Code Walkthrough (Generator)

Before our system is able to process an event and send notifications, information about the event types, subscription types and available communication methods must be inserted in the tables.

**Event Type example entry**

|  |  |
| --- | --- |
| **IO\_EVENT\_TYPE** | |
| **Column Name** | **Data** |
| EVTP\_CODE | 10 |
| EVTP\_DECODE | vesg.insert |
| EVTP\_BTABLE | IO\_VERANSTALTUNG |
| EVTP\_IDBTABLE | VESG\_ID |

**Subscription Type example entry**

|  |  |
| --- | --- |
| **IO\_SUBSCRIPTION\_TYPE** | |
| **Column Name** | **Data** |
| SBTP\_CODE | 6 |
| SBTP\_DECODE | New Courses |
| SBTP\_CONDITIONS | <xmldata>  <data name=”Thema” format=”’%#1#%’” operator=”like” />  <data name=”Ort” format=”#1#” operator=”=” />  </xmldata> |
| SBTP\_FIELDS | <xmldata>  <data name=”Beginn” dbcolumn=“ VESG\_BEGINN“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”BeginnUhrzeit” dbcolumn=“ VESG\_#BEGINN\_ZEIT“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”Ende” dbcolumn=“ VESG\_ENDE“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”EndeUhrzeit” dbcolumn=“ VESG\_#ENDE\_ZEIT“ format=”convert(nvarchar(10),#1#,104)” />  ....  </xmldata> |
| SBTP\_EMAILADR | events@institution.com |
| SBTP\_EMAILSUB | <xmldata>  <fields><field name=“ VESG\_BEGINN”/> <field name=“ VESG\_ENDE”/></fields>  <data subject=“ New Course #VESG\_BEGINN # To #VESG\_ENDE#” />  </xmldata> |
| SBTP\_ISDYNAMICSUB | 1 |
| SBTP\_EMAILREPLYADR | events2@institution.com |
| SBTP\_EMAILCC | events3@institution.com |
| SBTP\_EMAILBCC | events4@institution.com |

This event and this subscription are related to each other, this is described on the IO\_EVENT\_SUBSCRIPTION\_TYPE table.

|  |  |
| --- | --- |
| **IO\_EVENT\_SUBSCRIPTION\_TYPE** | |
| **Column Name** | **Data** |
| EVST\_ID | 1 |
| EVTP\_CODE | 10 |
| SBTP\_CODE | 6 |

**Communication method**

|  |  |
| --- | --- |
| **IO\_COMMUNICATION\_METHOD** | |
| **CMMD\_CODE** | **CMMD\_DECODE** |
| 1 | Email |
| 2 | SMS |

When the needed information is present on these tables the system will be able to work.

After adding this information a subscriber needs to add his information to the system.

**Subscriber Table example entry**

|  |  |
| --- | --- |
| **IO\_SUBSCRIBER** | |
| **Column Name** | **Data** |
| SBER\_ID | 1 |
| USER\_ID | 56 |

**Device Table example entry**

|  |  |
| --- | --- |
| **IO\_DEVICE** | |
| **Column Name** | **Data** |
| DEVC\_ID | 1 |
| DEVC\_ADDRESS | user@email.com |
| SBER\_ID | 1 |
| CMMD\_CODE | 1 |

**Subscription Table example entry**

|  |  |
| --- | --- |
| **IO\_SUBSCRIPTION** | |
| **Column Name** | **Data** |
| SBPN\_ID | 9 |
| SBPN\_PARAMETERS | <xmldata>  <data name=”Thema” dbcolumn=”VESG\_THEMA” value=”Mathematik” />  <data name=”Ort” dbcolumn=”VESG\_VERANSTALTUNGORT value=”Erfurt” />  </xmldata> |
| DEVC\_ID | 1 |
| SBER\_ID | 1 |
| SBTP\_CODE | 6 |

**Notification Text table example entry**

|  |  |
| --- | --- |
| **IO\_NOTIFICATION\_TEXT** | |
| **Column Name** | **Data** |
| NOTX\_ID | 1 |
| SBPN\_CODE | 6 |
| CMMD\_CODE | 1 |
| NOTX\_TEXT | <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  <xsl:template match="/">  <xsl:for-each select="xmldata/data">  Betreff: Anbieterprofil geändert  Werte Mitarbeiter der Akkreditierungsstelle,  die Daten des akkreditierten Anbieters  <xsl:value-of select="@ABIE\_NUMMER" />  wurden vom Anbieter am <xsl:value-of select="@ABIE\_\_x005F\_x0023\_PTS" />  geändert und sind zu prüfen.  </xsl:for-each>  </xsl:template>  </xsl:stylesheet> |

Now when a “vesg.insert” occurs it will check this subscription to see if it needs to generate a notification.

In this example a new “Veranstaltung” is registered. The Thema is “Mathematik-Kurs” and the Ort is “Erfurt” and its VESG\_ID is 102.

The following is inserted into the Notification Event table.

|  |  |
| --- | --- |
| **IO\_NOTIFICATION\_EVENT** | |
| **Column Name** | **Data** |
| NFEV\_ID | 1 |
| NFEV\_EVENTID | 102 |
| NFEV\_INFORMATION | Null (this field is used for when the action handler passes more information than just the ID of the event to the table) |
| NFEV\_ISREAD | False |
| NFEV\_USERID | NULL |
| NFEV\_EMAILCC | event@provider.com |
| NFEV\_EMAILBCC | event2@provider.com |
| NFEV\_ATTACHMENT | <attachments><attachment id=”1”/></attachments> |
| NFEV\_DIRECTINFORMATION | null |
| EVTP\_CODE | 10 |

The Generator service is going to check for changes in this table, so when this information is inserted it will start working.

*ListEvents = Helpers.NotificationEvent.SelectNew();*

In here the Generator selects all the new events from IO\_NOTIFICATION\_EVENTS, in this example only the event “1” is on the table

*While(read ListEvents)*

*{*

The while starts reading all the elements retrieved from the notification events table (in this case it only read event 1).

*if (NotificationEvent.notificationEventDirectInformation != null)*

*{*

*GenerateDirect(NotificationEvent);*

*}*

First the NFEV\_DIRECTINFORMATION field is checked in case the notification is direct (without subscription). In case it is direct, the method GenerateDirect is called, this method uses the stored procedure pss\_GenerateNotification to generate the notification without subscription.

*else*

*If(ListEvents.userId != null)*

*{*

*ListSubscription=Helpers.SusbcriptionInformation.SelectUser(ListEvent.eventTypeCode,ListEvent.userId);*

*Generate(ListEvent,ListSubscription);*

*}*

Then it is going to check if the field NFEV\_USERID of the current event has a value, in this case it is null, but when it has a value, the generator will generate the notification directly for this user instead of checking all the subscriptions.

*Else*

*{*

*ListSubscription=Helpers. SubscriptionInformation.Select(ListEvents.eventTypeCode);*

When there is no user id on the event like in our example, the generator gets all the subscriptions along with all the information needed to process it from the subscription information view.

*While(read ListSubscription)*

*{*

*Generate(ListEvent,ListSubscription);*

*}*

It starts reading all the entries from the subscription information view with a while loop. In this example only the information related to event type 10 is present. Then a method is called to start the generation of the notification for each subscription related to the event.

The following table is the entry read from the V\_SUBSCRIPTION\_INFORMATION view.

|  |  |  |
| --- | --- | --- |
| V\_SUBSCRIPTION\_INFORMATION | | |
| Object Variable | Column Name | Data |
| subscriptionId | SBPN \_ID | 9 |
| subscriptionParameters | SBPN \_PARAMETERS | <xmldata>  <data name=”thema” dbcolumn=”VESG\_THEMA” value=”Mathematik” />  <data name=”Ort” dbcolumn=”VESG\_VERANSTALTUNGORT value=”Erfurt” />  </xmldata> |
| subscriberId | SBER \_ID | 1 |
| userId | USER\_ID | 56 |
| deviceId | DEVC \_ID | 1 |
| deviceAddress | DEVC \_ADDRESS | user@email.com |
| communicationMethodCode | CMMD \_CODE | 1 |
| subscriptionTypeCode | SBTP\_CODE | 6 |
| subscriptionTypeConditions | SBTP\_CONDITIONS | <xmldata>  <data name=”Thema” format=”#1#” operator=”like” />  <data name=”Ort” format=”#1#” operator=”=” />  </xmldata> |
| subscriptionTypeFields | SBTP\_FIELDS | <xmldata>  <data name=”VESG\_BEGINN” dbcolumn=“ VESG\_BEGINN“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”VESG\_#BEGINN\_ZEIT” dbcolumn=“ VESG\_#BEGINN\_ZEIT“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”VESG\_ENDE” dbcolumn=“ VESG\_ENDE“ format=”convert(nvarchar(10),#1#,104)” />  <data name=”VESG\_#ENDE\_ZEIT” dbcolumn=“ VESG\_#ENDE\_ZEIT“ format=”convert(nvarchar(10),#1#,104)” />  </xmldata> |
| subscriptionTypeEmailAddress | SBTP\_EMAILADR | events@institution.com |
| subscriptionTypeEmailSubject | SBTP\_EMAILSUB | New Course Available |
| emailIsDynamicSub | SBTP\_ISDYNAMICSUB | 1 |
| subscriptionTypeEmailReplyAddress | SBTP\_EMAILREPLYADR | events2@institution.com |
| subscriptionTypeEmailCC | SBTP\_EMAILCC | events3@institution.com |
| subscriptionTypeEmailBCC | SBTP\_EMAILBCC | events3@institution.com |
| eventTypeCode | EVTP\_CODE | 10 |
| businessTableName | EVTP\_BTABLE | IO\_VERANSTALTUNG |
| bussinesTableId | EVTP\_IDBTABLE | VESG\_ID |

*Generate()*

*{*

*ParameterList=GetXmlParameters(ListSubscription.subscriptionParameters);*

*ConditionList=GetXmlParameters(ListSubscription.subscriptionTypeConditions);*

*FieldsList=GetXmlParameters(ListSubscription.subscriptionTypeFields);*

When the Generate method is called, it starts reading the information from the view and creates lists from the xml fields by using different methods. This is used so the information becomes easier to handle.

*String fields;*

*Foreach(object in FieldList)*

*{*

*fields+=”,”+ FieldList.format.replace(“#1#”, FieldList.dbcolumn)+” as ”+FieldList.dbcolumn;*

*}*

The fields are read from the list and added to a string. The generator also adds the right format for each field and comas between them.

*String query=”Select ” + fields+ “ from ” + ListSubscription.businessTableName + “ where ”+ ListSubscription.bussinesTableId + “ = ” + ListEvents.eventId;*

A select query is created using the fields from the view, for this select we also use the table name from the view and create a where using the name of the ID column from the business table and the ID number from the event (NFEV\_EVENTID). The query generated for this example is the following:

**Query = “** **Select convert(nvarchar(10), vesg\_beginn ,104), convert(nvarchar(10), vesg\_#beginn\_zeit,104), convert(nvarchar(10), vesg\_ende,104), ….**

**From IO\_VERANSTALTUNG**

**Where VESG\_ID = 102 „**

*Foreach(object in ParameterList)*

*{*

*query+= GetWhere(object,ConditionList);*

*}*

After generating the first part of the query the Generator needs to create the rest of the where statement. It starts by reading each parameter from the parameter list. It compares the “name” attribute of the parameter to the “name” attribute of the conditions by using the “GetWhere” method. This method will return a string that contains the rest of the where clause.

*GetWhere()*

*{*

*foreach (condition in ConditionList)//*

*{*

*if (condition.name == object.name)*

*{*

*String Condition = condition.operator;*

*object.value= condition.format.replace(“#1#”,object.value}*

*}*

*Return “ and ” + object.dbcolumn + “ ” + conditions + “ ” + object.value*

*}*

The “GetWhere” method will scan the list of conditions, it will compare the name of the parameter with the name of the condition like we said before and then it will add the value of each condition (e.g. “=”) and replace the placeholder in the format if this is needed.

The conditions added to the query are the following:

**“and VESG\_THEMA like ‘% Mathematik %’ and VESG\_VERANSTALTUNGORT** **=“Erfurt”**

*Helpers.NotificationGenerator.CallGenerateNotificationProcedure(query);*

The generator calls a procedure that selects the information from the business table using the generated query and transforms it to an xml parameter. This parameter can then be inserted in the notification table. The procedure is used so that the program is able to execute the query as text and the program is able to remain generic by not having the need to describe which business object to use for each different table.

(This would be needed if there was no stored procedure because when trying to open the results from an iBatis query you need to specify explicitly the parameters mapped to each column from the table.)