED then Array of String containing errors will be returned using *getErrors()*.

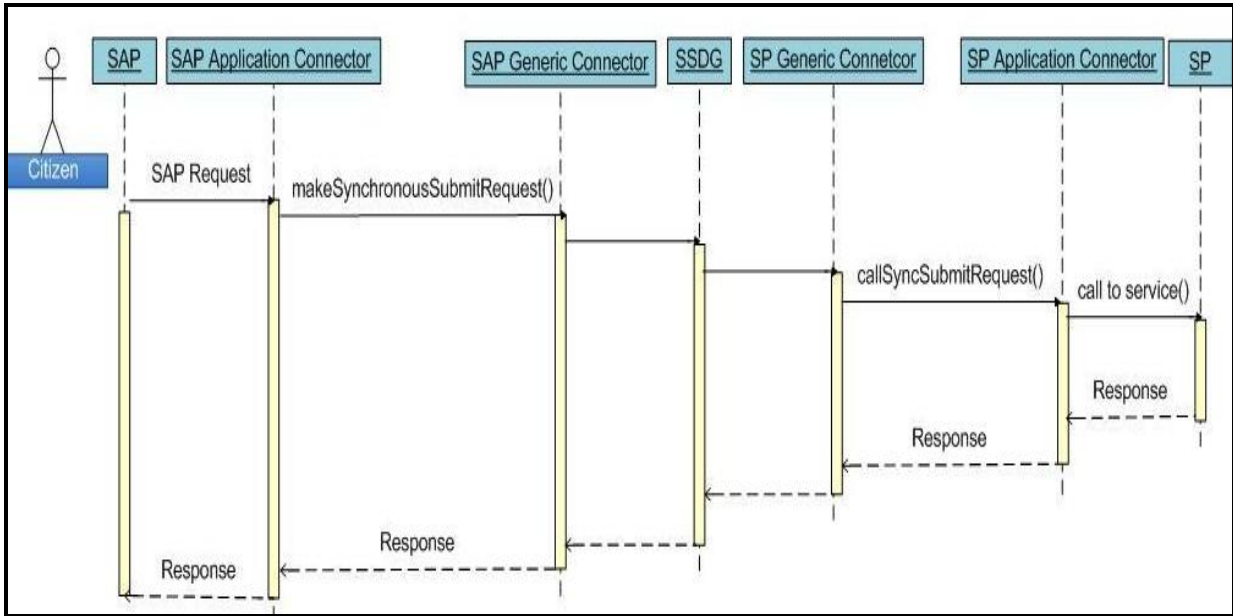
**Table 23 : makeListRequest Output Parameters in case of Resubmit**

Class	Method	Returns	Date type of Return Value
connector.messages. ListResponse	<b>getStatus()</b>	<b>RequestStatus.RESUBMIT</b>	java.lang.Enum
connector.messages. ListResponse	getReason ()	reason	java.lang.String

If Status = RESUBMIT then reason will be returned using *getReason()*.

### 3.2.3.2 Sequence Diagram for Synchronous Submit Request

The Sequence Diagram in Figure 4 shows the illustrative flow of **Synchronous Request** from SAP to SP. It also depicts how SP routes the response back to SAP. During this exchange of request and response, several function calls are made between SAP and SP connectors. This handshake between various components involved is elaborated ahead.



**Figure 9 : Sequence Diagram for Synchronous Submit Request**

The following steps describe how SAP and SP interact with each other through SSDG by invoking various function calls.

- 1) Supposing a citizen wants to look up for details pertaining to a telephone number. The citizen initiates a request for the same using SAP application.
- 2) SAP converts this request into the format that is understood by SP and routes it to *SAP Application Specific Connector*.
- 3) This request is then passed to *SAP Generic Connector* by invoking a function *makeSynchronousSubmitRequest()* on *SAP Generic Connector*.
- 4) *SAP Generic Connector* delivers the request to SSDG which in turn delivers it to *SP Generic Connector*.
- 5) *SP Generic Connector* invokes the *callSyncSubmitRequest()* on *SP Application Specific Connector* and forwards the request to *SP Application Specific Connector*.
- 6) *SP Application Specific Connector* then routes it to SP through appropriate service call and awaits response from SP.

- 7) As this is a Synchronous call, SP processes the desired request and delivers the response to *SP Application Specific Connector* without breaking the flow.
- 8) *SP Application Connector* then routes the response back to *SP Generic Connector* from where it is passed to SSDG which channels it back to *SAP Generic Connector*. The response finally reaches SAP Application through *SAP Application Connector* and is delivered to the citizen.

The details about the various input and output parameters of the API functions of SAP Generic Connector are given in the following tables.

### a) Synchronous Submit Request

- i) Type of Response Mode- **Synchronous**  
Authentication Type- **Clear**

**Table 24 : makeSynchronusSubmitRequest Input Parameters**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.SYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType.CLEAR	java.lang.Enum	Y
connector.messages.SubmitRequest	setCorrelationId ()	correlationid	java.lang.String	Y
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y

- ii) Type of Response Mode- **Asynchronous**  
Authentication Type- **SHA1**

**Table 25 : makeSynchronusSubmitRequest Input parameters with SHA1 encoding**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.SYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y

connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType. ENCRYPTED	java.lang.Enum	Y
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y
connector.messages.SubmitRequest	setCorrelationId ()	correlationid	java.lang.String	Y

iii) Type of Response Mode- **Asynchronous**  
Authentication Type- **W3CSigned**

**Table 26 : makeSynchronousSubmitRequest Input parameters with Digital Signature**

Class	Method	Argument to be passed	Data Type of Argument	Required
connector.messages.SubmitRequest	setTargetEndPointURL()	targetEndPoint	java.lang.String	Y
connector.messages.SubmitRequest	setClassId()	classId	java.lang.String	Y
connector.messages.SubmitRequest	setResponseMode()	ResponseMode.ASYNCHRONOUS	java.lang.Enum	Y
connector.messages.SubmitRequest	setNsdgTest()	NSDGTTest.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setSpAuthentication()	SPAuthentication.NO	java.lang.Enum	Y
connector.messages.SubmitRequest	setAuthenticationType()	AuthenticationType. DIGITAL_SIGNATURE	java.lang.Enum	Y
connector.messages.SubmitRequest	setMessageBody()	xmlObject	xmlobject	Y
connector.messages.SubmitRequest	setCorrelationId ()	correlationid	java.lang.String	Y

The response that SAP receives from SSDG on submission of *makeSynchronousSubmitRequest()* operation will be either of the following;

- “SUCESS” - Indicates that the request is successfully submitted and response is awaited from SP. correlationId from SSDG is sent to SAP to track its pending requests.
- “FAILED” - Indicates that there was some error in the request that was sent due to insufficient information sent by the citizen.
- “RESUBMIT” - Indicates that there was some problem while processing the request hence citizen may be asked to resend his/her request.

**Output**

**Received Data Type:** connector.messages.SubmitResponse

**Received Type:** connector.messages.parameters.RequestStatus

**Table 27 : makeSynchronousSubmitRequest Output Parameters in case of Response**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	<b>getStatus()</b>	<b>RequestStatus.SUCCESS</b>	java.lang.Enum
connector.messages.SubmitResponse	<b>getCorrelationId ()</b>	<b>coorelationid</b>	java.lang.String
connector.messages.SubmitResponse	<b>getResponseEndPointURL ()</b>	<b>responseendpointurl</b>	java.lang.String
connector.messages.SubmitResponse	<b>getPollInterval ()</b>	<b>pollinterval</b>	java.math.BigInteger
connector.messages.SubmitResponse	getResponse ()	Xmlobject	org.apache.xmlbeans.XmlObject

If Status = SUCCESS, look for ‘correlationId’, ‘responseEndPoint’ and ‘pollInterval’ using `getCorrelationId()`,`getResponseEndPointURL()`,`getPollInterval()`.

**Table 28 : makeSynchronousSubmitRequest Output Parameters in case of Error**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	<b>getStatus()</b>	<b>RequestStatus.FAILED</b>	java.lang.Enum
connector.messages.SubmitResponse	getErrors()	Error array String	java.lang.String

If Status = FAILED then Array of String containing errors will be returned using `getErrors()`.

**Table 29 : makeSynchronousSubmitRequest Output Parameters in case of Resubmit**

Class	Method	Returns	Date type of Return Value
connector.messages.SubmitResponse	<b>getStatus()</b>	<b>RequestStatus.RESUBMIT</b>	java.lang.Enum
connector.messages.SubmitResponse	getReason ()	reason	java.lang.String

If Status = RESUBMIT then reason will be returned using `getReason()`.

### 3.3 SP Generic Connector

SP generic connector takes the request from SSDG gateway and forwards it to the targeted SP. It invokes the `callSyncSubmitRequest()` / `callAsyncSubmitRequest()` for synchronous and asynchronous calls respectively for submitting the request to the SP application connector. The SP application connector will call the appropriate service to cater to the request.

The SP generic connector also provides an API for submitting the response in case of asynchronous request. The SP application connector implements this interface and calls

*makeSubmitResposne()* to submit the response to SP generic connector. SP generic connector then submits the response to SSDG.

**SP Generic Connector Functions** - The SP generic connector interface defines a single function as mentioned below;

*makeSubmitResposne()*- The SP application connector invokes this function on SP generic connector using the API provided in the form of jar. The jar carries information and properties defined in Table 24. The function *SubmitResponse()* of class *SPImpl* will be used to submit the response using the properties defined in Table 24 to Generic connector. The generic connector then forwards it to SSDG.

This function is explained in detail below in Section 3.3.1

### 3.3.1 SP Generic Connector Functions

The SP Generic Connector Function defined in the Generic Connector API is explained in detail below along with the inputs and outputs for asynchronous mode.

a) *makeSubmitResponse()*

*method signature : public ReturnResponse makeSubmitResponse(CommitResponse submitResponse)*

*parameters :*

*classname : com.cdac.messages.async.CommitResponse*

*Properties:*

**Table 30 : CommitResponse Properties/Fields**

Name	Type	Description	Required
ClassId	String	Class id of the requested service.	Y

TransactionId	String	value may be present. Otherwise it is optional.	N
CorrelationId	String	Unique Identifier of Submit Request Message for which Response is to be committed.	Y
TargetEndPointURL	String	URL of the gateway	Y
ResponseType	com.cdac.messages.async.parameters.ResponseType	Type of Response. e.g. ERROR, ACKNOWLEDGEMENT)	Y
MessageBody	Object	XMLObject(Response Payload from SP in the format understood by SAP).	Y

SAP Application Specific Connector should set these properties to appropriate values using the setter methods.

e.g. `commitResp.setPropertyName(propertyValue)`.

Return Parameters: ReturnResponse object  
 classname: `com.cdac.messages.async.ReturnResponse`

Properties:

*Table 31 : ReturnResponse Properties/Fields*

Name	Type	Description	Required
Status	com.cdac.messages.async.parameters.RequestStatus	Enum. Returns the status of “Committed Response”.	Y
TransactionId	String	String (value may be present. Otherwise it is optional).	N
CorrelationId	String	Unique Identifier of Submit Request for which Response was sent.	Y
MessageBody	Object	XMLObject(Response Payload from SP in the format understood by SAP).	Y

SAP Application Specific Connector should fetch the values of these properties using getter methods of ReturnResponse object.

e.g. `String myCorrId=returnedResp.getPropertyName();`

**Table 32 : ReturnResponse Property values in case of Successful Submission**

<i>Key</i>	<i>Value</i>
<b>status</b>	RequestStatus.SUCCESS
<b>correlationId</b>	string
<b>transactionId</b>	string
<b>auditId</b>	string

**Table 33 : ReturnResponse Property values in case of Erroneous Request found @Gateway**

<i>Key</i>	<i>Value</i>
<b>status</b>	RequestStatus.FAILED
<b>errors</b>	String []

**Table 34 : ReturnResponse Property values in case of Validation Error found @SP Generic connector.**

<i>Key</i>	<i>Value</i>
<b>status</b>	RequestStatus.RESUBMIT
<b>reason</b>	String

## Chapter 4 Application Connector

An application specific connector should use the APIs exposed by the generic connector to communicate with SSDG. Also, on the SP side the role of application specific connector is to provide methods for the SP generic connector to forward the request from SSDG to SP. The application specific connector also converts the request/message in a format understandable to SP.

## 4.1 SAP Application Connector

The SAP application connector accepts the request from the SAP application and converts the request to a format compatible to the SAP generic connector standards. It invokes the methods of SAP generic connector API and sets the properties for the request. The request is then submitted to SSDG by the generic connector.

## 4.2 SP Application Connector

The SP Application connector interface implements the API provided to SP in the form of EJB jar. The EJB name should be *SPConnectorClient*. The Remote class should contain 2 methods namely *callSyncSubmitRequest ()* and *callAsyncSubmitRequest ()* which are used to accept synchronous or asynchronous request respectively from the SP generic connector. The SP application connector then receives values in the defined functions and calls the appropriate service on the SP to cater to the request. It should be noted here that the Remote class shall only contain the method declarations whereas the implementation of these methods is given in the bean class.

**SP Application Specific Connector Functions** - The SP application specific connector interface defines the following functions;

- a) *callSyncSubmitRequest()*
- b) *callAsyncSubmitRequest()*

This function is explained in detail below in Section 4.2.1 and 4.2.2

### 4.2.1 Synchronous Submit Request

- i) Type of Response Mode – **Synchronous**

**When Synchronous request is received at SP, SP generic connector will call *callSyncSubmitRequest* of *com.cdac.sp.SPConnectorClientBean***

**method signature:** *public Object callSyncSubmitRequest(Object body, String classid, String transactionid, String correlationID)*

**classname:** *com.cdac.sp.SPConnectorClientBean*

*Table 35 : Synchronous Submit Request Parameters*

Name	Type	Description	Required
ClassId	String	indicates the class id of requested service.	Y
TransactionId	String	String (value may be present. Otherwise it is optional).	N
CorrelationId	String	Unique Identifier of message.	Y
Body	Object	Payload from SAP in the format understood by SP.	Y

After processing the request SP Application Specific Connector should return an Object of type *connector.sp.util.SPResponse*

**Return Object**

Type: *connector.sp.util.SPResponse*

*Table 36 : SPResponse Properties/Fields*

Name	Type	Description	Required
ResponseType	Enum: connector.sp.util.ResponseType	Response Type. e.g. Error or Response Acknowledgement.	Y
Response	Object	String (value may be present. Otherwise it is optional).	N

SP ASC should set the appropriate values for these properties in returned object using setter methods of SPResponse.

e.g. spResponse.setPropertyName(propertyValue);

*Table 37 : SPResponse Field Values in case of Response*

Field Name	Value
ResponseType	Response
Response	Xmlobject

If ResponseType=Response then response will be populated with the payload response which is to be sent back from SP to SAP.

*Table 38 : SPResponse Field Values in case of Error*

Key	Value
ResponseType	Error
Response	Xmlobject (Description of Error)

## 4.2.2 Asynchronous Submit Request

ii) Type of Response Mode – Asynchronous

When Synchronous request is received at SP, SP generic connector will call `callSyncSubmitRequest` of `com.cdac.sp.SPConnectorClientBean`

method signature: *public Object callAsyncSubmitRequest(Object body, String classid, String transactionid, String correlationID)*

classname: `com.cdac.sp.SPConnectorClientBean`

*Table 39 : Asynchronous Submit Request method Parameters*

Name	Type	Description	Required
ClassId	String	indicates the class id of requested service.	Y
TransactionId	String	String (value may be present. Otherwise it is optional).	N
CorrelationId	String	Unique Identifier of message.	Y
Body	Object	Payload from SAP in the format understood by SP.	Y

SP ASC should set the appropriate values for these properties in returned object using setter methods of `SPResponse`.

e.g. `spResponse.setPropertyName(propertyValue);`

### Return Object

Type: `connector.sp.util.SPResponse`

*Table 40 : SPResponse Field Values in case of Acknowledgement*

Key	Value
ResponseType	Acknowledgement
Response	Not populated

If `ResponseType=Acknowledgement` then result key will not be populated i.e. payload response will not be sent back from SP to SAP.

*Table 41 : SPResponse Field Values in case of Error*

Key	Value
ResponseType	Error

<b>Response</b>	Xmlobject (Error Description)
-----------------	-------------------------------

## **Part III: Connector Troubleshooting**

### **Chapter 5 Testing and Troubleshooting**

This section deals with the nearly critical aspect of troubleshooting errors which are likely to arise and cause disruption in the smooth functioning of connectors. Below is a guide that points out the input errors and improper property settings in connector methods, their implications and the correct input and output values.

## 5.1 SAP Connector

### i) Submit Request

#### Input-Correct way of specifying values in Submit Request

The following lines of code show the correct property settings while invoking the `makeSubmitRequest()` on SAP generic connector. All properties pertaining to `makeSubmitRequest()` are explained in Table 2,3,4,24,25 and 26 for both synchronous and asynchronous requests . The comments provide the description of the properties.

```
// End Point Url of the Gateway.
String
targetEndPoint="http://nsdgstaging.cdacmumbai.in/gateway/services/NSDGService";

//End Point URL for the SP service provided at the time of Service Registration
by SAP.

//for asynchronous request set classid as follows
String classId=" http://192.168.0.133/204";
//for synchronous request set classid as follows
String classId="http://192.168.0.133/209";

SubmitRequest submitRequest=new SubmitRequest();
submitRequest.setTargetEndPointURL(targetEndPoint);
submitRequest.setClassId(classId);
// Response Mode of the Service for which SAP is registered.
submitRequest.setResponseMode(ResponseMode.SYNCHRONOUS);
//for Asynchronous mode.
// submitRequest.setResponseMode(ResponseMode.ASYNCHRONOUS);
submitRequest.setNsdgTest(NSDGTTest.NO);
//Authentication on SP Side.
submitRequest.setSpAuthentication(SPAuthentication.NO);
//Authentication Type.
submitRequest.setAuthenticationType(AuthenticationType.CLEAR);
//XmlContent to be sent to SP. (Should be Valid Xml)
submitRequest.setRequestBody(xmlObject);
SAPConnector connector = new SAPConnectorImpl();
System.out.println("Making request for class : " + classId);

//call Generic Connector API for Synchronous Submit Request.
if(request.getParameter("responsemode").equals("sync"))
{
```

```
connector.messages.SubmitResponse
submitResponse=connector.makeSynchronousSubmitRequest(submitRequest);
}
```

//Also apart from these values, specify correct values in SAP.properties and keystore.properties files.

```
System.out.println("Creating SubmitRequest to send to connector...");

SubmitRequest submitRequest=new SubmitRequest ();
submitRequest.setTargetEndPointURL(targetEndPoint);
submitRequest.setTransactionId(transactionId);
submitRequest.setClassId(classId);
submitRequest.setResponseMode(ResponseMode.SYNCHRONOUS);
submitRequest.setNsdgTest(NSDGTTest.YES);
submitRequest.setSpAuthentication(SPAuthentication.NO);
submitRequest.setAuthenticationType(AuthenticationType.CLEAR);

xmlObject=XmlObject.Factory.parse(xmlBody);
submitRequest.setMessageBody(xmlObject);

System.out.println("Submit Request Parameters have been set!");

SAPConnector connector = new SAPConnectorImpl();
System.out.println("Making submit request for class : " + classId);

if(request.getParameter("responsemode").equals("sync"))
{
    submitResponse = connector.makeSynchronousSubmitRequest(submitRequest);
}
else
{
    submitResponse = connector.makeAsynchronousSubmitRequest(submitRequest);
}
```

### Output-Response for Submit Request(Synchronous Mode)

The following is the output of makeSubmitRequest() in Synchronous mode.

//status can be either SUCCESS or FAILED.

```
System.out.println("Returned from makeSynchronousSubmitRequest");
if(submitResponse.getStatus().equals(RequestStatus.RESUBMIT))
{
    System.out.println("Submit Request is not valid.");
}
```

```
        System.out.println("Please RESUBMIT the request. Reason :  
"+submitResponse.getReason());  
    }  
    else if(submitResponse.getStatus().equals(RequestStatus.FAILED))  
    {  
        System.out.println("Request FAILED.");  
        String error="";  
        String [] errors=submitResponse.getErrors();  
        for(int i=0;i<errors.length;i++)  
        {  
            System.out.println(errors[i]);  
            error=error+"\n"+errors[i];  
        }  
    }  
    else if(submitResponse.getStatus().equals(RequestStatus.SUCCESS))  
    {  
        System.out.println("Submitted Request Successfully.");  
        System.out.println("Response Document"+submitResponse.getResponse().toString());  
    }  
}
```

### **Output-Response for Submit Request(Asynchronous Mode)**

The following is the output of makeSubmitRequest() in Asynchronous mode. If the property settings are correct, the output status will be an acknowledgement for the makeSubmitRequest().

```
if(submitResponse.getStatus().equals(RequestStatus.SUCCESS))  
{  
    //In case of Success.  
    String correlationId= submitResponse.getCorrelationId();  
    String pollInterval = submitResponse .getPollInterval ();  
}
```

### **Input-Incorrect Way of specifying values in Submit Request**

The following lines of code show the incorrect property settings while invoking the makeSubmitRequest() on SAP generic connector.

```
SubmitRequest submitRequest=new SubmitRequest();  
//blank value provided in place of classid.  
submitRequest.setClassId("");  
submitRequest.setResponseMode("");  
// Response Mode of the Service for which SAP is registered
```

Kindly refer Table 39 for more examples of incorrect values and the corresponding error message from gateway.

*Table 42 : Incorrect values for makeSubmitRequest and the resultant errors*

Sr. No	Method Name	Value	Result
1	setClassId()	Blank	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1001</b> <b>Text: The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document.The necessary element for submission are missing,please refer schema for element presence for valid submission.</b>
2.	setClassId()	12345	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1058</b> <b>Text: The supplied user credentials failed validation for the requested service.</b>
3.	setTargetEndPointURL()	Any value other than gateway url	<b>Axis Fault</b>
4.	setResponseMode()	Null	<b>Error:</b> <b>Raised by : NSDG</b> <b>Number:1002</b> <b>Text: Authentication Failure. The credentials submitted with the document are invalid.</b>
5.	setResponseMode()	Any other value than Synchronous /Asynchronous	<b>Error:</b> <b>Raised by : NSDG</b> <b>Number:1002</b> <b>Text: Authentication Failure. The credentials submitted with the document are invalid.</b>

Sr. No	Method Name	Value	Result
7.	setMessageBody()	Any other object than valid XmlObject	<b>Error :</b> Will return HashMap with key-value <b>1.reason=body was not a correct</b> <b>2.status=RESUBMIT</b>

### Output-Response for Submit Request(Synchronous/Asynchronous Mode)

The following is the output of SubmitRequest() in Synchronous/Asynchronous mode in case of incorrect property settings. The output status will generate an error with an error code and error list. The explanation for the error code can be looked up in the error codes list specified in Appendix A or in the ErrorList given in the output.

//When incorrect values are provided as input, the following code can be used to retrieve Error Number & error Text.

```
if(submitResponse.getStatus().equals(RequestStatus.FAILED))
{
System.out.println("Request FAILED.");
String error="";
String [] errors=submitResponse.getErrors();
for(int i=0;i<errors.length;i++)
{
System.out.println(errors[i]);
error=error+"\n"+errors[i];
}
}
```

#### ii) Submit Poll

#### Input-Correct way of specifying values in Submit Poll

The following lines of code show the correct property settings while invoking the makeSubmitPoll() on SAP generic connector. All properties pertaining to makeSubmitPoll() are explained in Table 8.

```
String classId="http://192.168.0.133/204";
```

```
String transactionId="";  
//Correlation id of the request for which Response is to be checked.  
//This correlationid will be received when any Asynchronous request is sent to Gateway.  
  
String correlationId="1DD174252C6648A38D6E6D410B5F431C";  
String targetEndPoint="http://nsdgstaging.cdacmumbai.in/gateway/services/NSDGService";  
SubmitPoll pollRequest=new SubmitPoll();  
pollRequest.setTargetEndPointURL(targetEndPoint);  
pollRequest.setClassId(classId);  
pollRequest.setTransactionId(transactionId);  
pollRequest.setCorrelationId(correlationId);  
pollRequest.setNsdgTest(NSDGTTest.NO);  
  
SAPConnector connector = new SAPConnectorImpl();  
SubmitPollResponse pollResponse = connector.makeSubmitPoll(pollRequest);  
  
//Apart from the above values, specify correct entries in SAP.properties file.
```

```
SubmitPoll pollRequest=new SubmitPoll();  
  
pollRequest.setTargetEndPointURL(targetEndPoint);  
pollRequest.setClassId(classId);  
pollRequest.setTransactionId(transactionId);  
pollRequest.setCorrelationId(correlationId);  
pollRequest.setNsdgTest(test);  
  
SAPConnector connector = new SAPConnectorImpl();  
SubmitPollResponse pollResponse=new SubmitPollResponse();  
  
pollResponse = connector.makeSubmitPoll(pollRequest);
```

### Output-Response for Submit Poll

The following is the output of makeSubmitPoll(). If the property settings are correct, the output status will be an acknowledgement for makeSubmitPoll()

```
SubmitPollResponse pollResponse=new SubmitPollResponse();  
if(pollResponse.getStatus().equals(RequestStatus.RESUBMIT))  
{  
System.out.println("Request failed. Please RESUBMIT.");
```

```
System.out.println(pollResponse.getReason());
}
else if(pollResponse.getStatus().equals(RequestStatus.FAILED))
{
System.out.println("Request FAILED.");
String [] errors=pollResponse.getErrors();
String error="";
for(int i=0;i<errors.length;i++)
{
System.out.println(errors[i]);
error=error+"\n"+errors[i];
}
}
else if(pollResponse.getStatus().equals(RequestStatus.SUCCESS))
{
//Check what is received - Response or ACKNOWLEDGEMENT.
if(pollResponse.getResponseDocumentType().equals(ResponseDocumentType.RESPONSE))
{
System.out.println("Response is received.");
System.out.println("RESPONSE          DOCUMENT          :
"+pollResponse.getResponse().toString());
}
elseif(pollResponse.getResponseDocumentType().equals(ResponseDocumentType.ACKNOWLEDGEMENT))
{
System.out.println("Correlation ID : "+pollResponse.getCorrelationId());
System.out.println("Response          End          PointURL          :
"+pollResponse.getResponseEndPointURL());
System.out.println("Poll Interval : "+pollResponse.getPollInterval());
}
}
```

### **Input-Incorrect Way of specifying values in Submit Poll**

The following lines of code show the incorrect property settings while invoking the makeSubmitPoll() on SAP generic connector.

In this case correlation id is not provided.

```
String classId="http://192.168.0.133/204";
String transactionId="";
String correlationId="";
```

```
String targetEndPoint="http://192.168.0.37:8080/gateway/services/NSDGService";
SubmitPoll pollRequest=new SubmitPoll();
pollRequest.setTargetEndPointURL(targetEndPoint);
pollRequest.setClassId(classId);
```

Kindly refer Table 40 for more examples of incorrect values and the corresponding error message from gateway.

*Table 43 : Incorrect values for makeSubmitPoll and the resultant errors*

Sr. No.	Field Name	Value	Result
1	setClassId()	Blank	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1001</b> <b>Text: The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document.The necessary element for submission are missing,please refer schema for element presence for valid submission.</b>
2.	setClassId()	12345	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1058</b> <b>Text: The supplied user credentials failed validation for the requested service.</b>
3.	setTargetEndPointUrl()	Any value other than gateway url	<b>Axis Fault</b>
4.	setCorrelationId()	Blank	<b>Error:</b> <b>Raised by : NSDG</b> <b>Number:1048</b> <b>Text: The submitted document contains an invalid entry for CorrelationID. If the field Function contains the value delete, then the field CorrelationID must be populated.</b>
5.	setCorrelationId()	Any String less than 32 characters	<b>Error:</b> <b>Raised by : NSDG</b>

Sr. No.	Field Name	Value	Result
			<b>Number:1001</b> <b>Text:</b> The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document.The necessary element for submission are missing,please refer schema for element presence for valid submission.
6.	setCorrelationId()	Valid 32 character string but if Record doesnot exist with Gateway for that id	<b>Error:</b> <b>Raised By:</b> NSDG <b>Number:1047</b> <b>Text:</b> Unable to retrieve data for the supplied CorrelationID. Please ensure the CorrelationID is correct, and that you have the required authentication credentials to poll this CorrelationID.

### Output-Response for Submit Poll In case of Incorrect Input.

The following is the output of makeSubmitPoll() in case of incorrect property settings. The output status will generate an error with an error code and error list. The explanation for the error code can be looked up in the error codes list specified in Appendix A or in the ErrorList given in the output.

In case of error, the SubmitPollResponse will contain following values.

```
if(pollResponse.getStatus().equals(RequestStatus.FAILED))
{
System.out.println("Request FAILED.");
String [] errors=pollResponse.getErrors();
String error="";
for(int i=0;i<errors.length;i++)
{
System.out.println(errors[i]);
error=error+"\n"+errors[i];
}
}
```

### iii) Delete Request

### **Input-Correct way of specifying values in Delete Request**

The following lines of code show the correct property settings while invoking the `makeDeleteRequest()` on SAP generic connector. All properties pertaining to `makeDeleteRequest()` are explained in Table 13.

```
String classId="http://192.168.0.133/204";
String transactionId="";
String correlationId="1DD174252C6648A38D6E6D410B5F431C";
String targetEndPoint="http://nsdgstaging.cdacmumbai.in/gateway/services/NSDGService";
```

```
DeleteRequest deleteRequest=new DeleteRequest();
deleteRequest.setTargetEndPointURL(targetEndPoint);
deleteRequest.setClassId(classId);
deleteRequest.setTransactionId(transactionId);
deleteRequest.setCorrelationId(correlationId);
deleteRequest.setNsdgTest(NSDGTTest.NO);
```

```
SAPConnector connector = new SAPConnectorImpl();
DeleteResponse deleteResponse=new DeleteResponse();
//Call API for Delete Request.
deleteResponse = connector.makeDeleteRequest(deleteRequest);
```

```
DeleteRequest deleteRequest=new DeleteRequest ();
deleteRequest.setTargetEndPointURL(targetEndPoint);
deleteRequest.setClassId(classId);
deleteRequest.setTransactionId(transactionId);
deleteRequest.setCorrelationId(correlationId);
deleteRequest.setNsdgTest(NSDGTTest.NO);

SAPConnector connector = new SAPConnectorImpl ();
DeleteResponse deleteResponse=new DeleteResponse ();

deleteResponse = connector.makeDeleteRequest(deleteRequest);
```

### **Output-Response for Delete Request**

The following is the output of `makeDeleteRequest()`.

If the input values provided are correct then the return DeleteResponse will contain following values.

```

if(deleteResponse.getStatus().equals(RequestStatus.SUCCESS))
{
//Check what is received - Response or ACKNOWLEDGEMENT.
if(deleteResponse.getResponseDocumentType().equals(ResponseDocumentType.RESPONSE))
{
System.out.println("Response is received.");
}
elseif(deleteResponse.getResponseDocumentType().equals(ResponseDocumentType.ACKNOWLEDGEMENT))
{
System.out.println("Correlation ID : "+deleteResponse.getCorrelationId());
System.out.println("Response          End          PointURL          :
"+deleteResponse.getResponseEndPointURL());
System.out.println("Poll Interval : "+deleteResponse.getPollInterval());
}
}

```

### **Input-Incorrect Way of specifying values in Delete Request**

The following lines of code show the incorrect property settings while invoking the makeDeleteRequest() on SAP generic connector.

*//In case Correlation Id provided is less than 32 characters.*

```

String classId="http://192.168.0.133/204";
String transactionId="";
String correlationId="1DD174252C6648A38D6E6D410B5F43";
String targetEndPoint="http://nsdgestaging.cdacmumbai.in/gateway/services/NSDGService";

```

```

DeleteRequest deleteRequest=new DeleteRequest();
deleteRequest.setTargetEndPointURL(targetEndPoint);
deleteRequest.setClassId(classId);
deleteRequest.setTransactionId(transactionId);
deleteRequest.setCorrelationId(correlationId);
deleteRequest.setNsdgTest(NSDGTTest.NO);

```

```

SAPConnector connector = new SAPConnectorImpl();
DeleteResponse deleteResponse=new DeleteResponse();
//Call API for Delete Request.
deleteResponse = connector.makeDeleteRequest(deleteRequest);

```

Kindly refer Table 41 for more examples of incorrect values and the corresponding error message from gateway

*Table 44 : Incorrect values for makeDeleteRequest and the resultant errors*

Sr. No.	Field Name	Value	Result
1	setClassId()	Blank	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1001</b> <b>Text: The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document.The necessary element for submission are missing,please refer schema for element presence for valid submission.</b>
2.	setClassId()	12345	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1058</b> <b>Text: The supplied user credentials failed validation for the requested service.</b>
3.	setTargetEndPointURL()	Any value other than gateway url	<b>Axis Fault</b>
4.	setCorrelationId()	Blank	<b>Error:</b> <b>Raised by : NSDG</b> <b>Number:1048</b> <b>Text: The submitted document contains an invalid entry for CorrelationID. If the field Function contains the value delete, then the field CorrelationID must be populated.</b>
5.	setCorrelationId()	Any String less than 32 characters	<b>Error:</b> <b>Raised by : NSDG</b> <b>Number:1001</b> <b>Text:The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document.The</b>

Sr. No.	Field Name	Value	Result
			necessary element for submission are missing, please refer schema for element presence for valid submission.
6.	setCorrelationId()	Valid 32 character string but if Record doesnot exist with Gateway for that id	<b>Error:</b> <b>Raised By: NSDG</b> <b>Number:1049</b> <b>Text: Unable to retrieve data for the supplied CorrelationID. Please ensure the CorrelationID is correct, and that you have the required authentication credentials to poll this CorrelationID.</b>

### Output-Response for the Delete Request

The following is the output of makeDeleteRequest() in case of incorrect property settings. The output will generate an error string with an error code and error list. The explanation for the error code can be looked up in the error codes list specified in Appendix A or in the ErrorList given in the output.

```

if(deleteResponse.getStatus().equals(RequestStatus.RESUBMIT))
{
System.out.println("Request failed. Please RESUBMIT.");
System.out.println(deleteResponse.getReason());
}
else if(deleteResponse.getStatus().equals(RequestStatus.FAILED))
{
System.out.println("Request FAILED.");
String [] errors=deleteResponse.getErrors();
String error="";
for(int i=0;i<errors.length;i++)
{
System.out.println(errors[i]);
error=error+"\n"+errors[i];
}
}

```

#### iv) List Request for SubmitRequest & for SubmitResponse.

#### Input-Correct way of specifying values in List Request

The following lines of code show the correct property settings while invoking the `makeListRequest()` on SAP generic connector. All the properties pertaining to `makeListRequest()` are explained in Table 18,19 and 20.

```
String classId="http://192.168.0.133/204";  
String transactionId="";  
String targetEndPoint="http://nsdgstaging.cdacmumbai.in/gateway/services/NSDGService";
```

```
ListRequest listRequest=new ListRequest();  
listRequest.setTargetEndPointURL(targetEndPoint);  
listRequest.setClassId(classId);  
listRequest.setTransactionId(transactionId);  
listRequest.setNsdgTest(NSDGTTest.NO);  
listRequest.setAuthenticationType(AuthenticationType.CLEAR);  
DateFormat formatter ;  
Date date ;  
Calendar endTime=Calendar.getInstance();  
endTime.setTime(date);  
Calendar startTime=Calendar.getInstance();  
startTime.setTime(date);
```

```
formatter = new SimpleDateFormat("dd/MM/yyyy");  
date = (Date)formatter.parse(startTime);  
date = (Date)formatter.parse(endTime);  
System.out.println("End Time : "+date);  
listRequest.setStartTime(startTime);  
listRequest.setEndTime(endTime);
```

```
SAPConnector connector = new SAPConnectorImpl();
```

```
ListResponse listResponse= connector.makeListRequestForSubmitRequest(listRequest);
```

```

ListRequest listRequest=new ListRequest();
listRequest.setTargetEndPointURL(targetEndPoint);
listRequest.setClassId(classId);
listRequest.setTransactionId(transactionId);
listRequest.setNsdgTest(test);
listRequest.setAuthenticationType(authenticationType);

DateFormat formatter ;
Date date ;
formatter = new SimpleDateFormat("dd/MM/yyyy");
date = (Date)formatter.parse(stTime);

System.out.println("Start Time : "+date);

Calendar startTime=Calendar.getInstance();
startTime.setTime(date);

date = (Date)formatter.parse(enTime);
Calendar endTime=Calendar.getInstance();
endTime.setTime(date);
System.out.println("End Time : "+date);

listRequest.setStartTime(startTime);
listRequest.setEndTime(endTime);
System.out.println("List Request Object Prepared.");

ListResponse listResponse=new ListResponse();

SAPConnector connector = new SAPConnectorImpl();

if(requestType.equals("request"))
{
    listResponse = connector.makeListRequestForSubmitRequest(listRequest);
}
else
{
    listResponse = connector.makeListRequestForSubmitResponse(listRequest);
}

```

### Output-Response for List Request

The following is the output of makeListRequest(). If the property settings are correct, the output status will be an acknowledgement for the makeListRequest().

```

if(listResponse.getStatus().equals(RequestStatus.RESUBMIT))
{
    System.out.println("Request failed. Please RESUBMIT.");
    System.out.println(listResponse.getReason());
}
if(listResponse.getStatus().equals(RequestStatus.FAILED))
{

```

```

        System.out.println("Request FAILED.");
        String [] errors=listResponse.getErrors();
        String error="";
        for(int i=0;i<errors.length;i++)
        {
            System.out.println(errors[i]);
            error=error+"\n"+errors[i];
        }
    }
}
if(listResponse.getStatus().equals(RequestStatus.SUCCESS))
{
    System.out.println("LIST RESP DOC : "+listResponse.getResponse().toString());
}

```

### Input-Incorrect Way of specifying values in List Request

The following lines of code show the incorrect property settings while invoking the makeListRequest() on SAP generic connector.

```

//      In case of errors
if(listResponse.getStatus().equals(RequestStatus.FAILED))
{
    System.out.println("Request FAILED.");
    String [] errors=listResponse.getErrors();
    String error="";
    for(int i=0;i<errors.length;i++)
    {
        System.out.println(errors[i]);
        error=error+"\n"+errors[i];
    }
}

```

Kindly refer Table 42 for more examples of incorrect values and the corresponding error message from gateway

*Table 45 : Incorrect values for makeListRequest and the resultant errors*

Sr. No.	Field Name	Value	Result
1.	setClassId()	Blank	<b>Error :</b> <b>Raised by : NSDG</b>

Sr. No.	Field Name	Value	Result
			<b>Number: 1001</b> <b>Text: The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. The necessary element for submission are missing, please refer schema for element presence for valid submission.</b>
2.	setClassId()	12345	<b>Error :</b> <b>Raised by : NSDG</b> <b>Number: 1058</b> <b>Text: The supplied user credentials failed validation for the requested service.</b>
3.	setTargetEndPointUrl()	Any value other than gateway url	<b>Axis Fault</b>
4.	setStartTime() , setEndTime()	<b>If value of startTime is greater than endTime</b>	<b>Error:</b> <b>RaisedBy: NSDG</b> <b>Number: 1051</b> <b>Text: The submitted document contains an invalid entry for StartDateTime and EndDateTime. You cannot have a greater StartDateTime than EndDateTime.</b>

## 5.2 SP Connector

### i) callAsyncSubmitRequest

Values received from SSDG can be retrieved in following parameters and class SPConnectorClientBean needs to be implemented as per application specific logic. Response Mode for this application is of Asynchronous type.

#### **Input-Correct Way of Receiving the Input.**

Table 43 enlists the correct property settings that are retrieved when callAsyncSubmitRequest() is invoked on SP application connector by SP generic connector. All the properties pertaining to callAsyncSubmitRequest() are explained below.

*Table 46 : Description of Input Parameters for callAsyncSubmitRequest*

Sr. No	Retrieved Input Parameters	Description
1.	BODY	Object Type (Valid XML Format)
2.	CLASSID	String Type(Used to determine the service in case the SP application provides more than one service)
3..	transactionId	String Type (It is present only if SAP has set it. It can also be null)
4.	correlationId	String Type(set by Gateway)

**Method Call- *callAsyncSubmitRequest (Object body, String classid, String transactionid, String correlationid)***

```
//Retrieve TransactionID if set by SAP
String TransactionId = transactionid;
```

```
//Retrieve Correlation ID
String CorrelationID = correlationid;
// //Retrieve Class ID
String ClassID = classid;
//Retriev Body Content
XmlObject BodyContent = (XmlObject)body;
```

Body Retrieved in this manner is used by application for further processing.

## Response Generation

The response from SP application specific connector is in the form of **class named *SPResponse***. The SP generic connector expects an object of type *SPResponse* from SP application connector.

The SP application connector uses the class id to find out the service method of SP Application to be called and then sends the body in XML format or by converting the data from XML format to java format to the appropriate method of the SP application. The SP application returns the specific connector a boolean value indicating whether it is a success condition or failure condition and sends a error document only in case of error.

The callAsyncSubmitRequest creates the response which is to be sent to the generic SP connector using this body content.

The Response might be of two types : ACKNOWLEDGEMENT and ERROR which can be set as given below

```
SPResponse response=new SPResponse();
//If NO error set Response Type as ACKNOWLEDGEMENT.
response.setResponseType(ResponseType.ACKNOWLEDGEMENT);

//Else if error Prepare an Error object.
XmlObject error = XmlObject.Factory.parse(body.toString());
response.setResponseType(ResponseType.ERROR);
response.setResponse(error);
return response;
```

**Sample Error Document**

```
<Body>
<ErrorResponse xmlns="http://www.mit.gov.in/eGov/IndiaOne/schema/errorresponse">
<Error>
<RaisedBy>Eforms SP Application</RaisedBy>
<Number>7000</Number>
<Type>Application Error</Type>
<Text>There was problem processing your request, Please retry after some time.</Text>
<Location>SP</Location>
</Error>
</ErrorResponse>
</Body>
```

**ii) callSyncSubmitRequest**

Values received from SSDG can be retrieved in fields specified in SPConnectorClientBean Class and method needs to be implemented as per application specific logic. Response Mode for this application is of Synchronous type.

**Input-Correct Way of Receiving the Input.**

Table 44 enlists the correct property settings that are retrieved when callSyncSubmitRequest () is invoked on SP application connector by generic connector. All the properties pertaining to callSyncSubmitRequest () are explained below.

Table 47 : Description of Input Parameters for callSyncSubmitRequest

Sr. No	Retrieved Input Parameters	Description
1.	BODY	Object Type (Valid XML Format)
2.	CLASSID	String Type(Used to determine the service in case the SP application provides more than one service)
3..	transactionId	String Type (It is present only if SAP has set it. It can also be null)
4.	correlationId	String Type(set by Gateway)

**Method Call-** *callSyncSubmitRequest (Object body, String classid, String transactionid, String correlationID )*

```

//Retrieve TransactionID if set by SAP
String TransactionId = transactionid;
//Retrieve Correlation ID
String CorrelationID = correlationid;
//Retrieve Class ID
String ClassID = classid;
//Retriev Body Content
XmlObject BodyContent = (XmlObject)body;

```

Body Retrieved will be used by application for further processing in synchronous manner.

## Response Generation

The response in SP application connector is in the form of **SPResponse Object**. The SP generic connector expects an object of type SPResponse from SP application connector. The SP specific connector uses the **classid** to find out the service method of SP Application to be called and sends the body either in XML format or by converting the data from XML format to java format to the appropriate method of the SP application. The SP application returns the body part either in XML format or in Java format.

The callSyncSubmitRequest creates the SPResponse object where it contains ResponseType of two types (RESPONSE and ERROR) which is sent to the generic SP connector as given below.

```
//XmlObject
//Prepare Response
XmlObject obj = XmlObject.Factory.parse(body.toString());

SPResponse response=new SPResponse();
//If no error set Response Type as RESPONSE.
response.setResponseType(ResponseType.RESPONSE);
response.setResponse(obj);

//Else if error Prepare Error object.
response.setResponseType(ResponseType.ERROR);
response.setResponse(obj);
return response;
```

### iii) makeSubmitResponse(Response Component)

**Following is the sample code for calling Response Component.**

```
SPResponseEJBHome home =
(SPResponseEJBHome)ctx.lookup("SPConnectorResponseComponent");
System.out.println("BusinessClass::sendResponse Starts...");
System.out.println("Got Home object");
SPResponseEJBRemote remote = home.create();

//process the payload which is stored in xml formatted in "ss" String variable
String ss="<?xml
version='1.0'?><FormDetails><PersonFirstName>FVSIVVJKO</PersonFirstName><Person
MiddleName>OHV
KIVJKCVNA</PersonMiddleName><PersonLastName>CKBVJKVNCVNKN</PersonLast
Name></FormDetails>";
Object resxml=null;
resxml=XmlObject.Factory.parse(ss.toString()) ;

CommitResponse commitResponse=new CommitResponse();
commitResponse.setTargetEndPointURL("http://nsdgstaging.cdacmumbai.in/gateway/service
s/NSDGService");
commitResponse.setTransactionId("123456789123456789123456789123");
commitResponse.setClassId("http://192.168.0.133/204");
commitResponse.setResponseType(ResponseType.RESPONSE);
commitResponse.setResponseBody(resxml);
System.out.println("calling EJB");
ReturnResponse returnResponse=remote.makeSubmitResponse(commitResponse);
```

**OUTPUT:**

When object of type *CommitResponse* is passed to function *makeSubmitResponse()*, the output is in the form of Object of type *ReturnResponse*, where the *STATUS* can be of three types :

SUCCESS  
FAILED  
RESUBMIT

Code for calling Response Component

```
//STATUS=RESUBMIT
if (returnResponse.getStatus().equals(RequestStatus.RESUBMIT))
{
System.out.println("status: "+returnResponse.getStatus());
System.out.println("Reason:"+returnResponse.getReason());
}
//STATUS=FAILED
else if (returnResponse.getStatus().equals(RequestStatus.FAILED))
{
System.out.println("i am end of exec"+ returnResponse.getStatus());
String[] erorResponse= returnResponse.getErrors();
for(int i=0;i<erorResponse.length;i++)
{
System.out.println("printing error :"+erorResponse[i] );
}
}

// Status =SUCESS
else if (returnResponse.getStatus().equals(RequestStatus.SUCCESS))
{
System.out.println("status :"+ returnResponse.getStatus());
System.out.println("correlationId :"+ returnResponse.getCorrelationId());
System.out.println("TransactionID :"+ returnResponse.getTransactionId());
System.out.println("AuditID :"+ returnResponse.getAuditId());
}
```

### 5.3 Error Logs

Error Logging is very critical to trouble shooting and as a norm errors should be logged for any future problem fixing. Developer would be specifying the path where the errors will be logged and in case of any disruption in working of the system, these error logs shall be made available to the concerned personnel fixing the problem.

# Appendix

## Appendix A: Error Codes

Error Code	Description
1000-1073	Major Tests
1000	The processing of your document submission failed. Please re-submit.
1001	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. The necessary element for submission are missing, please refer schema for element presence for valid submission.
1002	Authentication Failure. The credentials submitted with the document are invalid.
1003	The submitted document contained a SOAP Header. Envelope. Gateway doesn't process messages with SOAP headers.
1004	The submitted XML document failed to validate against the NSDGMessageEnvelope schema for this class of document. Missing ResponseMode field.
1006	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing Key field.
1007	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing NSDGErrors field.
1008	The submitted XML document failed to validate against the NSDGMessageEnvelope schema for this class of document. Missing NSDGTTest field.
1009	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing Transformation field.
1010	The submitted XML document failed to validate against the NSDGMessageEnvelope schema for this class of document. Missing Keys field".
1011	The submitted XML document failed to validate against the NSDGMessageEnvelope schema for this class of document. Missing Role field.

1012	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing EmailAddress field.
1013	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing Organisation field.
1014	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing Correlation ID field.
1015	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing IDAuthentication field.
1016	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing Body field.
1017	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing NSDGTimestamp Field”
1019	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Missing Location field
1020	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Missing Target Details field.
1021	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Missing Text field
1031	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Empty or Incorrectly populated SenderID Field.
1032	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Body Field Should not be left Blank for this kind of Submission.
1033	The submitted document contains an entry for CorrelationID, which is a reserved system field. This field should be left blank.
1034	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Incorrectly populated NSDGTimestamp field for this kind of Submission.”
1035	The submitted document contains an entry for ResponseEndPoint, which is a reserved system field. This field should be left blank.
1036	The submitted document contains an invalid entry for TransactionID.

1037	The submitted document contains an invalid entry for AuditID.
1038	The submitted document should not contain NSDGErrors Field
1039	When submitting a document to Gateway the only valid values for the Qualifier field are request poll, response or error
1040	The submitted document contains an invalid entry for EnvelopeVersion.
1041	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Empty or Incorrectly populated Class Field.
1042	The submitted document contains an invalid entry for Function. The only valid values for the Function field are submit, delete or list.
1043	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Empty or Incorrectly populated Transformation Field.
1044	The submitted document contains an invalid entry for EmailAddress. If the field Qualifier contains the value “poll”, then the field EmailAddress must not be populated.
1045	The submitted document contains an invalid entry for Organisation. If the field Qualifier contains the value “poll”, then the field Organisation must not be populated.
1046	The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Empty or Incorrectly populated Key Field.
1047	Unable to retrieve data for the supplied CorrelationID. Please ensure the CorrelationID is correct, and that you have the required authentication credentials to poll this CorrelationID.
1048	The submitted document contains an invalid entry for CorrelationID. If the field Function contains the value delete, then the field CorrelationID must be populated.
1049	Unable to retrieve data for the supplied CorrelationID. Please ensure the CorrelationID is correct, and that you have the required authentication credentials to delete this CorrelationID.
1050	The submitted document contains an invalid entry for Function. If the field Qualifier contains the value poll, then the only valid value for the Function field is submit.
1051	The submitted document contains an invalid entry for StartDateTime and EndDateTime. You cannot have a greater StartDateTime than EndDateTime.
1052	The submitted document contains an invalid entry for any one of the following fields StartDateTime, EndDateTime.
1053	The submitted document contains an invalid entry for Method, or an inconsistent Value entry. This field must contain the value 'MD5', 'clear' or 'W3Csigned'. If W3Csigned is specified the Value

	tag must be omitted and a Signature block must be present
1054	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Empty or Incorrectly populated Role Field.
1055	The submitted document contains an entry in the Body field. This field must be left blank for this transaction type.
1056	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated NSDGMTest Field.
1057	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. The Target Details Should not be present for this kind of submission
1058	The supplied user credentials failed validation for the requested service.
1059	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Empty or Incorrectly populated ResponseMode field.
1060	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Empty or Incorrectly X509Data field.
1061	When communicating with Gateway via HTML you MUST specify a valid ResponseEndPoint.
1062	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Empty or Incorrectly populated IDAuthentication field.
1063	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Empty or Incorrectly Populated AuthTimestamp field in Authentication Block.
1064	The submitted document contains an entry for SenderDetails, which is a reserved system field . This field should be left blank for this kind of submission.
1065	The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing or Incorrectly populated Signature field.
1066	The submitted document contains an entry for NSDGMDetails, which is a reserved system field . This field should be left blank for this kind of submission.
1067	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated Location field
1068	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Missing or Incorrectly Populated Text field

1069	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated Error field
1070	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated RaisedBy field

The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated Type field

1072	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated Number field
1073	The submitted XML document failed to validate against the NSDGMMessageEnvelope schema for this class of document. Incorrectly populated Authentication field
1100-1103	Synchronous behavior - Polling

1100	The document you have submitted has not been processed yet. The system is currently experiencing high volumes. To view this submission later use the URL provided.
1101	You have exceeded your allowable number of poll attempts. To view this submission later use the URL provided.
1102	The time allowed for the poll attempts has elapsed. To view this submission later use the URL provided.
1103	The record you have attempted to poll has been marked for deletion and therefore cannot be retrieved.
1500-1504	Messages that refer to the Test Service
1501	The total number of bytes in a document submitted to the Test Service in Gateway must not exceed 1 megabyte.
1504	The authentication details in the NSDGMMessageEnvelop Header do not match the credentials

	provided in the authentication to the Test Service.
2000-2005	Messages that refer to NSDG Services
2000	Gateway could not locate a record for the supplied Correlation ID - the submission may have been deleted or the Correlation ID may be invalid; If you have not received a response you should resubmit the document
2001	The document was an invalid size
2002	The document was an invalid size - not enough data was supplied
2003	The service associated with the submitted document type is currently unavailable
2005	Gateway has not received an acknowledgement of your submission from the Department within the permitted timescale. Please either resubmit or contact the Department directly to determine if your submission has been accepted
3000-3001	Messages that refer to Departmental services
3000	System Failure. The processing of this document has failed. Please resubmit.
3001	The submission of this document has failed due to departmental specific business logic in the <i>Body</i> .

### Error codes raised by Service Providers

Error Code	Description
<b>6000-6016</b>	<b>Major tests</b>
<b>6000</b>	"The processing of your document submission failed. Please re-submit."
<b>6001</b>	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document."
<b>6002</b>	"Authentication Failure. The credentials submitted with the document are invalid."

6003	"The submitted document contained a SOAP <b>Header.Envelope</b> . The Back Office Service doesn't process messages with SOAP headers."
6004	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>Qualifier</b> field."
6005	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated Envelope field."
6006	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>Class</b> field."
6007	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>Function</b> field."
6008	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>NSDGTest</b> field."
6009	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>Transformation</b> field."
6010	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>SenderID</b> field."
6011	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing,  Empty or incorrectly populated <b>Role.Value</b> field."
6012	<b>The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>EmailAddress</b> field."</b>
6013	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <b>Organisation</b> field."
6014	"The submitted XML document failed to validate against the NSDGMessageEnvelop schema for this class of document. Missing, Empty or incorrectly

	populated <i>CorrelationID</i> field."
6015	"The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <i>Method</i> field."
6016	"The submitted XML document failed to validate against the NSDGMMessageEnvelop schema for this class of document. Missing, Empty or incorrectly populated <i>Body</i> field."
7000-7001	<b>Messages that refer to Departmental services</b>
7000	"System Failure. The processing of this document has failed. Please resubmit."
7001	"The submission of this document has failed due to departmental specific business logic in the <i>Body</i> ."

## Appendix B: Configuration files for SAP Generic Connector

### keystore.properties file

```
filePath=C:\\nsdg_keystore\\nsdg.KeyStore
password=welcome123
aliasName=nsdg.cdacmumbai.in
```

### sap.properties file

```
#details for sap registered against nsdgstaging.cdacmumbai.in
#senderId=SAP1235550898094
#senderId=SAP1238072453839
#AsyncRequest
#senderId=SAP1316666340222
#SyncRequest
senderId=SAP1316666378862
#eBiz
#senderId=SAP1275295353872
methodClear=clear
#methodClear=W3Csigned
methodMD5=MD5
role=role
#clearPassword=[B@1f19353
#AsyncRequest
#clearPassword=[B@1831d18
#SyncRequest
clearPassword=[B@1fa3bee
#eBiz
#clearPassword=Infosys@12
SHA1Password=Infosys@12
eMailAddress=kapil@cdacmumbai.in
#sp specific authentication values
roleSP=role
```

passwordSP=passwordSP

## Appendix C: Configuration files for SP Generic Connector

jndi.properties file

This refers to the jndi.properties in the Generic SP connector(.aar file)

If the specific connector is deployed in Jboss the jndi.properties will contain:

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=202.141.151.147:1099
```

If the specific connector is deployed in Weblogic, the jndi.properties will contain:

```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://192.168.0.141:7001
```

## Appendix D: Error Schema

**Schema files for retrieving results by SAP and SP for Business related problems, List Response and List Request.**

### Business Error Response Schema

```
<?xml version="1.0" ?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:gt="http://
www.mit.gov.in/eGov/schema/NSDG/ "
xmlns:err="http://www.mit.gov.in/eGov/schema/NSDG/errorresponse"
targetNamespace="http://www.mit.gov.in/eGov/schema/NSDG/errorresponse"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
id="Error_Response"
version="1.0">
<xsd:annotation>
<xsd:documentation>This schema is used for errors returned by business systems. In these
circumstances, the header contains a single Error element of type "business" and the Body contains
further information.</xsd:documentation>
</xsd:annotation>
<xsd:element name="ErrorResponse">
<xsd:complexType>
<xsd:element ref="err:Application" minOccurs="0" maxOccurs="1" />
<xsd:element name="Error" maxOccurs="unbounded">
<xsd:complexType>
<xsd:element name="RaisedBy" type="xsd:string" />
<xsd:element name="Number" minOccurs="0" maxOccurs="1" type="xsd:integer" />
<xsd:element name="Type" type="xsd:string" />
<xsd:element name="Text" minOccurs="0" maxOccurs="unbounded" type="xsd:string" />
<xsd:element name="Location" minOccurs="0" maxOccurs="1" type="xsd:string" />
<xsd:element ref="err:Application" minOccurs="0" maxOccurs="1" />
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
<xsd:element name="Application">
<xsd:complexType>

<xsd:any namespace="##any" processContents="lax" />
<xsd:anyAttribute namespace="##any" />
</xsd:complexType>
```

```
</xsd:element>  
</xsd:schema>
```

## List Response Schema

```
<?xml version="1.0" encoding="UTF-8" ?>  
<xsd:schema xmlns="http://www.mit.gov.in/eGov/schema/NSDG/BODY/ListResponse"  
targetNamespace="http://www.mit.gov.in/eGov/schema/NSDG/BODY/ListResponse"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"  
attributeFormDefault="unqualified" version="1.0" id="NSDGBodySchemaListResponse">  
<xsd:annotation>  
<xsd:documentation>This schema is used for the Body elements for the List Response message  
document</xsd:documentation>  
</xsd:annotation>  
<xsd:element name="StatusReport">  
<xsd:complexType>  
<xsd:sequence>  
<xsd:element name="SenderID" minOccurs="0" maxOccurs="1" type="xsd:string" />  
<xsd:element name="StartTimeStamp" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />  
<xsd:element name="EndTimeStamp" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />  
<xsd:element name="StatusRecord" minOccurs="0" maxOccurs="unbounded">  
<xsd:complexType>  
<xsd:sequence>  
<xsd:element name="TimeStamp" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />  
<xsd:element name="CorrelationID" minOccurs="0" maxOccurs="1">  
<xsd:simpleType>  
<xsd:restriction base="xsd:string">  
<xsd:length value="32" />  
<xsd:pattern value="[0-9A-F]{0,32}" />  
</xsd:restriction>  
</xsd:simpleType>  
</xsd:element>  
<xsd:element name="Status" minOccurs="0" maxOccurs="1" type="xsd:string" />  
</xsd:sequence>  
</xsd:complexType>  
</xsd:element>  
</xsd:sequence>  
</xsd:complexType>  
</xsd:element>  
</xsd:schema>
```

## List Request Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.mit.gov.in/eGov/schema/NSDG/BODY/ListRequest"
targetNamespace="http://www.mit.gov.in/eGov/schema/NSDG/BODY/ListRequest"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0" id="NSDGBodySchemaListRequest">

  <xsd:annotation>
  <xsd:documentation>
This schema is used for the Body elements for the List
Request message document
  </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="StatusQuery" >
  <xsd:complexType>
  <xsd:sequence>
  <xsd:element name="StartTimeStamp" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />
  <xsd:element name="EndTimeStamp" minOccurs="0" maxOccurs="1" type="xsd:dateTime" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```