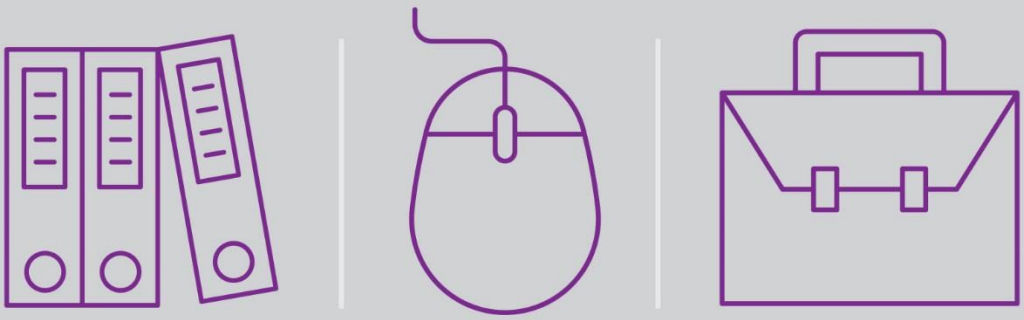


Magic xpi 4.5 with Sugar Connector Seminar

Self-Paced Tutorial

Book ID: UTLSUGMI4

Edition: 1.0, July 2016

Course ID: UCLSUGMI45

Magic University Official Courseware

The information in this manual/document is subject to change without prior notice and does not represent a commitment on the part of Magic Software Enterprises Ltd.
Magic Software Enterprises Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.
The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms and conditions of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.
No part of this manual and/or databases may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or information recording and retrieval systems, for any purpose other than the purchaser's personal use, without the prior express written permission of Magic Software Enterprises Ltd.
All references made to third-party trademarks are for informational purposes only regarding compatibility with the products of Magic Software Enterprises Ltd.
Unless otherwise noted, all names of companies, products, street addresses, and persons contained herein are part of a completely fictitious scenario or scenarios and are designed solely to document the use of Magic xpi.

Magic™ is a trademark of Magic Software Enterprises Ltd.
Btrieve® and Pervasive.SQL® are registered trademarks of Pervasive Software Inc.
IBM®, Topview™, System i5/System i™, pSeries®, xSeries®, RISC System/6000®, DB2®, WebSphere®, Domino®, and Lotus Notes® are trademarks or registered trademarks of IBM Corporation.
Microsoft®, FrontPage®, Windows™, WindowsNT™, ActiveX™, Exchange 2007™, Dynamics® CRM, SharePoint®, Excel®, and Word® are trademarks or registered trademarks of Microsoft Corporation.
Oracle®, JD Edwards EnterpriseOne®, JD Edwards World®, and OC4J® are registered trademarks of the Oracle Corporation and/or its affiliates.
Google Calendar™ and Google Docs™ are trademarks of Google Inc.
Salesforce® is a registered trademark of salesforce.com Inc.
SAP® Business One and SAP® R/3® are registered trademarks of SAP AG in Germany and in several other countries.
SugarCRM is a trademark of SugarCRM in the United States, the European Union and other countries.
Linux® is a registered trademark of Linus Torvalds.
UNIX® is a registered trademark of UNIX System Laboratories.
GLOBEtrotter® and FLEXlm® are registered trademarks of Macrovision Corporation.
Solaris™ and Sun ONE™ are trademarks of Sun Microsystems Inc.
HP-UX® is a registered trademark of the Hewlett-Packard Company.
Red Hat® is a registered trademark of Red Hat Inc.
WebLogic® is a registered trademark of BEA Systems.
Interstage® is a registered trademark of the Fujitsu Software Corporation.
JBoss™ is a trademark of JBoss Inc.
GigaSpaces, GigaSpaces eXtreme Application Platform (XAP), GigaSpaces eXtreme Application Platform Enterprise Data Grid (XAP EDG), GigaSpaces Enterprise Application Grid, GigaSpaces Platform, and GigaSpaces, are trademarks or registered trademarks of GigaSpaces Technologies.

Clip art images copyright by Presentation Task Force®, a registered trademark of New Vision Technologies Inc.
This product uses the FreeImage open source image library. See http://freeimage.sourceforge.net for details
This product uses icons created by Axialis IconWorkShop™ (http://www.axialis.com/free/icons)
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).
This product includes software developed by Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/).
Copyright © 1989, 1991, 1992, 2001 Carnegie Mellon University. All rights reserved.
This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).
This product includes software that is Copyright © 1998, 1999, 2000 of the Thai Open Source Software Center Ltd. and Clark Cooper.
This product includes software that is Copyright © 2001-2002 of Networks Associates Technology Inc All rights reserved.
This product includes software that is Copyright © 2001-2002 of Cambridge Broadband Ltd. All rights reserved.
This product includes software that is Copyright © 1999-2001 of The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved.

All other product names are trademarks or registered trademarks of their respective holders.

Magic xpi 4.5 with Sugar Connector Seminar

July 2016

Copyright © 2013-2016 by Magic Software Enterprises Ltd. All rights reserved.

Table of Contents

Introduction

Welcome to Magic Software University's **Magic xpi 4.5 with Sugar Connector Seminar**.

We, at Magic Software University, hope that you will find this tutorial informative and that it will assist you in getting started with this exciting product.

About the Seminar

The seminar is intended for people with a knowledge of SugarCRM who want to know how to successfully use Magic Software Enterprises' Magic xpi product, and how to integrate Magic xpi with SugarCRM.

During the seminar you will learn about the Magic xpi Sugar connector and how Magic xpi integrates with SugarCRM.

Topics that will be covered include:

- Queries, including advanced queries
- Referencing objects by IDs
- Creating a SugarCRM Quote scenario
- Relationships between objects using the Link method
- Capturing events

Course Prerequisites

Before you start with the course there is basic knowledge that you need to have:

Development knowledge	Familiar with Magic xpi 4.5, Magic xpi 4.1 or iBOLT/Magic xpi 3.x
SugarCRM	Knowledge of SugarCRM

Your computer must also meet some basic requirements:

Hardware	• Windows XP Pro and later. The course was tested on Windows 7 • Pentium processor 1.8GHz and upwards • 4Gb RAM or greater • At least 1Gb of free space • Screen resolution of at least 1024x768 pixels
Magic xpi	You will need to install Magic xpi V4.5
License	For deployment purposes, you need the SUGCRM license from your Magic Software Enterprises representative. This is not required for development purposes.
SugarCRM	This seminar has been designed using the SugarCRM installation with the **Populate Database with Demo Data** option set to **Yes**. The demonstration data is based on Version 7.5.0.1 of SugarCRM (which uses their v10 API). If you use a different version you will need to provide your own sample data files.
Email Server	You need access to an email server so that you can send emails. You can use your Gmail or Yahoo accounts as well. Check the internet for instructions on how to configure those mail servers for POP3, IMAP and SMTP.

How to Use This Guide

The self-paced guide provides detailed step-by-step instructions. If you are learning using this self-pace tutorial, feel free to contact your Magic Software Enterprises representative or the Support department for further assistance.

Lesson 1

Sugar Connector

The Magic xpi Sugar connector enables a work flow between Magic xpi and SugarCRM.

Using the Sugar connector, you can add, modify, and delete objects in SugarCRM.

You can also trigger a Magic xpi flow when actions such as add, update, or delete are performed in SugarCRM.

As was mentioned in the Prerequisites section, for deployment purposes, to work with the Sugar connector, you need a special Magic xpi license: SUGCRM.

This lesson covers various topics including:

- An introduction to the Sugar connector
- Installing the newest version of the Sugar connector
- Creating a SugarCRM resource
- Connecting Magic xpi to SugarCRM

Magic xpi Architecture with the Sugar Connector

The Magic xpi Sugar connector works with SugarCRM's REST API. The XML interface enables flexibility when working with SugarCRM, since you do not need to frequently update the Sugar connector.

The Sugar connector can create, query, update, and delete data objects in SugarCRM using the REST API.

The Magic xpi Sugar connector supports both the XML interface and Method interface.

The Sugar connector has the following methods:

- Create Product Bundles
- Document Add Revision
- Get Server Info
- Get User ID
- Link
- Note Add Attachment

Connecting to SugarCRM

The Sugar connector needs to be connected to a specific user in SugarCRM.

Therefore, before working with the Magic xpi Sugar connector, you need:

- A valid SugarCRM user name
- A valid SugarCRM password

Installation

Magic xpi 4.5 supports SugarCRM version 6.4x, which works with the v4_1 REST and SugarCRM version 7.x, which works with the v10 API.

This course uses SugarCRM Enterprise, Version 7.5.0.1.
The Sugar Community Edition works with 6.4x, which will not be demonstrated in this course. However, most of the functionality is similar.

As mentioned in the prerequisites, you should already have Magic xpi 4.5 installed on your computer.

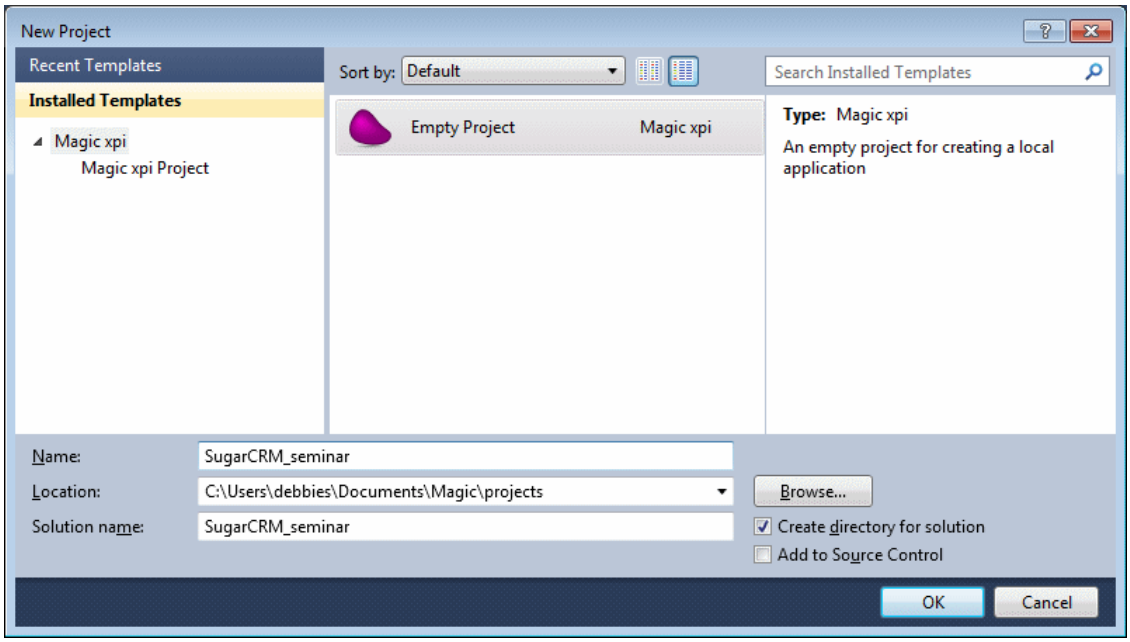
You're now ready to start developing.

Creating a Project

As with any development project, the first step is to create a new Magic xpi project.

To create a new Magic xpi project:

1. Open Magic xpi.
2. Click on the **File** menu, and select **New**. The **New Project** dialog box will open.
3. Create a new project called **SugarCRM_seminar**.



For the purpose of this course, data has been prepared for you.

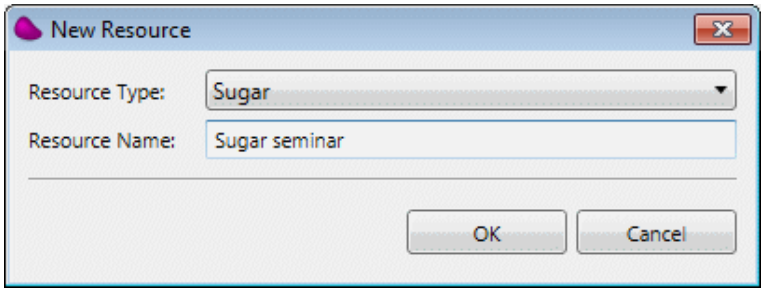
4. Copy the **course_data** folder into the **SugarCRM_seminar\SugarCRM_seminar** folder. This folder and subfolder were created when you created the new project.

A final version of the project is provided in the **Final_SugarCRM_Project** folder, which you can copy to the **projects** folder and refer to if needed.

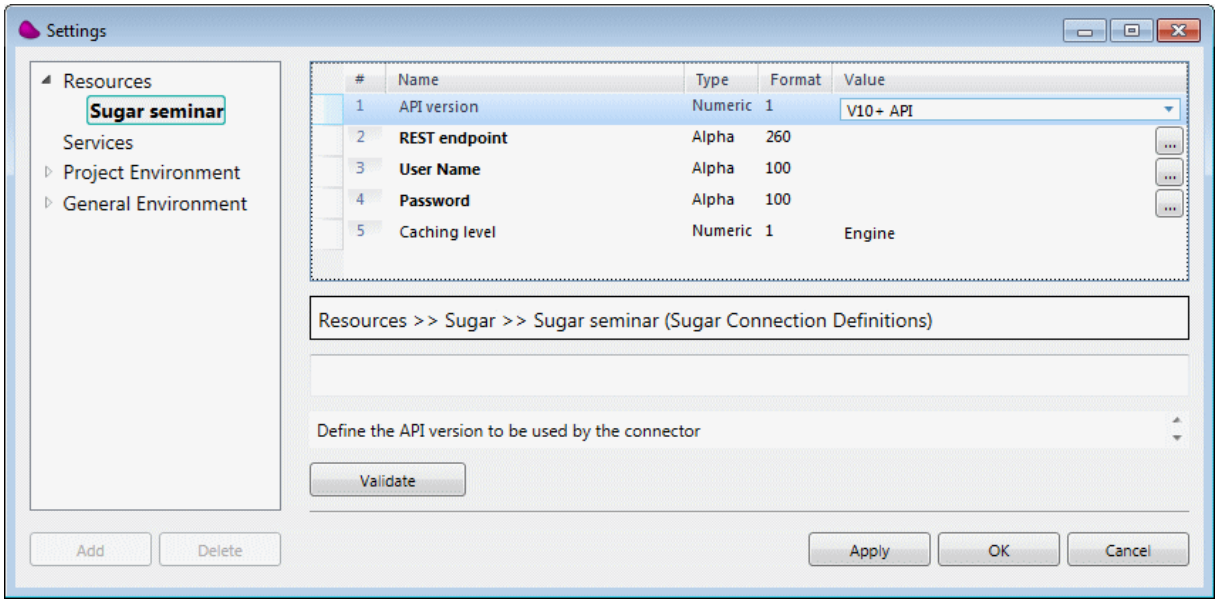
Defining a Resource

Before using the Sugar connector in a step, you need to define the **Sugar** resource.

1. From the **Project** menu, select **Settings**.
2. While parked on the **Resources** option, click **Add** to add a resource.
3. From the **Resource Type** field, select **Sugar**.
4. Name the resource: **Sugar seminar**.



There are three mandatory settings to be defined in the Sugar resource. These are the settings that appear in bold.



5. From the **API version** setting, select **V10+ API**. If you are using the free version, you'll use the **Legacy API** option.
6. In the REST endpoint setting, enter the URL for the V10 API, which is: **http://{site url}/rest/v10/**. In the image above, you can see that the site url that was used was: **sugarcrmsrv/sugar**. Therefore, the full URL entered was: **http://sugarcrmsrv/sugar7/rest/v10/**.

For SugarCRM v4.1, including the Sugar Community Edition, the syntax is: **http://{site url}/service/v4_1/rest.php**.

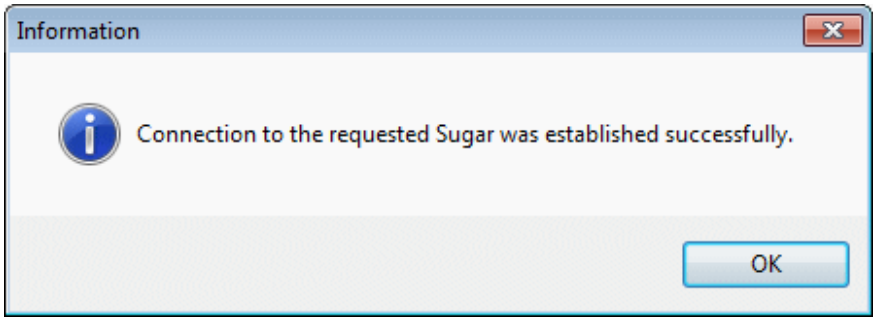
7. Enter your **User Name** and **Password** for the SugarCRM server.

You can leave the **Caching level** as is.

The basic assumption behind the need for caching is that the same data will be needed more than once. Therefore, if data can be re-fetched without performing disk I/O operations, overall performance will be enhanced. This optional setting has two options:

- **Context**: Login and module metadata is cached throughout the context. The login context is created by the first SugarCRM step in the flow and used by all other SugarCRM steps in the context configured with the same resource.
- **Engine** (default): Login and module metadata is cached for all contexts running under a Magic xpi engine. Login occurs only once for all contexts, until the login is no longer valid. This option enhances performance and lowers the use of SugarCRM API calls.

8. Click the **Validate** button to check your connection. If all of the settings were entered correctly, you should see the following message:



You have now successfully created a connection from Magic xpi to SugarCRM.

Summary

In this lesson:

- You were introduced to the Magic xpi Sugar connector.
- You installed the newest version of the Sugar connector.
- You created a project for the seminar.
- You created a SugarCRM resource and connected Magic xpi to SugarCRM.

Lesson 2

Querying SugarCRM via Magic xpi

A Sugar Query operation is used to retrieve data from an object according to specific search criteria.

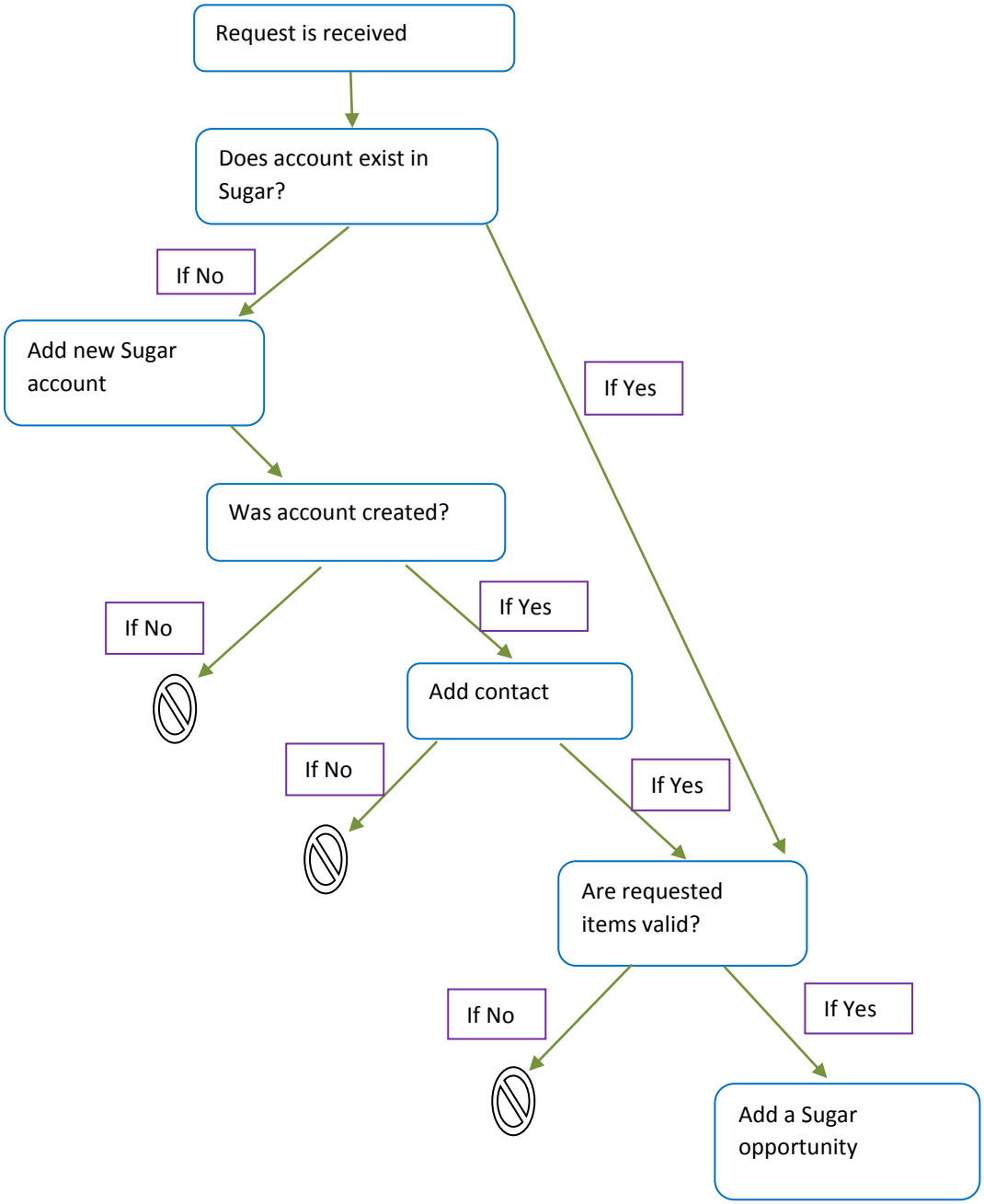
You can also define advanced WHERE clauses using the Data Mapper.

This lesson covers various topics including:

- Query operation
- WHERE clauses
- Filters

Preview of the Flow

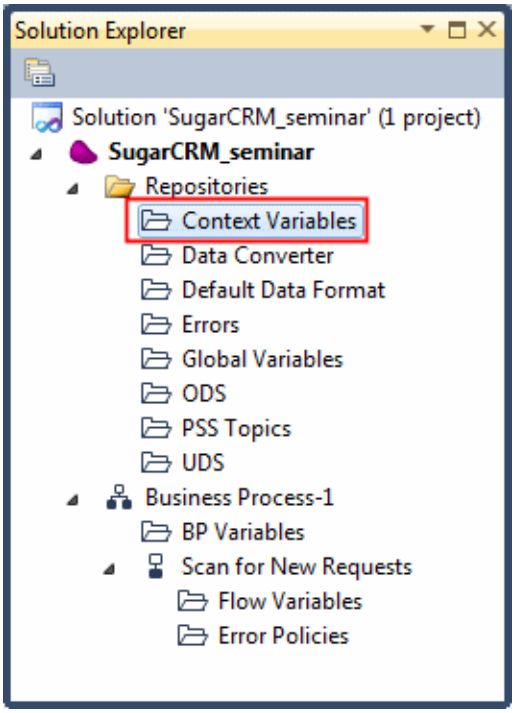
The business process logic of the Magic xpi flow that you will create is as follows:



Triggering the Flow

You'll use a trigger to activate the flow.

1. Rename the default flow and name it **Scan for New Requests**.
2. From the **Solution Explorer**, double-click on the **Context Variables** folder.



3. From the **Context Variables** tab, click **Add** and add the following variable:
   - **C.RequestXML, a BLOB variable**
4. Click the  **Save** button.
5. From the Solution Explorer, double-click on the **Flow Variables** folder (under the new **Scan for New Requests** flow).

6. From the **Flow Variables** tab, click **Add** and add the flow variables listed below. When you are asked to use these variables, an explanation about them will be provided.
   - **F.Account**, a BLOB variable
   - **F.RequestFileName**, an Alpha variable of size 255
   - **F.ContactXML**, a BLOB variable
   - **F.AccountExists**, a Logical variable with a default value of 'FALSE'LOG. The **F.AccountExists** variable will be used if a query returned a user record from SugarCRM.
   - **F.AccountId**, an Alpha variable of size 100
7. Click the  **Save** button.

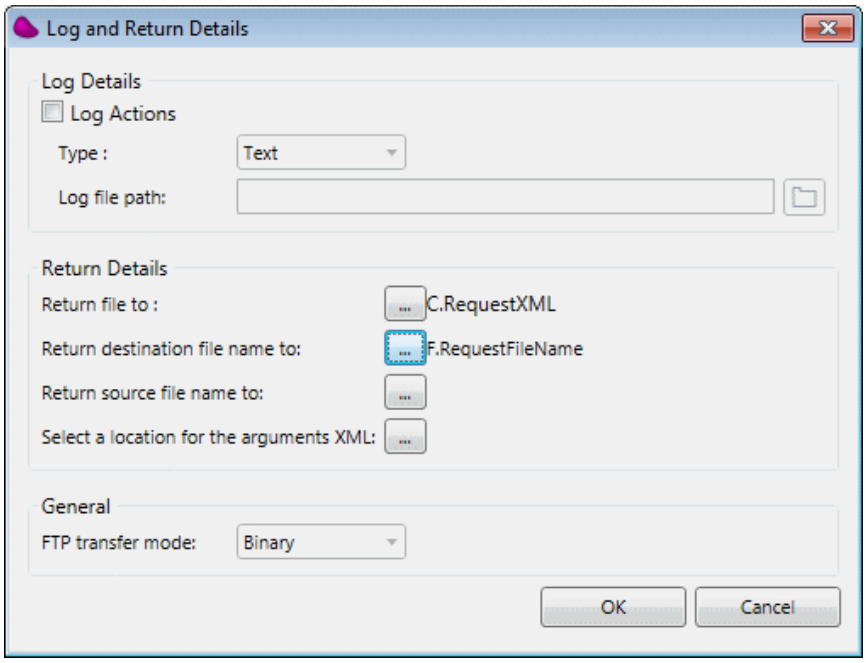
IDs in SugarCRM are long. For example:

c570b261-42da-3240-fe8e-557e48e862df

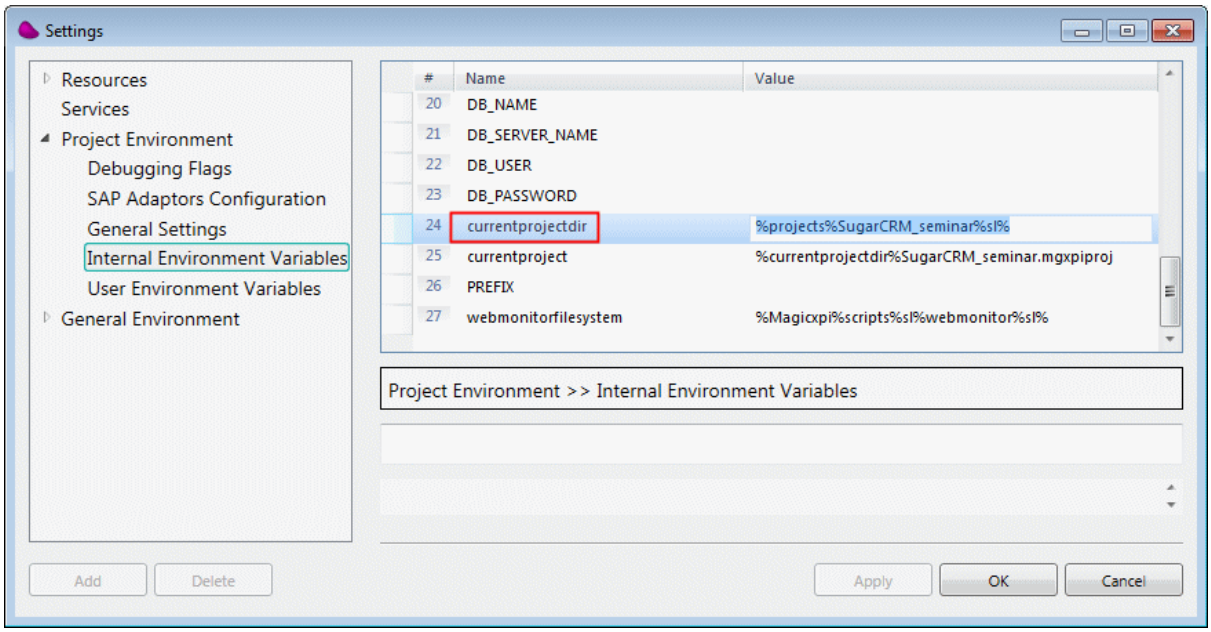
Therefore, the ID variables used in this course are set to 100.

You will receive the request using the Directory Scanner component. There are two ways of using the Directory Scanner component: Trigger or Step. In this example you will use the Trigger mode.

1. From the **Toolbox** (in the **Triggers** section), drag a **Directory Scanner** component to the Trigger area.
2. In the trigger's **Properties** pane, set the **Trigger Name** property to: **Wait for File**.
3. Right-click on the component and select **Configuration**. The **Component Configuration: Directory Scanner** dialog box opens.
4. Click **New** to define the trigger.
5. Define the following:
   a. Leave the **Source** as **LAN**.
   b. Set the **Directory** to: **EnvVal ('currentprojectdir')&'course_data\in\'**. The **currentprojectdir** environment variable contains the path to the directory where the current project resides.
   c. In the **Filter** property, leave the default of ***.***
   d. Leave the **Action** as **Move**.
   e. Set the destination **Directory** to **EnvVal ('currentprojectdir')&'course_data\out\'**.
6. Click the **Advanced** button.
   a. Set the **Return file to** property to **C.RequestXML**. This is the variable that the Directory Scanner will return the content of the file to.
   b. Set the **Return destination file name to** property to **F.RequestFileName**. This is the name of the variable that the Directory Scanner will return the name of the file to.



7. Click **OK**.
8. From the **Project** menu, select **Settings**.
9. Go to the **Project Environment** section and click the **Internal Environment Variables** option.
10. Check that the **currentprojectdir** environment variable is pointing to the correct location. We used this environment variable in the Directory Scanner component, so we need to check this environment variable so that the trigger will know where to take the files from.



You have finished defining the trigger.

Query Operation

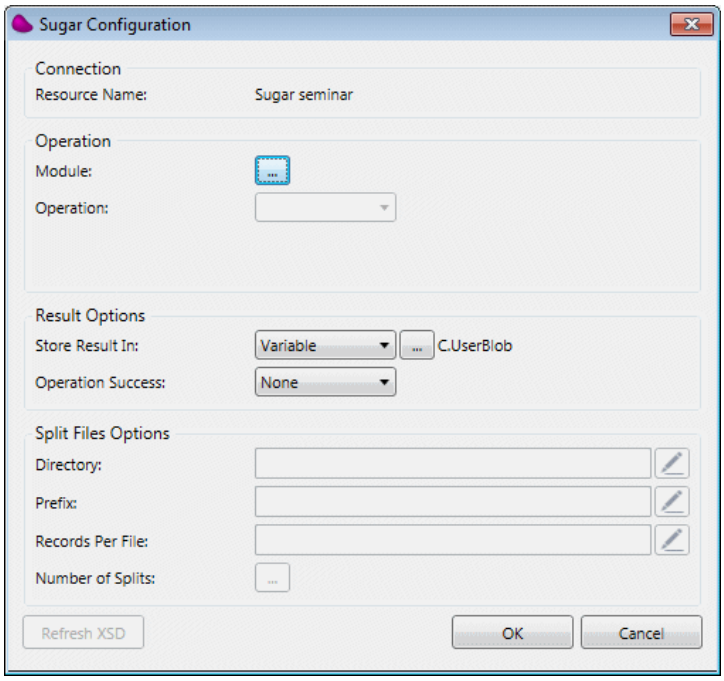
You will use the XML interface to check whether the customer exists in SugarCRM.

You will use the Sugar connector to check whether the customer exists as an account in SugarCRM.

An account in SugarCRM represents a single company.

1. Drag a Sugar connector  as the first step in the **Scan for New Requests** flow.
2. Set the Step Name property to Check for Account.
3. Leave the **Interface** as **XML**.
4. In the **Setting** section of the **Properties** pane, check that the **Resource Name** property is set to the **Sugar seminar** resource. Since this is, currently, the only SugarCRM resource, it was probably selected automatically by Magic xpi.
5. Right-click on the step and select **Configuration** or just double click on the step.

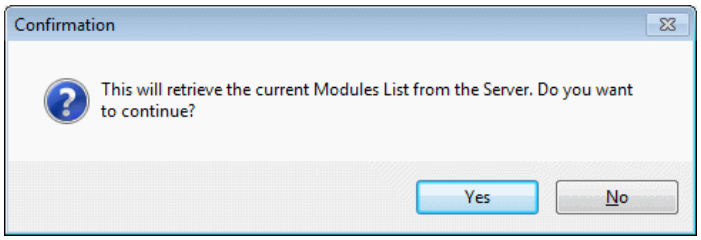
The **Sugar Configuration** dialog box enables you to perform operations on a SugarCRM object.



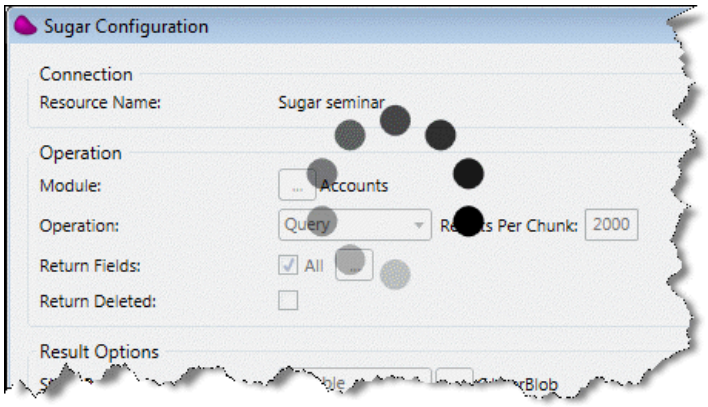
6. From the **Module** field, click the selection button  .

Magic xpi needs to fetch the objects exposed by the SugarCRM API before accessing them. Magic xpi connects directly to the SugarCRM server, retrieves the available modules, and displays them in a list. The **Modules List** contains all of the modules from the SugarCRM server.

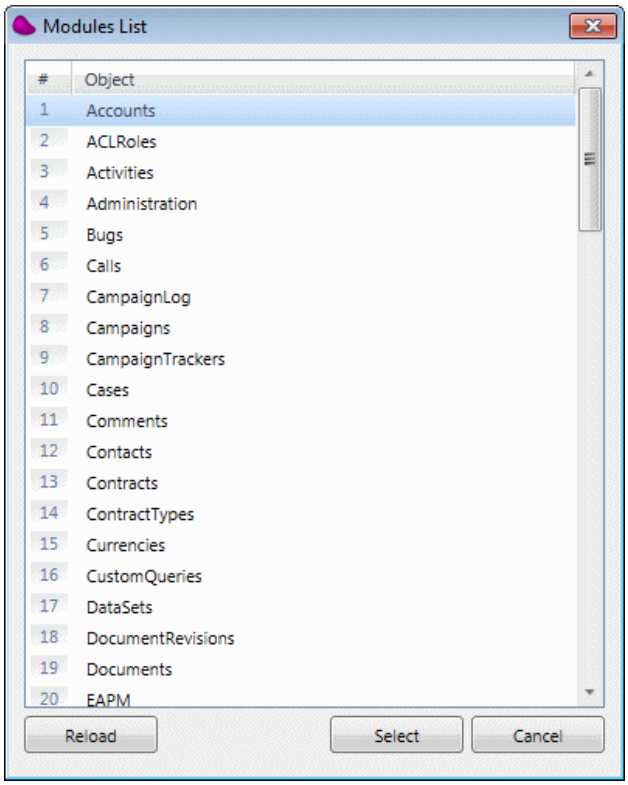
7. To retrieve the latest Module List from the SugarCRM server, click the **Reload** button. The following message will appear.



8. Click **OK.** The following image will appear, showing that Magic xpi is retrieving data from the SugarCRM server.



9. From the **Modules List**, select **Accounts**.



The Sugar connector supports CRUD operations: Create, Query, Update and Delete. Now since you're simply just browsing to see if an account exists in SugarCRM, you'll perform a Query.

10. From the **Operation** field, select **Query**.
11. The **Results Per Chunk** field is the maximum number of results that you want to fetch in each call to SugarCRM. Leave it set to **2000**.
12. The **Return Fields** option enables you to define which fields will be returned in the result XML.
13. Clear the **Return Fields** check box and select **id** and **name**. In general, this is recommended, because it reduces your result set, the size of the fields that are returned.
14. The **Store result in** field will hold the XML retrieved from SugarCRM. You can select either a file or a variable. Select **Variable**, and then select the **F.Account** variable that you defined earlier.
15. It's a good idea to click the **Refresh XSD** button to make sure that you're using the latest module metadata. Changes that are made in the SugarCRM environment – customizations and so forth – are all pulled into the integration environment so that when you do the data mapping, it's all there and available to you.
16. Click **OK**.

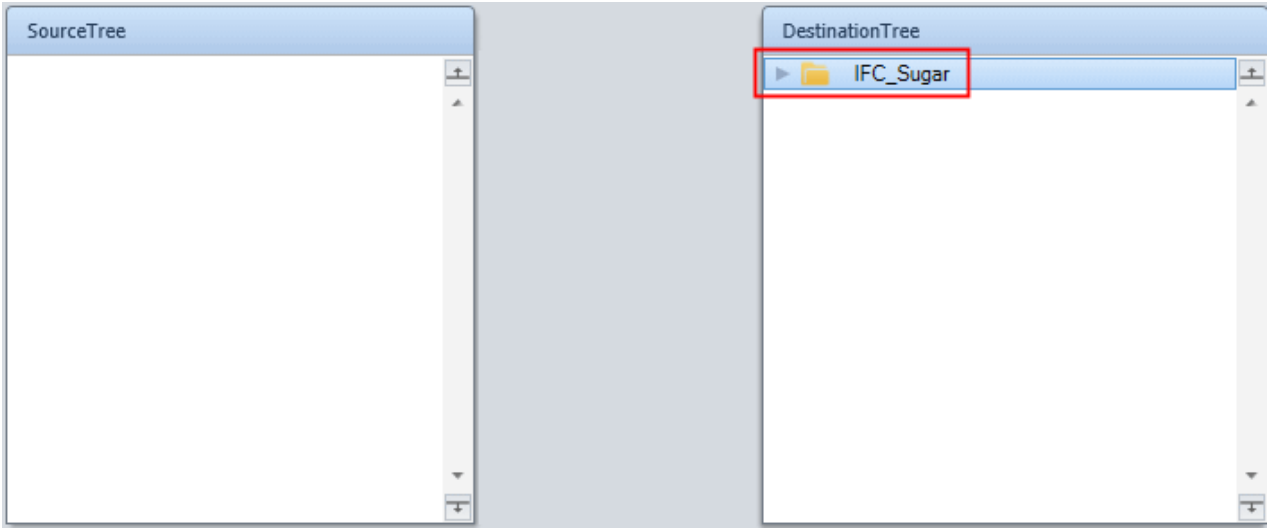
The Magic xpi Sugar connector saves the XML Schema, the XSD, in the following directory:

[project dir]\[project name]\[project name]\SugarCRM\XSD\[resource name]

In our case, this is:
projects\SugarCRM_seminar\SugarCRM_seminar\SugarCRM\XSD\Sugar seminar\Accounts.xsd

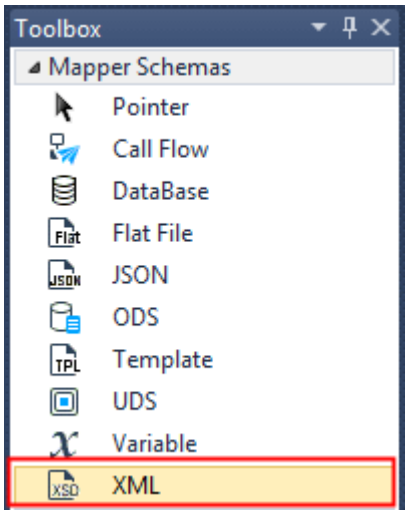
As you are currently using the Sugar connector with the XML interface, you will use the Data Mapper to configure it.

After defining the properties for the Sugar connector, a new **IFC Model** entry was created in the **Destination** section: **IFC_SugarCRM**.

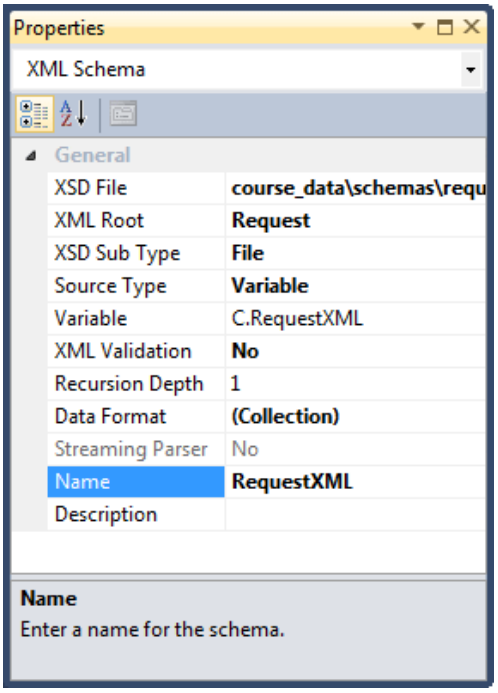


You need to use the request XML that was retrieved by the Directory Scanner to check whether the account exists in SugarCRM. Therefore, you need to have XML as the source.

1. From the **Toolbox**, drag an **XML** entry to the Source pane of the Data Mapper.



2. Go to the **Properties** pane.
   a. Set the **Name** property to **RequestXML**.
   b. In the **XSD File** property, select the following schema: **course_data\schemas\request.xsd**
   c. Set the **Source Type** to **Variable** and select the **C.RequestXML** variable.



3. Click the save icon.

The next stage is to map.

You need to send the customer name to SugarCRM to query its existence. Therefore, in the destination, you should use the **name** node in the SugarCRM **Accounts** module that you previously configured.

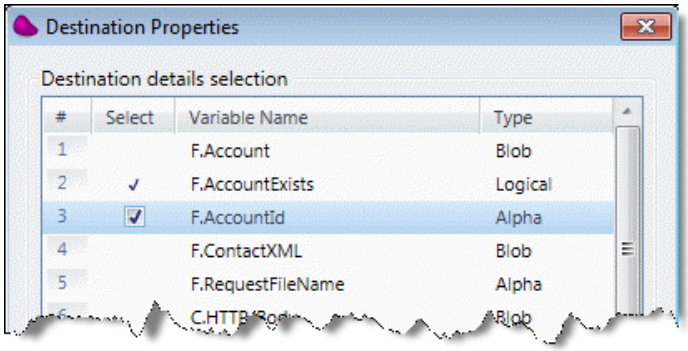
To expand all of the nodes, park on the top node of the Source or Destination side, right-click and select **Expand all**.

4. In the Data Mapper's **Source** pane, expand the **Request > CustomerDetail** node.
5. In the **Destination** pane, expand the **Accounts > row > Fields** node.
6. While standing on the **Fields** node, press the letter **n** to get to the **name** node.
7. Connect the **AccountName** node to the **name** node.
8. Click the save icon and return to the Flow area.

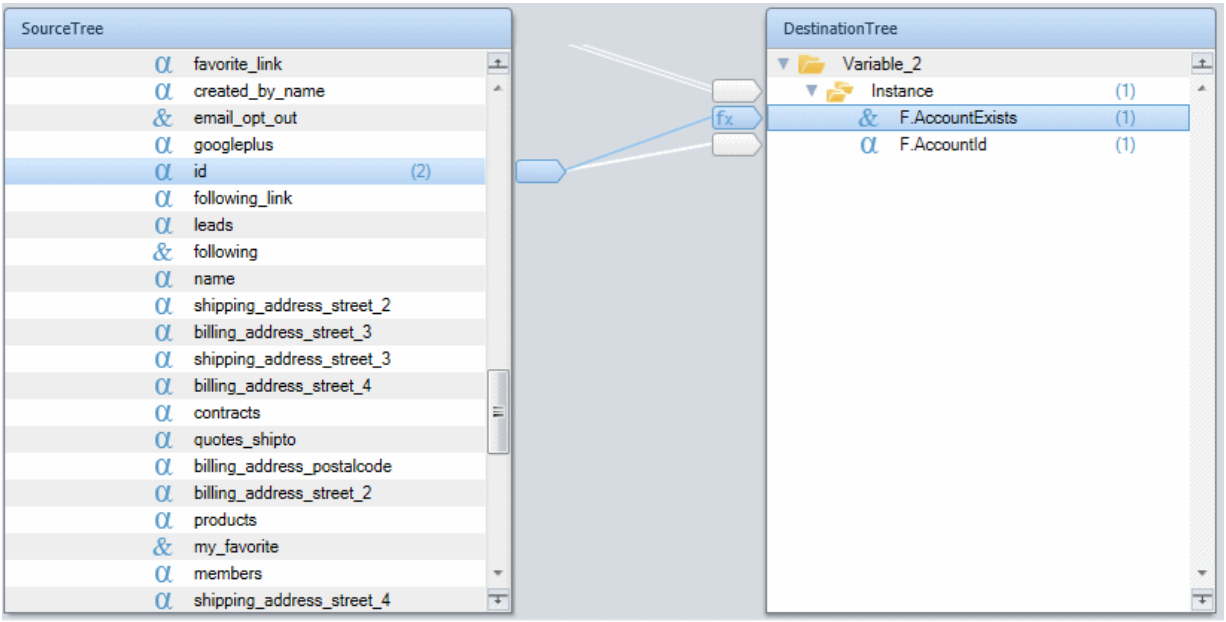
Another way to search is to press **Ctrl+Shift+F** from one of the panes or go to the **Edit** menu, select **Find** and then **Find Text**. In the **Find Text** dialog box, you can narrow your search. Once the cursor is on one of the items that meets your search criteria, you can press **F3** to go to the next item.

Check If Account Exists

1. Drag a Data Mapper component as a child of the **Check for Account** step.
2. Set the **Step Name** property to the following: **Check If Account Exists**.
3. Right-click on the step and select **Configuration** or double-click on the step.
4. Drag an **XML** entry onto the **Source** pane of the Data Mapper.
5. In the **XSD File** property, select the following schema: **SugarCRM\XSD\Sugar seminar\Accounts.xsd**
6. Set the **Source Type** to **Variable** and select the **F.Account** variable.
7. Drag a **Variable** entry onto the **Destination** pane.
8. In the **Properties** pane, go to the **Variables** property and click the selection button  .
9. Select both the **F.AcountExists** and **F.AccountId** variables.



10. On the Data Mapper's **Source** pane, open the following node: **Accounts > row > Fields**.
11. On the **Destination** pane, open the **Instance** node.
12. Connect the **id** node to the **F.AccountExists** node.



13. While the cursor is on the **F.AccountExists** node go to the **Calculated value** property and enter the following expression:

    NOT (Src.S1/Accounts/row/Fields/id =''OR ISNULL (Src.S1/Accounts/row/Fields/id))

In the expression above, the path **Src.S1/Accounts/row/Fields/id** is entered by clicking the **Source Nodes**  icon at the top of the Expression Editor.

This expression returns True if there is a value. This means that if the **id** field is not empty or null, then the account exists.

14. Also connect **id** to **F.AccountId**. This will update the **F.AccountId** variable with the value of the SugarCRM **id** fields. This connection will be used in a later step when an opportunity is created.
15. Save and return to the Flow area.

Testing Your Project

You will want to test your flow to make sure it works.

1. Right-click on the **Check for Account** step and select **Breakpoint**. A red dot will appear next to the step. A breakpoint means that processing will halt at that point.
2. Copy the **Non existing account.xml** file from the **out** folder to the **in** folder. This XML file includes a non-existing account named **Magic Hotels**.

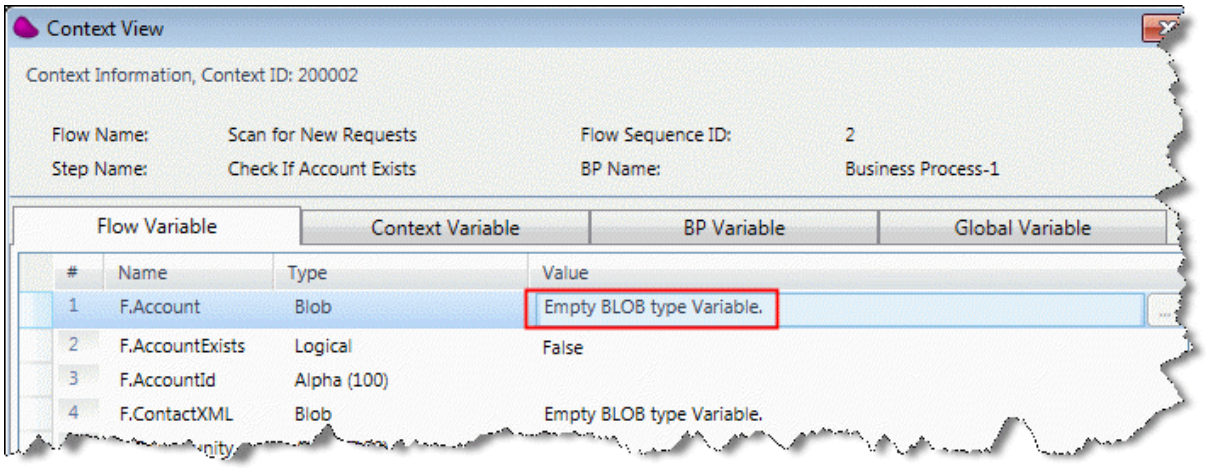
3. From the toolbar, click the Start Debugging icon (or from the **Debug xpi** menu, select **Start Debugging**). You can also press **F5** to start the Debugger. Magic xpi checks the project for any syntax errors. If there are syntax errors, you will not be able to continue. There are various types of syntax errors, such as a mandatory property that was not defined or was incorrectly defined.

When the breakpoint is reached (which can sometimes take a few seconds), the **Toolbox** will become the **Context Tree**.

4. From the Context Tree, right-click the **Check for Account** option and select **Step**. This will run the second step.

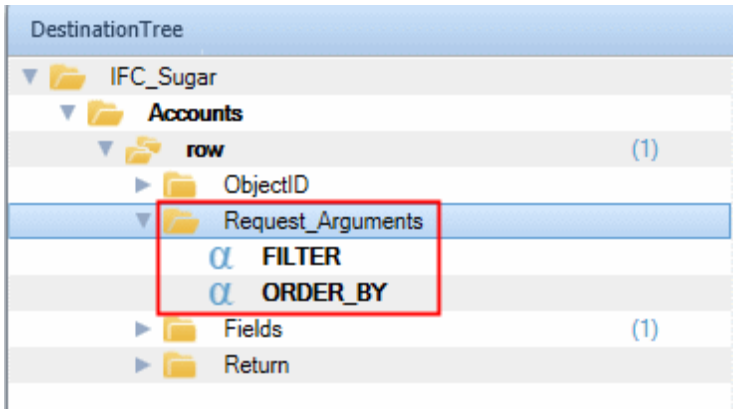
You now want to look at the **F.Account** variable, which is the variable that you selected in the **Store result in** field.

5. From the Context Tree, right-click on the **Check If Account Exists** option and select **Context View** (or select it from the **Debug xpi** menu).
6. Find the **F.Account** variable and notice that it says **Empty BLOB type Variable**. This is because the account does not exist.



7. Click **Close** and then **OK**.
8. Go to the **Debug xpi** menu and select **Stop Debugging** to go back to development mode.

WHERE Clauses



In the v10 API, you can also use a more advanced query.

If you open the **Check for Account** step's Data Mapper, you'll see the **Request_Arguments** compound and its **FILTER** and **ORDER_BY** elements. These are what you use for the advanced queries. Unlike the Legacy API, the query is not a direct SQL statement that goes as-is to the database.

FILTER

For example, to query for data where the last_name='Smith' and the first_name='John', you would use this FILTER clause:

filter[0][first_name][$starts]=John&filter[0][last_name]=Smith

The **[first_name]** element is a field name but **[$starts]** is a reserved keyword, which is part of SugarCRM filter syntax.

In the legacy version, the WHERE clause is sent directly to the database. In the **Accounts > SQL > WHERE** node, you use the following syntax: **<DB_TableName>.<FieldName>**

For example, if you want to query the data with last_name='Smith' and first_name='John', you would use the following syntax:

Contacts.last_name='Smith'AND Contacts.first_name='John'

Magic xpi passes the filters as is to SugarCRM, so you can use any of their supported filter operations, which are as follows.

Filter	Description
$equals	Performs an exact match on that field.
$not_equals	Matches on non-matching values.
$starts	Matches on anything that starts with the value.
$in	Finds anything where field matches one of the values as specified as an array.
$not_in	Finds anything where field does not matches any of the values as specified as an array.
$is_null	Checks if the field is null. This operation does not need a value specified.
$not_null	Checks if the field is not null. This operation does not need a value specified.
$lt	Matches when the field is less than the value.
$lte	Matches when the field is less than or equal to the value.
$gt	Matches when the field is greater than the value.
$gte	Matches when the field is greater than or equal to the value.

This table was replicated from: http://developer.sugarcrm.com/2014/02/28/sugarcrm-cookbook1/.

ORDER_BY

As mentioned above, you can also query using the **ORDER_BY** element in the **Request Arguments** node.

For example, enter **'name:desc'** in the **Calculated value** property of the **ORDER_BY** element. All of the records will be sorted in descending order of their name field for the selected module and displayed in the return variable.

Exercise

During this lesson, you created a flow named **Scan for New Requests**. The purpose of this flow is to scan the **in** folder to see if a new XML request is there. If Magic xpi found a request in the folder, then you were asked to check whether the customer exists in SugarCRM.

If the customer exists, then:

- Check whether the items requested in the XML file are valid SugarCRM products.

Hints:

- Refer back to the **Preview of the Flow** section on page 14.
- Use the **ProductTemplates** module.

Once you have tried this on your own, please make sure to look at the solution for this exercise on page 63. The following lessons build on the exercise.

Summary

In this lesson:

- You learned about the Query operation.
- You also learned about WHERE clauses and filters.
- You used the Sugar connector to query the **Accounts** module to check the existence of an account.

Lesson 3

Adding an Object

In the previous lessons you learned how to fetch information from SugarCRM.

Querying a database is not the only operation needed in a project. It is often necessary to add an object to the database.

In this lesson, you'll see how Magic xpi enables you to add an entry to the SugarCRM database.

You'll also learn about using entries in SugarCRM selection lists.

Adding an Object to SugarCRM

The steps needed to add an object are very similar to the steps required to query an object.

Now, you'll add a customer if the customer does not exist. In other words, if the **Check If Account Exists** step returns false, you'll add the customer to SugarCRM.

1. Open the **Scan for new requests** flow.
2. Add a Sugar connector as a child of the **Check If Account Exists** step and name it **Add Account**.
3. Double click on the step.
4. From the **Module** property, select the **Accounts** module.
5. Set the **Operation** field to **Create**.
6. From the **New Object ID** property, select **F.AccountId**. When an object is added, SugarCRM returns the object ID of the newly created object into this variable.

SugarCRM returns the ID of the last object created. If your step is adding or updating multiple records or objects, make sure to take the IDs from the result XML.

7. In the **Store Result In** field, select the **F.Account** variable.
8. Click **OK**.

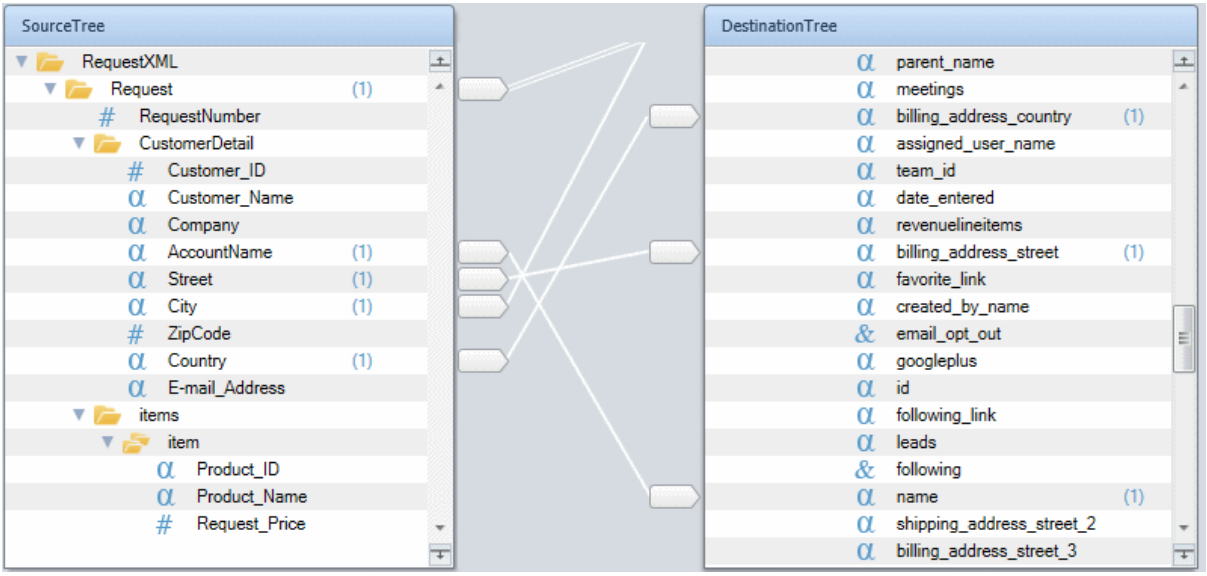
You need to use the request XML that was retrieved by the Directory Scanner. This contains the customer information. Therefore, you need to have an XML as the source.

1. Add an **XML** source and name it **RequestXML**.
2. In the **XSD File** property, select the following schema: **course_data\schemas\request.xsd**
3. From the **Variable** property, select the **C.RequestXML** variable.

You are now ready to map.

1. On the Source pane, open the following node: **Request > CustomerDetail**.
2. On the Destination pane, open **Accounts > row > Fields**.
3. Connect the following nodes:

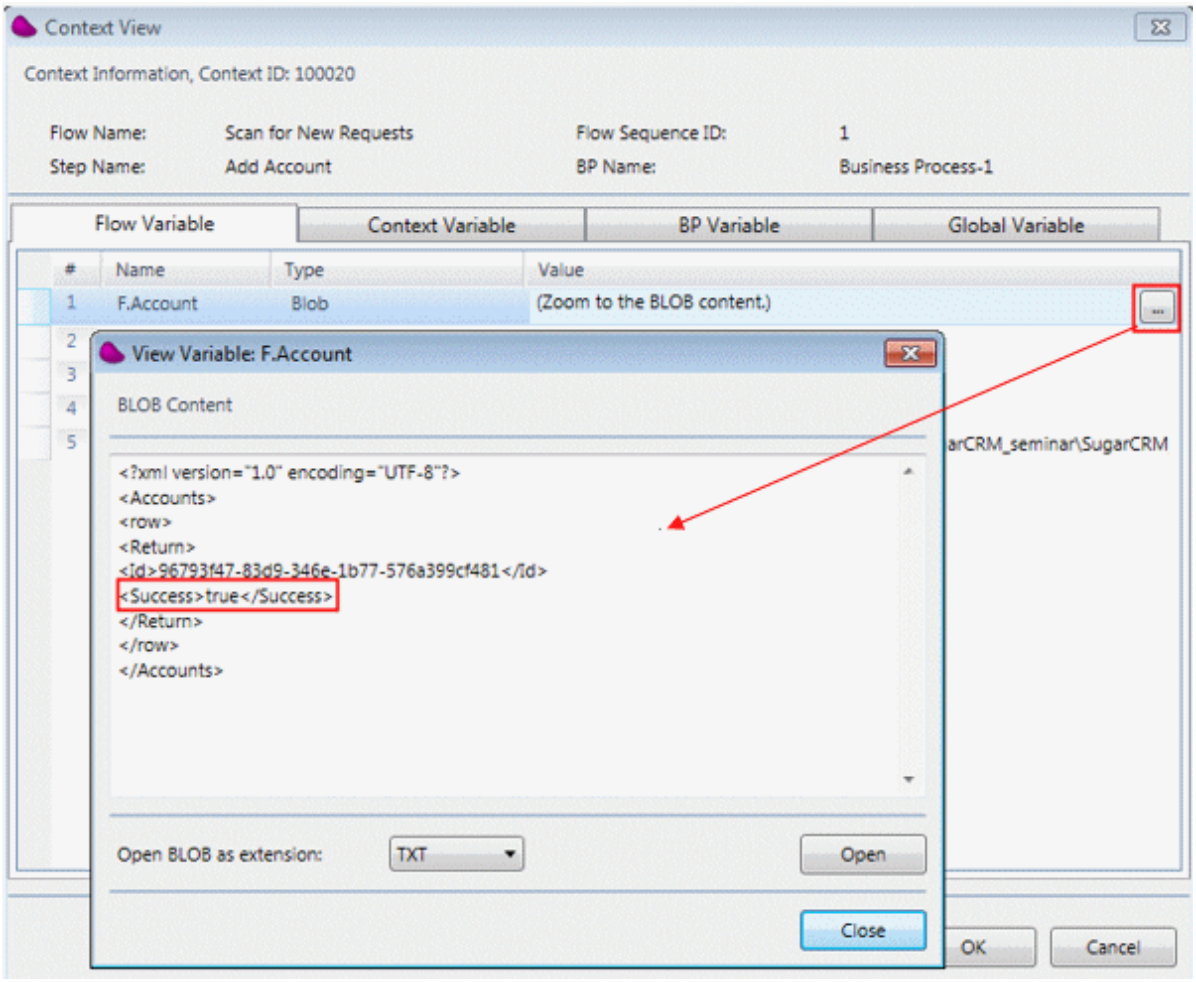
Source node	Destination node
AccountName	name
Street	billing_address_street
City	billing_address_city
Country	billing_address_country



You only want this step to be executed if the customer does not exist; in other words, the **Check If Account Exists** step's result was unsuccessful.

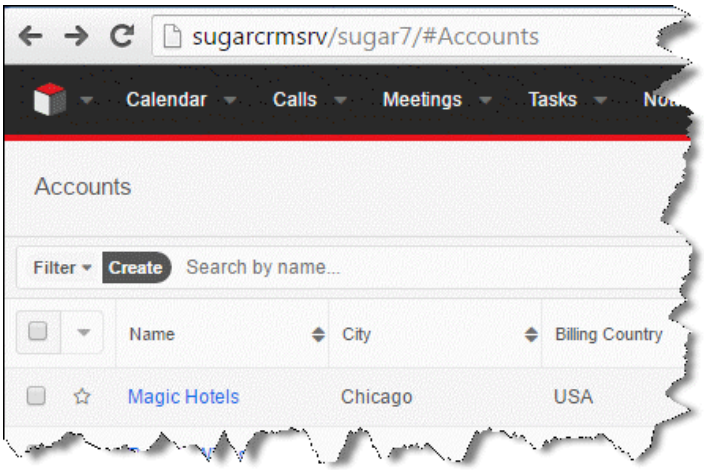
1. Park on the **Add Account** step.
2. Right-click and set the following condition: **NOT (F.AccountExists)**.
3. Copy the **Non existing account.xml** file from the **out** folder to the **in** folder. This file includes a non-existing account.
4. Remove any breakpoints and add a breakpoint to the **Add Account** step.
5. Run the Debugger on the project. The result for the **Create** operation is stored in the **Store result in** variable, which in the **Add Account** step is the **F.Account** variable.
6. When the Debugger reaches the **Add Account** step, open the **Context View**.
7. Zoom from the **F.Account** variable and you can see the content of the variable. For every **Create** operation, the returned XML contains a success or failure indication. In

the image below, you can see that the step was successful.



If there is an error, you will see the error in the returned XML.

8. In addition, open SugarCRM and if the process worked correctly, you should see the new account in SugarCRM.



Now you'll want, as part of the flow, to check if the account was created successfully.

1. Drop a Data Mapper component under the **Add Account** step and name it **Check If Account Created**.
2. Add an **XML** source.
3. From the **XSD File** property, select the following: **Sugar seminar\Accounts.xsd**.
4. From the **Variable** property, select the **F.Account** variable.
5. Add a **Variable** destination.
6. From the **Variables** property, select the **F.AccountExists** variable.
7. On the Data Mapper's **Source** side, this time you'll open the **Return** folder (**Accounts > row > Return**).
8. Connect the **Success** node to **F.AccountExists**.

Although the account has been added, the contact has not yet been added:

1. Drop a Sugar connector as a child step of the **Check If Account Created** step. Name the step **Add Contact**.
2. In **Sugar Configuration** dialog box, from the **Module** property, select the **Contacts** module.
3. Set the **Operation** field to **Create**.
4. Store the result in the **F.ContactXML** variable.
5. Click **OK**.
6. Add a new **XML** source and name it **FetchContactFromRequest**.
7. From the **XSD File** property, select the following: **course_data\schemas\request.xsd**.
8. From the **Variable** property, select the **C.RequestXML** variable.

You are now ready to map.

9. On the Data Mapper's **Source** pane, open the following node: **Request > CustomerDetail**.
10. On the Destination pane, open **Contacts > row > Fields**.
11. Connect the **Customer_Name** node to the **first_name** node and the **last_name** node.

In order to have the first name and last name appear together as the customer name, you'll use expressions.

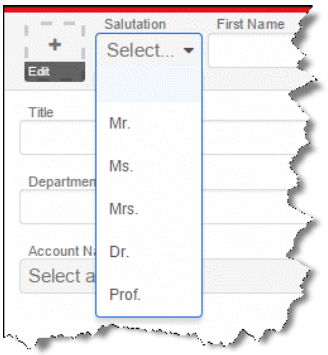
12. Place the cursor on the **first_name** node and in the **Calculated Value** property, enter the following expression: StrToken (RepStr (Trim (Src.S1/Request/CustomerDetail/Customer_Name),' ', '_') , 1 , '_')

In the expression above, remember that you enter the path **Src.S1/Request/CustomerDetail/Customer_Name** by clicking the **Source Nodes**  icon at the top of the Expression Editor. The expression first replaces the separating space with an underscore and then fetches the first token. This is because a space cannot be a token

delimiter. Now you'll do the same for the last name.

13. Place the cursor on the **last_name** node in the **Calculated Value** property, enter the **StrToken (RepStr (Trim (Src.S1/Request/CustomerDetail/Customer_Name),' ', '_') , 2 , '_')**
14. Connect **E-mail_Address** to **email1**. Make sure that you select **email1** and not just **email**.
15. In the **Destination** pane, place the cursor on the **account_id** node and from the **Calculated Value** property, select **F. AccountId**, the ID returned by the **Add Account** step.

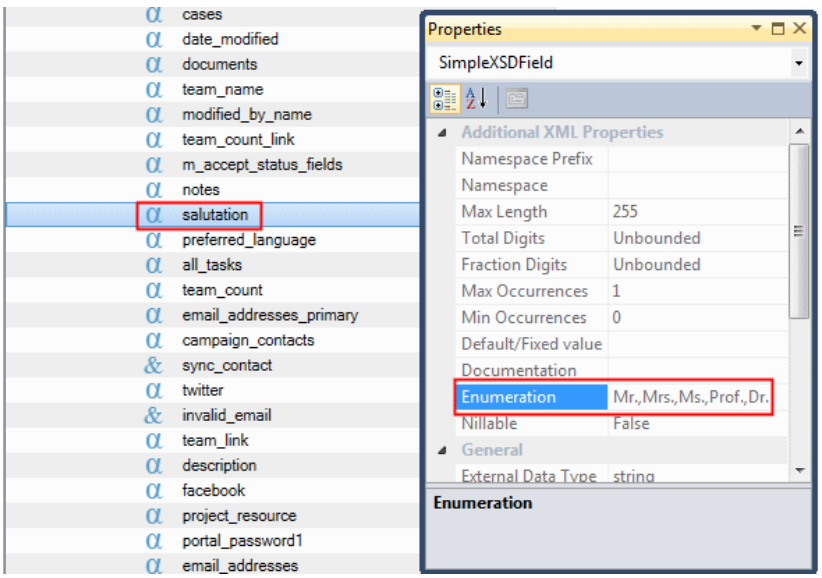
When adding a new object to SugarCRM from your Internet browser, a dropdown list provides a predefined list of available values.



For example, in the **Contacts** module, a dropdown list enables you to select whether the contact is Mr., Ms., Mrs., Dr., or Prof.

These values are provided internally by SugarCRM.

1. In the **Destination** pane, place the cursor on the **salutation** node.
2. In the **Additional XML Properties** section of the **Properties** pane, go to the **Enumeration** property and you will see the available options as defined by SugarCRM. You can also see these read-only options in the bottom left of the Data Mapper screen.
3. Manually enter **'Mr.'** as the value in the **Calculated Value** property.

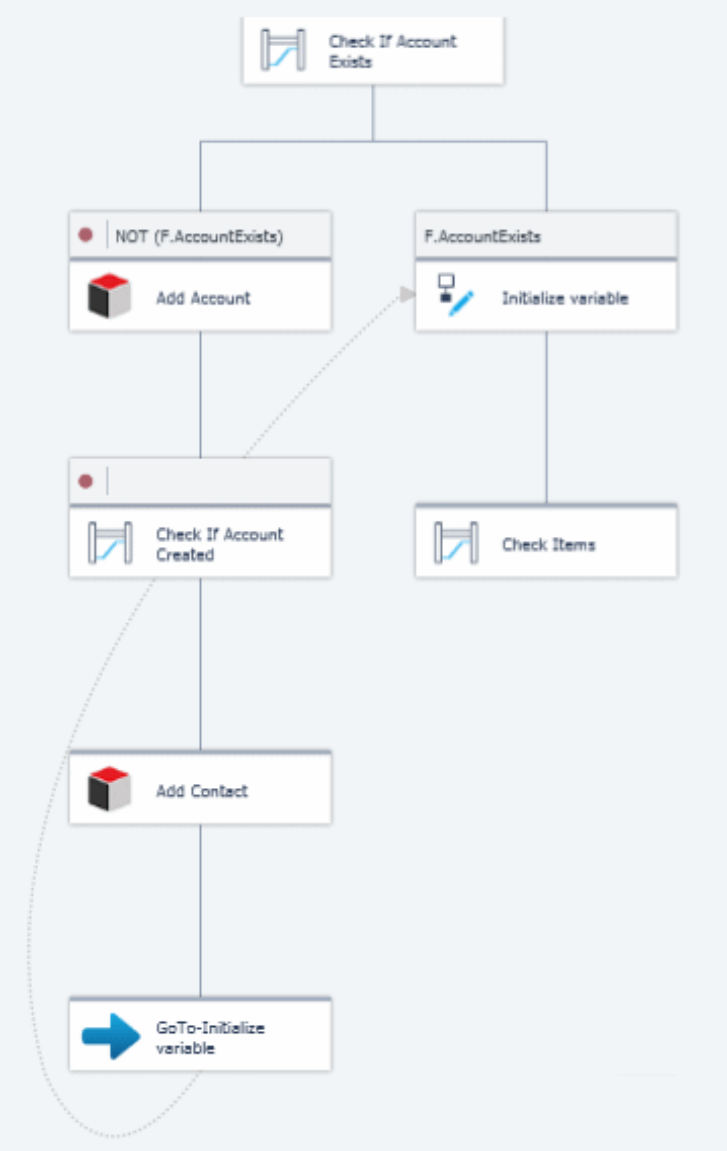


You have now finished adding the contact.

You can add a validation step like you did above when you added an account. However, the steps won't be presented here.

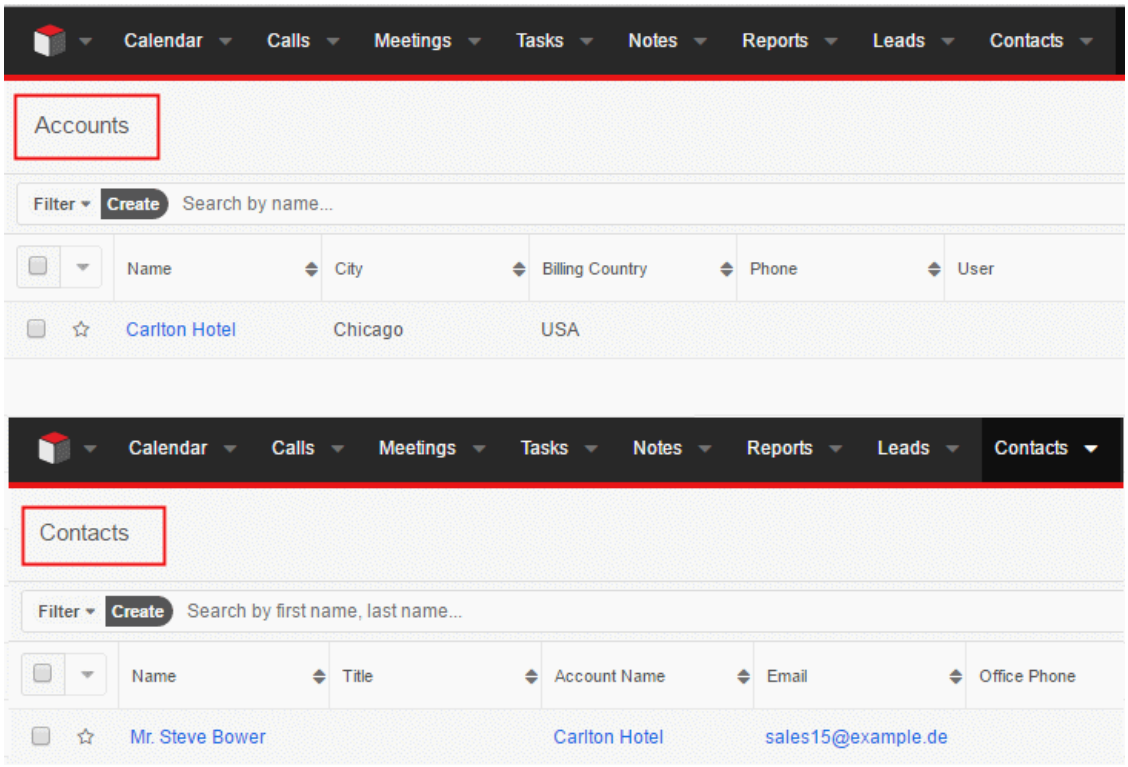
If the flow succeeds, you want to carry out the same steps that you did when the account existed.

1. Right-click on the **Add Contact** step, select **GoTo** and click on the **Initialize variable** step.
2. Add the following condition to the **Initialize variable** step: **F.AccountExists**.



Now you'll check the flow.

3. Move the **Non existing account and contact.xml** file from the **out** directory into the **in** directory.
4. Run the Debugger and then look in SugarCRM to make sure that a new account and contact were added as shown in the image below.



As with the **Query** operation, the Magic xpi Sugar connector saves the XML Schema, the XSD, in the following directory:

[project dir]\[project name]\[project name]\SugarCRM\XSD\[resource name]

Exercise

Check to see if the items in the request are valid, meaning that they exist and the requested prices is acceptable. If the items are valid, add the request as a SugarCRM **opportunity**.

The opportunity should meet the following criteria:

- Close this opportunity in two months' time.
- In the **Next Step** field, enter **Send email to customer**.
- For the **Stage Name** field, enter a value from the selection list.

Once you have tried this on your own, please make sure to look at the solution for this exercise on page 69.

Summary

In this lesson, you:

- Learned how to add an object to the SugarCRM database.
- Added an account and a contact for that account.
- Added a new opportunity.

Lesson 4

SugarCRM Object ID

In the previous lessons, you learned about the Magic xpi Sugar connector, and you were able to fetch objects from SugarCRM objects using criteria sent from Magic xpi. You were then able to use Magic xpi to perform other flow activities.

Any SugarCRM object can be queried in the manner that was discussed in the previous lesson.

In SugarCRM, every object has a unique identifier, an object ID. Some objects in SugarCRM require a query based on an ID from a parent object. Magic xpi enables you to query objects by the object ID.

This lesson covers various topics including:

- Creating an object by ID
- SugarCRM Object ID
  Magic xpi's getObjectIDbyField internal function

Creating Objects by ID

When you create or delete a SugarCRM object, which is dependent on a parent object, you need to retrieve the parent object's ID. To simplify this process, you can use the internal **getObjectIDbyField** function in the node's **Calculated Value** property.

The function can only be used within a Sugar connector step, and is not seen in the function list.

Syntax	'getObjectIDbyField (ModuleName, FieldName, FieldValue, ErrorIfEmpty)'
Parameters	**ModuleName** is the name of the SugarCRM module exactly as it appears in the API.
	FieldName is the name of the module field that is used in the operation.
	FieldValue is the value to create.

	ErrorIfEmpty, when set to true, determines that:
	• If the method does not find an ID, the operation will not be performed.
	• If the getObjectIDbyField function returns an empty value, then:
	  ◦ The Create operation will not create an object, and an error will be returned in the result XML.
	  ◦ The Update operation will not update an object, and an error will be returned in the result XML.

Return	The ID of the linked object.
Example	**'getObjectIDbyField (Accounts, account_id, Nelson Inc, true)'** returns the ID of the account whose name is **Nelson Inc**.
Note	• It is important to make sure that the whole expression is enclosed by single straight quotation marks (' '). It is a string, and the whole string is passed to the Sugar connector for parsing. This is why the whole string is encompassed by single apostrophes. • Currently, this function is not supported for the Query operation. • When you use this function, you can only search for a single value. You cannot find the ID based on more than one node, for example the Name and the City. If the search discovers more than one entry, only the first ID that was found is returned.

Fetching the ID of the SugarCRM Account

In the section above, the example looked for a SugarCRM account named **Nelson Inc**. This is the name of an account in a demo provided by SugarCRM. If you do not have this account in SugarCRM, use one from your SugarCRM database.

Now you will add a contact that belongs to that account.

For the purpose of this example, you will add a new flow.

1. Create a flow called **Scan for Contacts**.
2. Add one flow variable:
   - **F.Contact**, a BLOB variable. This variable will hold the result from the Create operation.
3. Drag a Sugar connector as the first step in the **Scan for Contacts** flow.
4. Set the step name to **Check for Contact**.
5. In **Settings** section of the **Properties** pane, make sure that the **Resource Name** property is set to the **Sugar seminar** resource. As this is the only SugarCRM resource, it was probably selected automatically by Magic xpi.
6. Open the **Sugar Configuration** dialog box.

Checking the contact's existence

You will use the Sugar connector to check whether a contact exists in an account in SugarCRM.

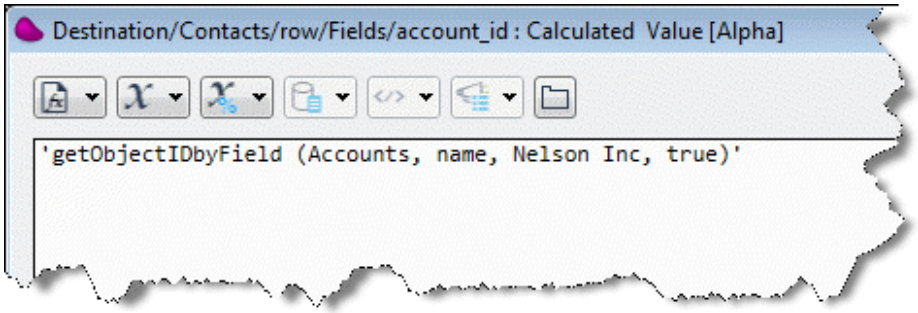
Make sure that the SugarCRM account that you are using has at least one contact. In the SugarCRM demo system, the **Nelson Inc** account has three contacts.

1. In the **Module** field, click the selection button  and select the **Contacts** module from the selection list.
2. In the **Operation** field, make sure that **Create** is selected.
3. In the **Store Result In** field, select the **F.Contact** variable that you defined earlier.
4. Click **OK**.

This is very similar to the previous lesson. As with the previous lesson, you are currently using the Sugar connector with the XML interface. Therefore, you use the Data Mapper to configure it.

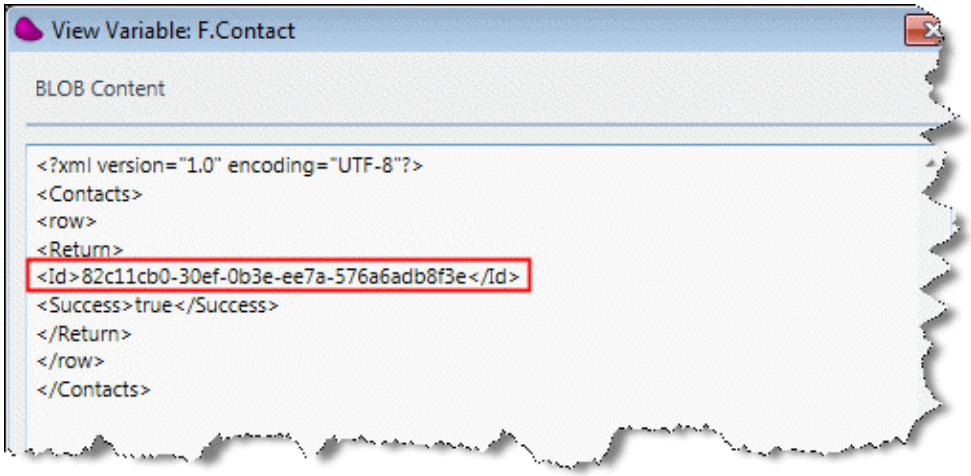
The **Contacts** module requires the **account_id** entry of the account that the contact belongs to. For this, you need the ID.

5. In the **Destination** pane, park on the **account_id** node.
6. From the **Calculated Value** property, zoom to the Expression Editor.
7. In the Expression Editor, enter **'getObjectIDbyField (Accounts, name, Nelson Inc, true)'**. Do not forget the apostrophes. If you do not have the **Nelson Inc** account, then add the account name or any of your own accounts. The **Nelson Inc** account is provided as an example.



In this example, the **getObjectIDbyField** searches for the **ID** of the **Nelson Inc** account.

8. In the **first_name** node's **Calculated Value** property, enter: **'David'**.
9. In the **last_name** node's **Calculated Value** property, enter: **'Martin'**.
10. Set a breakpoint on the step.
11. Check the functionality by using the Debugger on the flow. You do this by right-clicking on the flow in the **Solution Explorer** and selecting **Debug**.
12. When the Debugger stops, in the Context Tree, right-click on the **Check for Account** entry and select **Step**.
13. When the Debugger stops again, if everything was configured correctly, you will find the contact ID in the Context View for the **F.Contact** variable.



14. You will also see a new contact in SugarCRM named **David Martin** that is part of the **Nelson Inc** account.

Summary

In this lesson:

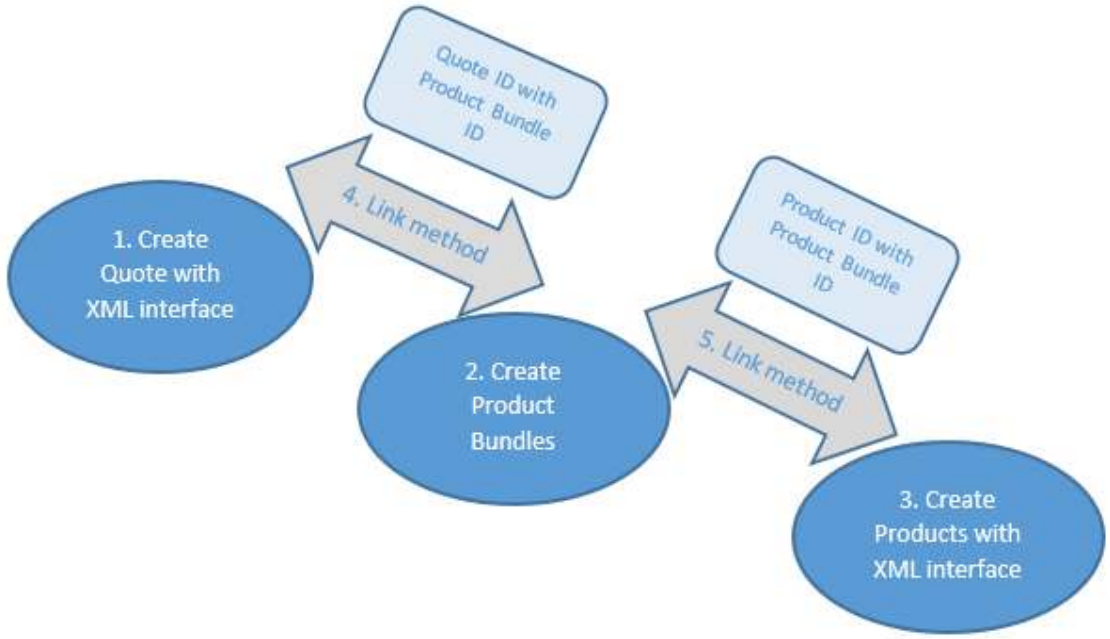
- You learned that each SugarCRM object has a unique ID that uniquely identifies it.
- You learned that by using the getObjectIDbyField Sugar connector function, you can retrieve the ID of a specific object by querying the value of a field.

Lesson 5

Creating a SugarCRM Quote Scenario

There are five steps for creating a SugarCRM quote:

1. Use the SugarCRM XML interface to create a quote.
2. Use the **Create Product Bundles** method to create a SugarCRM group.
3. Create the products that you want to have in your quote, making sure that you define values such as quantity, price, and relevant discounts.
4. Use the **Link** method to link the product bundle and the products.
5. Use the **Link** method to link the quote and the product bundle.



Using the XML Interface to Create a Quote

The quote scenario is not supported in the Legacy API implementation of the connector.

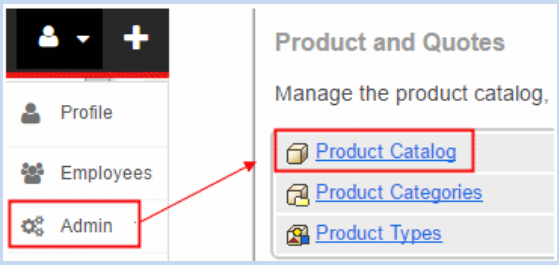
1. Create a flow and name it **Sugar Quote**.
2. Place the cursor on the flow in the **Solution Explorer** and set the **Auto Start** property to **Yes**. By setting this property to **Yes**, you're telling Magic xpi to start this flow when you run or debug the project.

Creating the Variables

You'll define now the variables that you'll need in this flow. As you continue on with the seminar, you'll see what each one is used for.

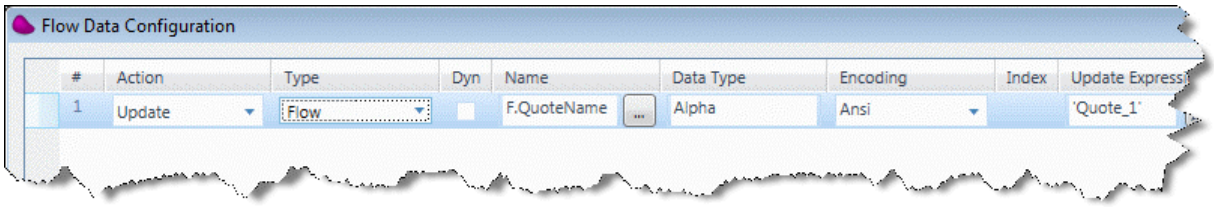
3. Create the following context variables:
   - **C.QuoteID**, Alpha 100
   - **C.ProductBundleID**, Alpha 100
   - **C.Products**, Blob
   - **C.ProductID**, Alpha 100
   - **C.LinkBundlewithProduct**, Alpha 100
   - **C.LinkQuotewithProduct**, Alpha 100
4. Create the following flow variables:
   - **F.QuoteXML**, Blob
     F.QuoteName, Alpha 30

In SugarCRM 7, the products are listed in the **Quoted Line Items** module and the Product Catalog is accessed via the **Admin** menu.



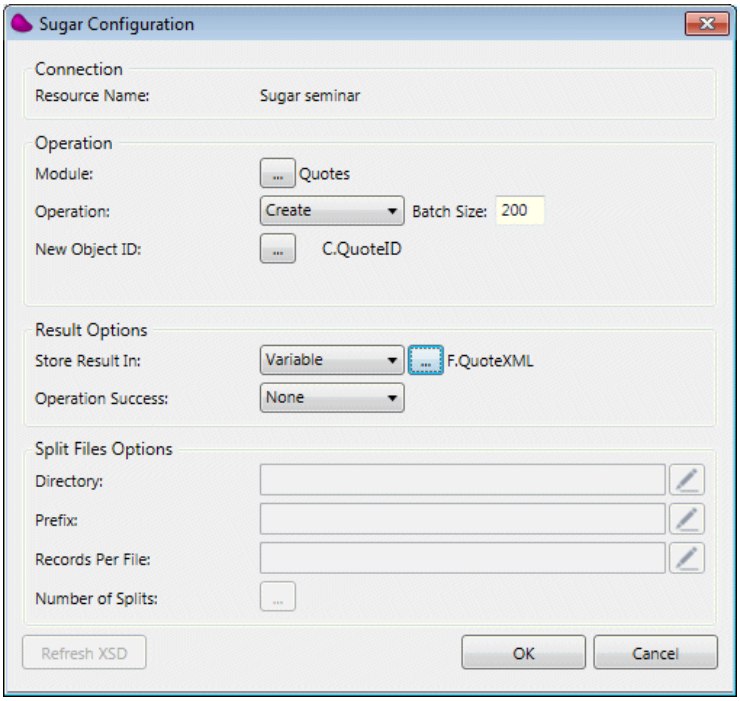
Defining a Flow Data Service

1. Add a Flow Data Service to the flow.
2. Create a new entry with the following values:
   - **Action = Update**
   - **Type = Flow**
   - **Name = F.quoteName**
   - **Data Type = Alpha**
   - **Encoding = Ansi**
   - **Update Expression = 'Quote_1'** This will serve as the name of the quote.



Creating a Quote

1. Add a Sugar step under the Flow Data step.
2. Set the **Name** property to: **Create a Quote**.
3. Leave the **Interface** as **XML**.
4. In the **Sugar Configuration** dialog box, set the **Module** to **Quotes**.
5. Set the Operation to Create.
6. From the **New Object ID** field, select **C.QuoteID**.
7. From the **Store Result In** field, select the **F.QuoteXML** variable.



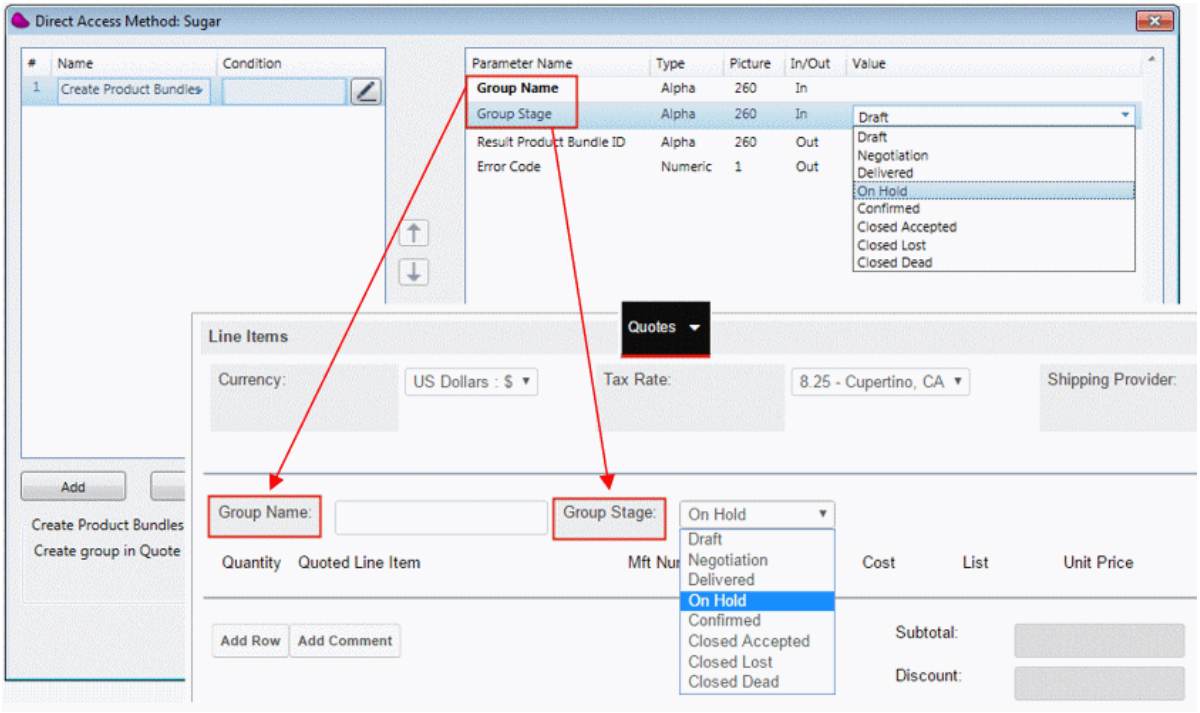
8. In the Data Mapper, right click on the following nodes and set their **Calculated value** properties:
   - **name = F.QuoteName**
   - **date_quote_expected_closed = '06/11/2016'DATE**
     This will be entered in the **Valid Until** column in SugarCRM. This is a quick way of creating a date object for specific data.
   - **quote_stage = 'Draft'**
     This will be entered in the Quote column in SugarCRM.

Creating a Product Bundle

1. Drag a Sugar connector under the **Create a Quote** step and name it **Create Product Bundle**.
2. Set the **Interface** property to **Method**.
3. Double click on the step. The **Direct Access Method: Sugar** dialog box opens.
4. Add a new **Create Product Bundles** method.

The Magic xpi **Create Product Bundles** method lets you bundle products into a group. This creates a Group in SugarCRM's **Quote Line Items**.

As you can see in the image below, the **Group Name** and **Group Stage** parameters in Magic xpi populates the **Group Name** and **Group Stage** fields in SugarCRM.



5. In the **Group Name** parameter, enter the text: **'Priority Customer'**.
6. In the **Group Stage** parameter, select **Draft**.
7. In the **Result Product Bundle ID** parameter, select the **C.ProductBundleID** variable. You'll

use this to Link the module to the quote.

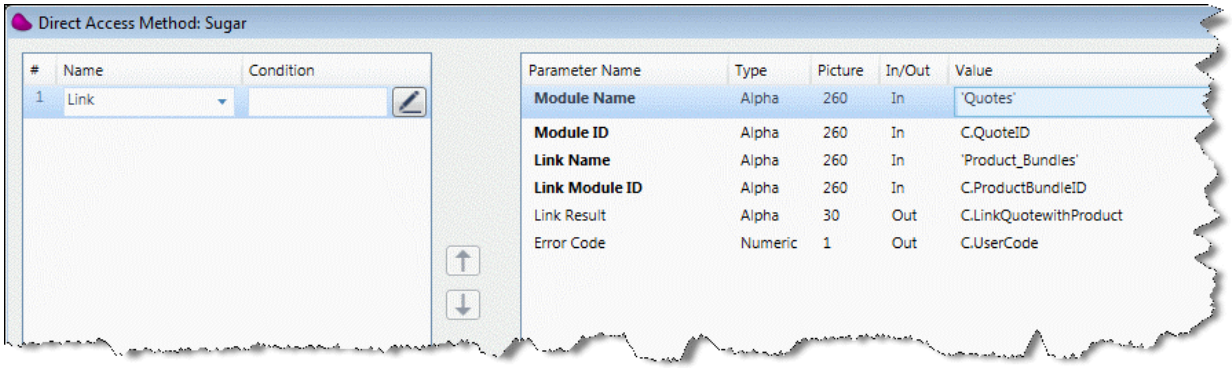
8. In the **Error Code** parameter, select **C.UserCode**.

Link the Quotes with the Product Bundle

Now go back to the **Sugar Quote** flow where you'll link the Quote ID with the Product Bundle. You'll use the Link method, which creates links between quotes, product bundles, and products.

Note: You can link multiple products to one product bundle and this product bundle will be linked to the quote object.

1. Drag another Sugar connector to the end of the flow and name it **Link Quote with Product**.
2. Create a **Link** method.



3. In the **Module Name** parameter, enter **'Quotes'**.
4. In the **Module ID** parameter, select **C.QuoteID**.
5. In the **Link Name** parameter, enter **'Product_Bundles'**.
6. In the **Link Module ID** parameter, select **C.ProductBundleID**.
7. In the **Link Result** parameter, select **C.LinkQuotewithProduct**.
8. In the **Error Code** parameter, select **C.UserCode**.

Creating Products

You will now create multiple products.

1. Drag a Sugar connector to the end of the flow and name it **Create Products**.
2. Set the **Interface** to **XML**.
3. Set the **Module** to **Products**.
4. Set the **Operation** to **Create**.
5. From the **New Object ID** field, select the **C.ProductID** variable.
6. From the **Store Result In** field, select the **C.Products** variable.

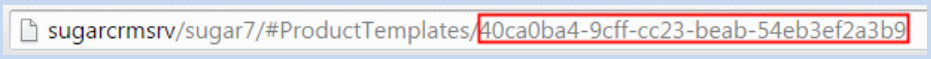
7. Click **OK**.

Now you'll use the **products.csv** file that is in the **course_data** folder. Two products have been defined here and you'll set up Magic xpi so that it creates these products in SugarCRM.

	A	B	C
1	Santo Gadget	11c7d71d-3928-c758-2fef-54eb3e0fc589	900
2	Angelica Gadget	8e0b984c-4f7f-5f89-6140-54eb3ec1e36f	856

You can see that the file includes two products with three columns.

The second column is the Product ID. In SugarCRM, you can find the Product ID in the URL. For example, you can see a Product ID at the end of the following URL:



8. On the **Source** side, add a new **Flat File** entry.
9. From the **Source Type** property, select **File.**
10. In the **File Path** property, enter the following expression:
    EnvVal ('currentprojectdir')&'course_data\Products.csv'
11. From the **Collection** property, click the ... button.
12. In the **Flat File Properties** dialog box, define the following three entries:

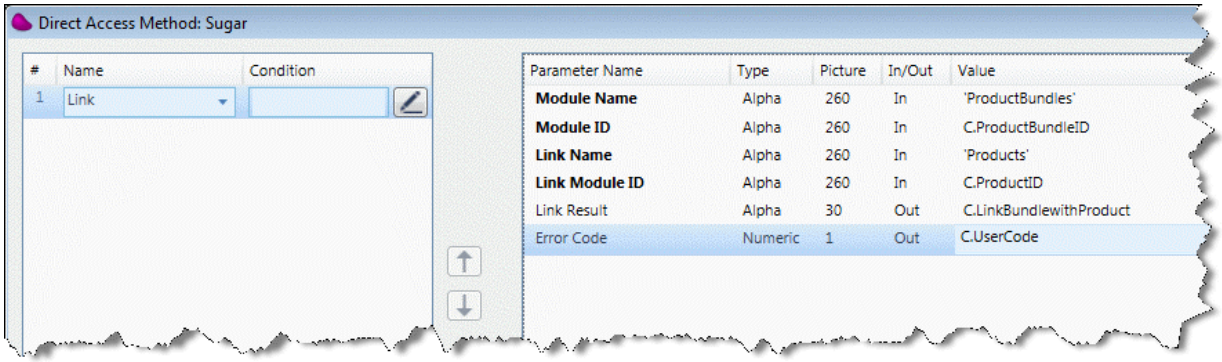
Name	Data Type	Format	Length
Name	Alpha	30	30
ID	Alpha	100	100
Price	Numeric	12.4	17

13. Click **OK**.
14. On the **Source** side, open the **Record** node.
15. On the **Destination** side, open the **Products > row > Fields** node.
16. Map the following nodes:
    - **Name** to **name**
    - **ID** to **product_template_id**
      The product ID will be generated automatically by SugarCRM once the product is created.
    - **Price** to **list_price**

Link the Product Bundle and the Products

You will now link all of the products to the Product Bundle.

1. Create a new flow.
2. Name the flow: **Link Product to Product Bundle**.
3. Drag a Sugar connector onto the flow and name it **Link Bundle with Products**.
4. Create a new **Link** method.



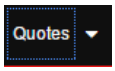
5. In the **Module Name** parameter, enter **'ProductBundles'**.
6. In the **Module ID** parameter, select the **C.ProductBundleID** variable.
7. In the **Link Name** parameter, enter **'Products'**.
8. In the **Link Module ID** parameter, select the **C.ProductID** variable.
9. In the **Link Result** parameter, select the **C.LinkBundlewithProduct** variable.
10. In the **Error Code** parameter, select the **C.UserCode** variable.

Now you'll define a Data Mapper that will call the **Link Product to Product Bundle** flow that you just created.

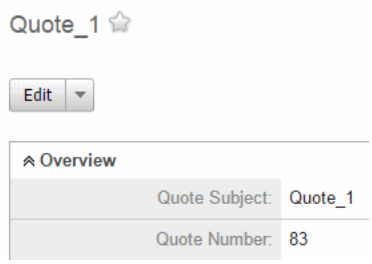
1. In the **Link Quote with Product** flow, drag a Data Mapper component below the **Create Products** step.
2. Create an **XML** source.
3. In the **XSD File** property, select the following: **SugarCRM\XSD\Sugar seminar\Products.xsd**.
4. In the **Variable** property, select **C.Products** variable.
5. Create a **Call Flow** destination.
6. From the **Flow Name** property, select the **Link Product to Product Bundle** flow.
7. Map the **Products > row > Return > Id** to **C.ProductId**.
8. On the **Destination** side, place your cursor on the **Link Product to Product Bundle** flow and set the **Condition** property with the following: **Src.S1/Products/row/Return/Success**.
9. Add a **NOP** step and name it **End**.

That's it. You've finished creating this flow.

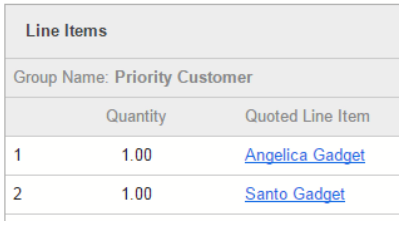
Running the Flow

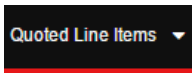
1. Run the Debugger by clicking the Start Debugging icon  .
2. Log into SugarCRM.
3. Go to **Quotes**: 

4. Open the quote that was just created.



5. You should see two items created: **Angelica Gadget** and **Santo Gadget**. This is the product bundle.



6. Go to Quoted Line Items: 

7. Click on the first new product and modify it.
8. You can then see the changes in the Quote (click again on **Quotes**).

Note that in this scenario, you did not carry out validation steps since this is just an example of how to implement the steps of the quote scenario.

Summary

In this lesson, you learned about a specific scenario in Magic xpi – how to create a SugarCRM Quote.

You learned that this is a five step process:

1. Using the SugarCRM XML interface to create a quote.
2. Using the **Create Product Bundles** method to create a SugarCRM group.
3. Creating the products that you want to have in your quote.
4. Using the **Link** method to link the product bundle and the products.
5. Using the **Link** method to link the quote and the product bundle.

Lesson 6

Capturing Events

In an integration project, you need to be able to handle actions that are invoked by the so-called other side, the entity that you want to integrate with.

Capturing events in SugarCRM enables the triggering of workflows, based on actions carried out in SugarCRM.

For example, if you need to add the customer details to a local database when an Account is created in SugarCRM, the workflow will be initiated by an action carried out in SugarCRM.

The Magic xpi Sugar connector trigger polls SugarCRM for necessary modifications and invokes the flow.

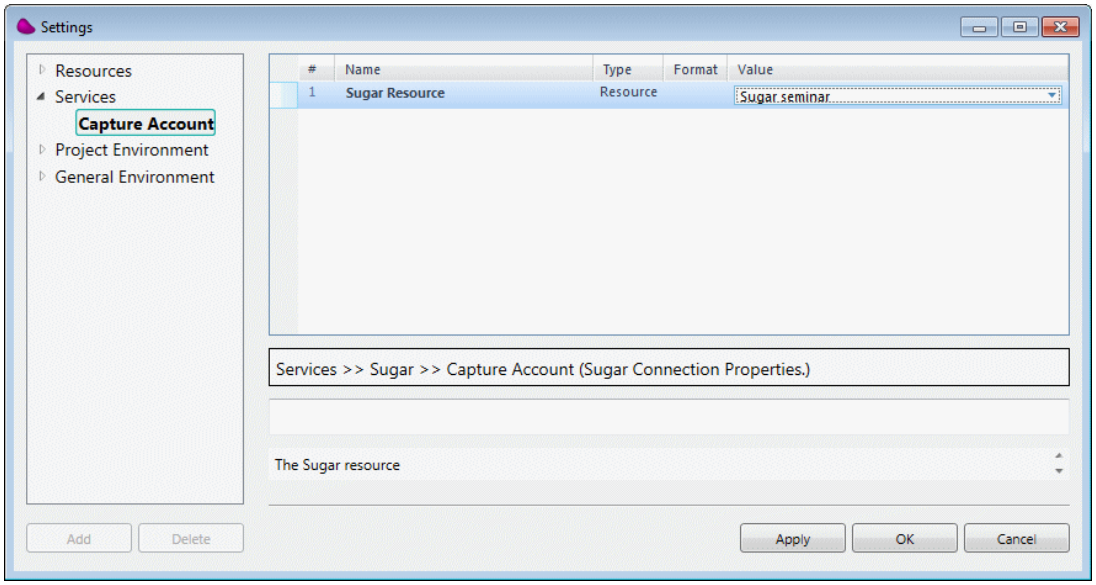
This lesson covers various topics including:

- Sugar service
- Sugar trigger
- DateTime fields

Sugar Connector Service

Before defining a Sugar connector trigger, you need to define a Sugar service as follows:

1. From the **Project** menu, select **Settings**.
2. Park on **Services** and click **Add**.
3. From the **Service Type** field, select **Sugar.**
4. Name the service: **Capture Account**.
5. From the **Sugar Resource** field, select the resource that you defined earlier: **Sugar seminar**.



Sugar Trigger

You are going to define a Magic xpi trigger that will invoke a flow whenever a new account is added in SugarCRM. The Magic xpi flow will send a welcome email to a salesperson.

Sending an email to the administrator is provided as an example of a process.

Once the flow is invoked you can add any Magic xpi component. For example, you might want to add the account as a customer in a local database or you might want to create a file of all customers added.

You are going to use a new flow for the purpose of this lesson:

1. Create a new flow and name it: **New Account Added**.
2. Before continuing, you need to add the following flow variables:
   - **F.AccountResult – BLOB**
   - **F.RowLabel – Alpha 30**
   - **F.Emailbody – BLOB**
3. Define the following **global** variable. This will be explained later on:
   - **G.TriggerStartDate – Alpha 30**
4. Initialize the **G.TriggerStartDate** global variable with the start date of the course by clicking the expression button and using the **DateTimeFormat** function in the following format: **DateTimeFormat (Date ()-1,'10:45:30'TIME,'+03:00',1).** This will be explained later on in this lesson.

Now you will define a trigger for the flow:

1. Drag a Sugar connector to the Trigger area and name it **Scan for Accounts**. The Sugar service that you created is automatically selected.
2. Double click on the trigger and in the **Sugar Trigger Configuration** dialog box, click **New**.
3. In the **Row Label** column, you can enter your own text to identify this row, for example **AccountAdded**. The **Row Label** is useful if you have multiple lines. This property is not mandatory. The use of the label will be explained later.
4. Select the SugarCRM module that you want to poll. In this example, you will select **Accounts**.
5. Determine if you want an indication of whether the object was updated or deleted. In this example, select **Created**. Note that there is no indication as to whether this is a new account or an updated account.
6. In the **Start Date** property, select the **G.TriggerStartDate** variable that you previously created. The trigger will retrieve accounts that were added or updated from the date that you defined in the variable.

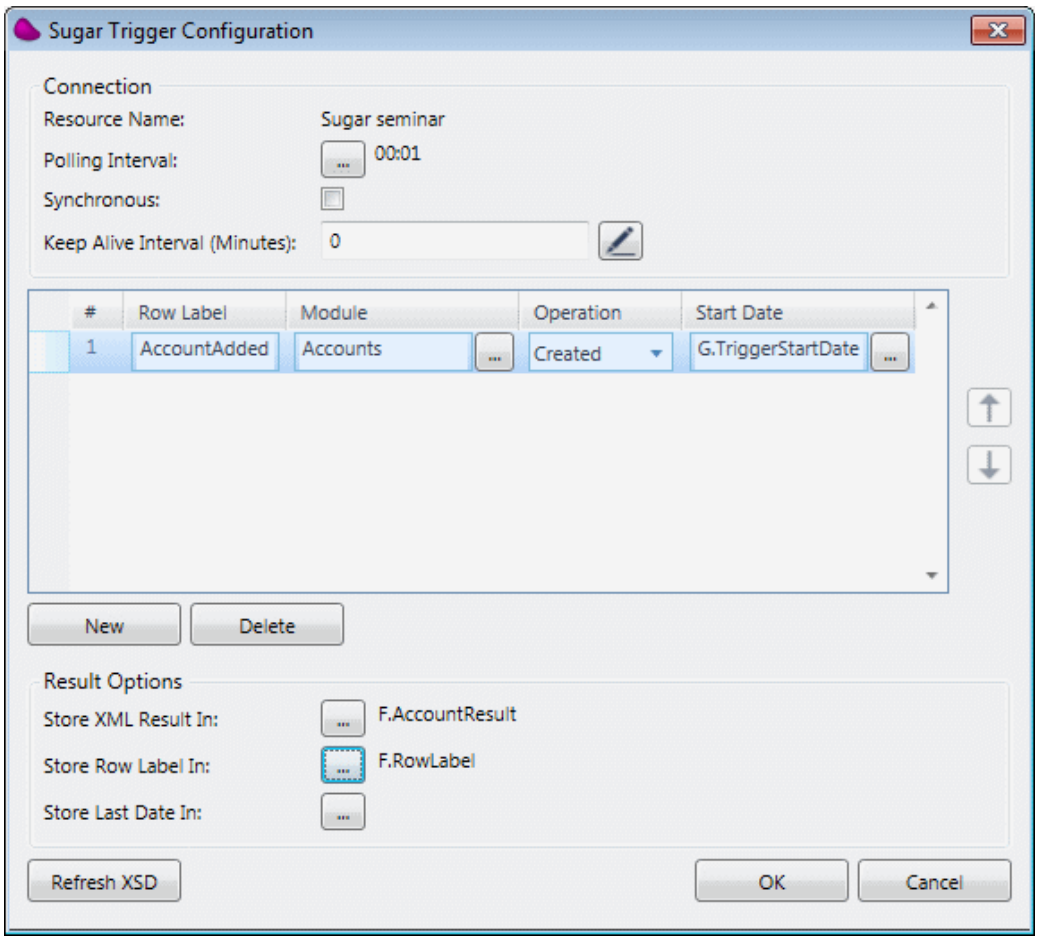
In the **Start Date** property, if the variable is empty or if you do not use a variable, Magic xpi starts polling from the next time you run the project. Magic xpi saves an indication of the last time that SugarCRM was polled for each resource, object and operation combination. The last timestamp is saved in the **Trigger.xml** file under **%currentprojectdir%SugarCRM**.

The required format for this property and SugarCRM is the XML DateTime format. This will be discussed in the next section.

Enter as many rows as needed. This is the same as adding different Sugar triggers. Then, by using the **Row Label** column *within the flow*, you can identify which trigger was actually invoked.

The Sugar connector trigger is a polling trigger. This means that Magic xpi will check the SugarCRM server at predefined intervals. By default, the interval is set to five minutes.

7. Set the **Polling interval** option to **00:01** (1 minute); otherwise, SugarCRM will use the default and wait five minutes.
8. In the **Store XML Result In** option, select the **F.AccountResult** variable, which returns the object details.
9. In the **Store Row Label In** options, select the **F.RowLabel** variable, which holds the value of the trigger that was invoked.



Data can be retrieved only for objects to which the logged-in user has access.

10. Click **OK**.

The trigger has now been defined.

You'll now create a Data Mapper step to extract specific information from the result variable, **F.AccountResult**. In this case, you'll use a template to display this field in a certain structure.

1. Drag a Data Mapper component as the first step in the flow.
2. Create an **XML** source.
3. In the **XSD File** property, select **SugarCRM\XSD\Sugar seminar\Accounts.xsd**.
4. In the **Variable** property, select the **F.AccountResult** variable.
5. Create a **Template** destination.
6. From the **Template File** property, select **course_data\Templates\AccountAdded.tpl**. This is an HTML template that you will use to send a personalized email to the administrator.
7. In the **Destination Type** property, select **Variable**.
8. In the **Variable** property, select the **F.Emailbody** variable. This will be used as the email's HTML body.
9. Map the **Accounts > row > Fields > name** node to the **CustomerName** node.

You'll now send the email to the administrator.

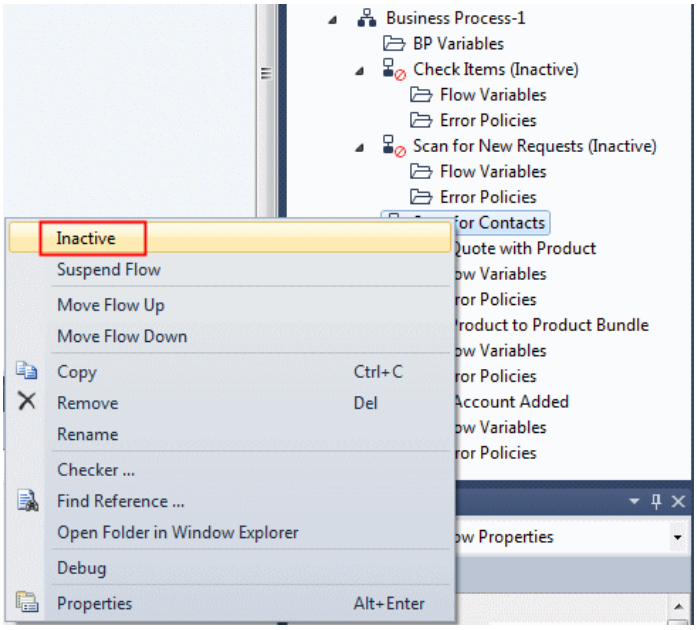
1. Add a user environment variable named **Admin_email** and set the email for your administrator (**Project** > **Settings** > **Project Environment** > **User Environment Variables**).
2. Define an **Email** resource with the settings relevant for your email server.

If you do not have the parameters of an email or are not able to define or connect to one, you can skip the email step, save the email body to the file system, and check the HTML page that was created.

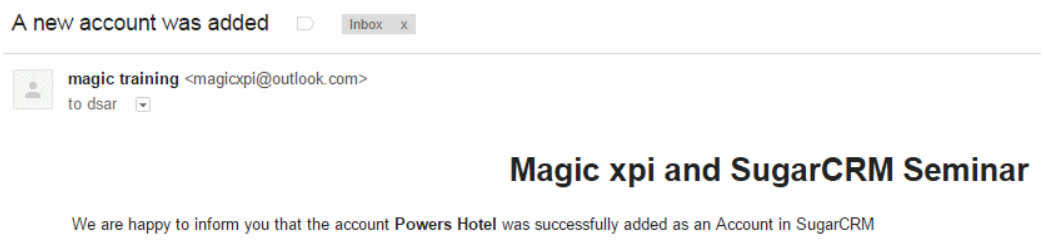
3. Click the **Validate** button to check the accuracy of the information you entered.
4. Drag an Email component as the first step and name it **Send email to administrator**.
5. Double click on the step.
6. Add a **Quick Send** method.
7. From the **To** parameter, select the **Admin_email** environment variable.
8. In the **Subject** parameter, enter the following: **A new account was added**.
9. In the **BodyType** parameter, select **HTML**.
10. In the **Body** parameter, select the **F.Emailbody** variable.

You'll now run the Debugger and see if you receive an email. However, since we don't want to run all of the flows, we'll make the other flows inactive in order to run the Debugger.

1. Right-click on each of the flows, except the last one, and select **Inactive**. In the Navigation pane, the inactive steps will appear in red.



2. Run the Debugger from the toolbar with a breakpoint on the Data Mapper step.
3. Since the trigger's polling interval was set to 1 minute, you might have to wait 1 minute until the process starts working.
4. Check to see if you receive an email.



5. When you finish working with the Debugger, remove the **Inactive** status from the flows.

DateTime Fields

SugarCRM stores **DateTime** field values as Greenwich Mean Time (GMT). When one of these values is returned in SugarCRM, it is automatically adjusted for the time zone specified in your organization's preferences.

The Magic xpi Date and Time formats do not conform to the SugarCRM convention. You need to handle this conversion in your Magic xpi project.

Syntax	**DateTimeFormat(date, time, timezone, format)**
Parameters	**date** is any date variable (or a hard-coded date, such as '05/06/2008'DATE, or an expression that evaluates to a date).
	time is any time variable (or a hard-coded time, such as '16:10:14'TIME, or an expression that evaluates to a time).
	timezone is the time zone that you want to use relative to GMT.
	format is one of the following: 1 – XML DateTime format, which is YYYY-MM-DDThh:mm:ssTZD. Make sure you add the first **T** as part of the string. 2 – JDE Julian day format, which is CYYDDD, where C is the century (0=1900 and 1=2000), YY is the year and DDD the day of the year.
Return	DateTime string in required format.
Example	If you have: DateTimeFormat('29/04/2008'DATE, '10:45:30'TIME, '+03:00',1), it returns 2008-04-29T10:45:30+3:00.
Note	The DATE that you see in the string above, '29/04/2008'DATE, is a Magic xpi literal. If you use this literal, the string is interpreted as a date. You can use it in arithmetic operations because it's internally represented as a Numeric value. So, for example, '01/01/97'DATE+14 is a valid expression that yields the date 15/01/97.

Exercise

For your own exercise (a solution is not provided with this seminar):

- If the customer exists but the contact is a new one, add the contact to the account and add this contact to the opportunity.
- Check that the contact was added to the account successfully.
- Check that the contact was added to the opportunity successfully.

Summary

In this lesson, you learned how to:

- Capture SugarCRM events.
- Trigger a flow when an event occurs in SugarCRM.
- Define SugarCRM dates in Magic xpi.

Solutions

Lesson 2 – Querying SugarCRM via Magic xpi

In this exercise, you are asked to check whether the products are valid SugarCRM products. You are asked to do this if the account exists.

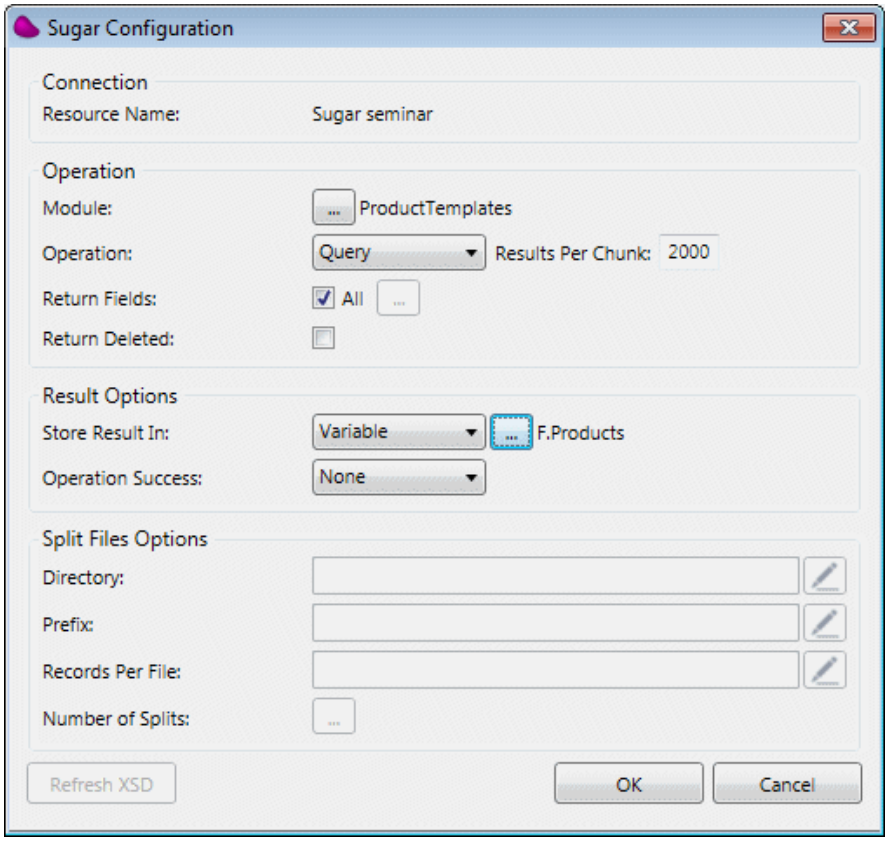
To perform this you will need a separate flow that will check each product.

1. Create a flow named **Check Items**.
2. Add the following context variable:
   - **C.All_Items_Exist**, a Logical variable with the default value set to 'FALSE'LOG.
3. Add the following flow variables:
   - **F.ItemCode**, an Alpha variable with a size of 100.
   - **F.Products**, a BLOB variable. This will hold the returned data from the SugarCRM query.
   - **F.ProductAvailable**, a Logical variable with a condition set to 'FALSE'LOG.

The **ProductTemplates** module is the SugarCRM Product Catalog.

Now you will query the **ProductTemplates** module.

1. Drop a Sugar connector as the first step of the **Check Items** flow. Name it **Query Product Template**.
2. Double-click on the step to open the **Sugar Configuration** dialog box.
3. In the **Module** property, select the **ProductTemplates** module.
4. Set the **Operation** to **Query**.
5. Set the **Store result in** property to **F.Products**.



The next stage is to map.

1. In the Data Mapper's **Destination** pane, expand **ProductTemplates > row > Fields**.
2. Park on the **id** node and in the **Calculated value** property, select the **F.ItemCode** variable.

Now you'll add a Data Mapper to check that the products exist and that the price is a relevant price.

1. Drop a Data Mapper component under the **Query Product Template** step.
2. Name it **Check Exists**.
3. Create a new **XML** source.
4. In the **XSD File** property, select the following schema: **SugarCRM\XSD\Sugar seminar\ProductTemplates.xsd**.
5. From the **Variable** property, select the **F.Products** variable.
6. Create a **Variable** destination and select the **F.ProductAvailable** variable.

The next stage is to map.

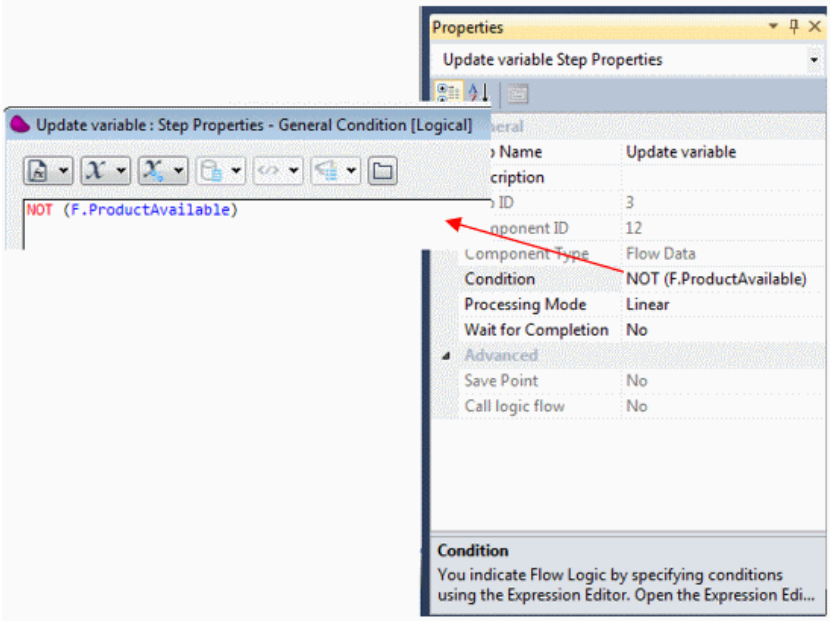
1. In the **Source** pane, expand **ProductTemplates > row > Fields**.
2. In the **Destination** pane, expand the **Instance** node.
3. Connect the **status** node with the **F.ProductAvailable** variable.
4. Set the **Calculated value** of the **F.ProductAvailable** variable to:
   Lower (Src.S1/ProductTemplates/row/Fields/status) = 'available'

We use the Lower function here because if we don't know how the data is saved in the database or application, we want to make sure that when comparing a string that we're forcing everything to be in the same case.

At this stage the **F.ProductAvailable** variable should have a value based on the logic that you defined in your mapping. In the next step you'll update the **C.All_Items_Exist** context variable based on the value of the two flow variables.

1. Drop a **Flow Data** step as a child step of the **Check Exists** step and name it **Update variable**.
2. Click on the **Flow Data** step.
3. Click **Add**.
4. Set the **Action** property to **Update**.
5. Set the **Type** to **Context**.
6. From the **Name** column, select the **C.All_Items_Exist** variable.
7. Set the **Update Expression** to **'FALSE'LOG**.
8. Click **OK**.

9. Set the following condition for this step: **NOT (F.ProductAvailable)**. This expression means that if the product is not available or if the requested price is lower than the catalog price, the order cannot be filled and we must set the **C.All_Items_Exist** variable to **False**.



If any of the previous runs of the **Check items** flow found a product that doesn't exist, no further check should be performed. You can prevent the **Check Items** flow from running by conditioning the first step as follows:

10. Right-click on the **Query Product Template** step and set the following condition: **C.All_Items_Exist**.

Now you need to call this flow for each item; but first you need to initialize the context variable.

1. Open the **Scan for New Requests** flow.
2. Drop a **Flow Data** service as a child step of the **Check if Account Exists** step. Name it **Initialize variable**.
3. Double click on the step.
4. Click **Add**.
5. Set the **Action** property to **Update**.
6. Set the **Type** to **Context**.
7. From the **Name** column, select the **C.All_Items_Exist** variable.
8. Set the **Update Expression** to **'TRUE'LOG**.

Now you are ready to call the new flow.

1. Drop a **Data Mapper** component as a child step of the **Initialize variable** step.

2. Name it **Check Items**.
3. Double click on the component.

You need to use the request XML that was retrieved by the Directory Scanner to retrieve the request items. Therefore, you need to have an XML as the source.

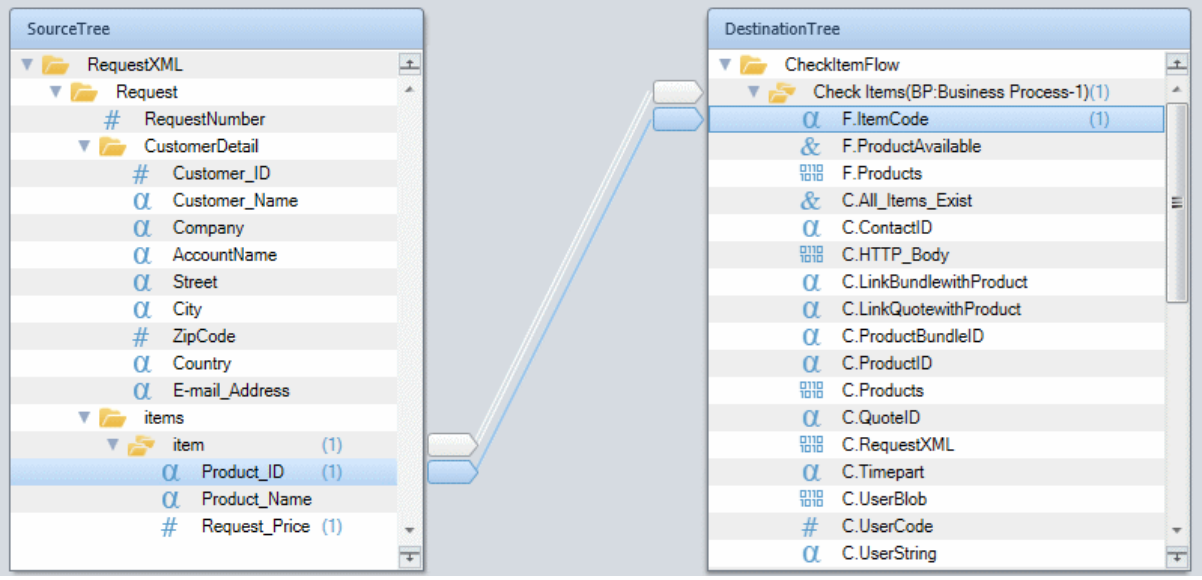
1. Add an **XML** entry to the **Source** pane of the Data Mapper.
2. Set the name to **RequestXML**.
3. In the **XSD File** property, select the following schema: **course_data\schemas\request.xsd**
4. From the **Variable** property, select the **C.RequestXML** variable.

You now need to call the new flow.

1. Add a **Call Flow** entry to the **Destination** pane of the Data Mapper.
2. From the **Flow Name** property, select the **Check Items** flow.
3. Set the **Name** property to **CheckItemFlow**.

The next stage is to map.

4. Connect the **Product_ID** node to **F.ItemCode** node.



You are ready to test.

In the **course_data/out** folder, you'll find XML requests, including:

- **Existing account and valid products.xml** – This includes a valid account and two items that exist.
- **Existing account and non existing product.xml** – This includes a valid account, but one of the items does not exist.
- **Non existing account.xml**
- **Non existing account and contact.xml**
- **Non existing account and contact for opportunity.xml** – This includes another non-existing account and contact to be used when creating the opportunity.

The valid accounts and products, including their IDs, are taken from the sample data that we installed. To make sure that the exercises run properly, open the XMLs and change the IDs so that they match the IDs that were created for these items when you installed the sample data.

You'll run the project three times using the first three XML files. You won't use the last two XMLs file in this exercise.

1. Add a breakpoint to the **Check Items** step.
2. Place one of the XML files in the **in** folder before running the project.
3. When the process is finished, for the **Existing account and valid products.xml** file, check that the **F.AccountExists** variable is set to **True**.
4. For the other two files, check that it's set to **False**.

If you get these results, the project is working as expected.

Lesson 3 – Adding an Object

In the exercise, you were asked to add an opportunity if all the items are valid.

1. In the **Scan for New Requests** flow, create two flow variables:
   - **F.Opportunity, BLOB**
   - **F.OpportunityID, Alpha 100**
2. Add a **Sugar** connector as a child step of the **Check Items** step.
3. Name the step the following: **Write New Opportunity**.
4. In the **Sugar Configuration** dialog box, from the **Module** property, select **Opportunities**.
5. Set the **Operation** column to **Create**.
6. From the **New Object ID** property, select the **F.OpportunityID** variable.
7. From the **Store Result In** property, select the **F.Opportunity** variable.
8. Click **OK**.

You need to connect the opportunity to a specific account. The **AccountId** is part of the XML returned by the **Account** Query operation. Therefore you can use this as the source.

1. Create an XML source and name it **AccountInfo**.
2. From the **XSD File** property, select the following: **SugarCRM\XSD\Sugar seminar\Accounts.xsd**.
3. From the **Variable** property, select **F.Account**.

You are now ready to map. In the **Data Mapper** screen:

1. In the **Source** pane, open the following node: **Accounts > row > Fields**.
2. In the **Destination** pane, open the following node: **Opportunities > row > Fields**.
3. Connect the **id** node to the **account_id** node.
4. Connect the **name** node to the **account_name** node.
5. In the **Destination** pane:
   a. Park on the **sales_stage** node and enter a calculated value for one of the entries for the drop-down list, such as **'Value Proposition'**. You can view the entries in the **Enumeration** property.
   b. In the **name** node, enter: **'Seminar opportunity'**.
   c. In the **next_step** node, enter the following calculated value: **'Send email'**.

The opportunity can only be added if the request is valid.

1. Park on the **Write New Opportunity** step.
2. Set the following condition: **C.All_Items_Exist**.

A SugarCRM opportunity has to have an associated Revenue Line Item.

Now you'll add a Revenue Line Item that's associated with the opportunity.

1. Create two flow variables:
   - **F.RevenueList, BLOB**
   - **F.RevenueListID, Alpha 100**
2. Add a **Sugar** connector as a child step of the **Write New Opportunity** step.
3. Name the step the following: **Create Revenue Line Items**.
4. In the **Sugar Configuration** dialog box, from the **Module** property, select **RevenueLineItems**.
5. Set the **Operation** column to **Create**.
6. From the **New Object ID** property, select the **F.RevenueListID** variable.
7. From the **Store Result In** property, select the **F.RevenueList** variable.
8. Click **OK**.
9. In the **date_closed** node, enter the following calculated value: **AddDate (Date (),0,2,0)**. This means that it will be closed in two months.
10. In the **opportunity_id** node's **Calculated Value** property, zoom to the Expression Editor and select the **F.OpportunityID** variable.
11. In the **name** node's **Calculated Value** property, enter: **'Magic products'**.
12. Set the **likely_case** node's **Calculated Value** property to **100**.

You can now test the flow using the **Non existing account and contact for opportunity.xml** file.

1. For testing purposes, add a **NOP** step under the **Create Revenue Line Items** step.
2. Put a breakpoint on the **NOP** step and remove any other breakpoints that are set.
3. Run the Debugger.

If the process worked, you should be able to find an opportunity in SugarCRM called **Seminar opportunity** and a revenue line item called **Magic products**.