



# Getting Started with Magic<sup>®</sup> xpi 4.5

Self-Paced Tutorial

**Book ID:** UTGSWMI45

**Edition:** 1.0, July 2016

**Course ID:** UCOGSXPI45

**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 xpa.

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®/IBM 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™, Dynamics® AX, 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 Drive™ 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.

GLOBEtrouter® 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.

Systinet™ is a trademark of Systinet Corporation.

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.

#### **Getting Started with Magic xpi 4.5, edition 1.0**

July 2016

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

# Contents

About the Course .....	1
Course Prerequisites .....	2
How to Use This Tutorial .....	3
Exercises .....	3
Course Material .....	3
<b>Magic xpi Overview .....</b>	<b>5</b>
Introduction .....	6
Magic xpi Studio .....	7
Components .....	8
Deploying Your Project .....	10
Version Control .....	11
Tools .....	12
Summary .....	13
<b>Magic xpi Methodology .....</b>	<b>15</b>
Identifying Business Processes .....	16
Identifying Participating Applications .....	16
Identifying Project Resources .....	17
Identifying Project Services .....	18
Designing Flows .....	19
Testing and Deploying .....	19
Summary .....	20
<b>Magic xpi Project .....</b>	<b>21</b>
Starting the Magic xpi GSA Service .....	22
Creating a Project .....	22
Environment Variables .....	25
Project Properties .....	27
Course Project .....	28
Summary .....	30
<b>Resources .....</b>	<b>31</b>
Advantages .....	32
Resource Types .....	32
Exercise .....	36
Summary .....	36
<b>Scan for New Requests .....</b>	<b>37</b>
Flow Editor .....	38
Defining a Magic xpi Flow .....	39
Introduction to Variables .....	42
Flow Components .....	43
Directory Scanner Component .....	44
Using a Component .....	45
Exercise .....	52
Summary .....	54

Flow Orchestration .....	55
Variables .....	56
Environment Variables .....	57
Project-Specific Variables .....	59
Flow Logic.....	62
Expression Editor .....	66
Exercise .....	69
Summary.....	69
Checking Customer Existence .....	71
Data Mapper.....	72
Checking for Customer Existence .....	80
Exercise .....	86
Summary.....	86
The Runtime Environment.....	87
The Executable File.....	88
Executing a Project.....	90
Magic Monitor.....	91
Exercise .....	95
Summary.....	95
Testing Your Project .....	97
Magic xpi Checker.....	98
Magic xpi Debugger .....	100
Adding User Messages.....	106
Exercise .....	108
Summary.....	108
Item Validity Check.....	109
Flow Data Utility .....	110
Operational Data Storage.....	113
Check Item Flow .....	116
Calling the Check Item Flow for Each Item .....	119
Exercise .....	121
Summary.....	121
Services.....	123
Services Section .....	124
HTTP Endpoints .....	127
Exercise .....	129
Summary.....	129
Checking Request Status.....	131
Intervention Process .....	132
Magic xpi Triggers.....	133
Trigger Types.....	134
HTTP Triggers .....	135
HTML Response Page .....	138
Templates.....	141
Exercise .....	143
Summary.....	143

<b>Error Handling .....</b>	<b>145</b>
Magic xpi Error Handling .....	146
Step Error Handling.....	147
Flow Level Error Handling .....	148
Dedicated Error Flow.....	150
Exercise .....	152
Summary.....	152
<b>Adding a Customer .....</b>	<b>153</b>
Systinet .....	154
Providing a Web Service .....	155
Web Service Trigger .....	158
Retrieving Information from the ODS .....	159
Testing the Web Service .....	162
Exercise .....	164
Summary.....	164
<b>Handling Approved Requests.....</b>	<b>165</b>
Publish and Subscribe Utilities .....	166
'Handle Request' Topic .....	166
Subscribing a Flow.....	167
Deleting the ODS from the System.....	170
Publish the 'Handle Request' Topic .....	171
Exercise .....	172
Summary.....	172
<b>Automatic Item Check .....</b>	<b>173</b>
Scheduler Utility .....	174
Flow Enablement.....	178
Exercise .....	179
Summary.....	179
<b>More About Magic xpi .....</b>	<b>181</b>
More about the Data Mapper.....	182
Email XML Configuration.....	188
User Defined Storage (UDS) .....	192
Exercise – Mapping a Flat File to an Order.....	197
Summary.....	198
<b>From Development to Deployment .....</b>	<b>199</b>
Recommendations .....	200
Deployment Issues .....	200
Summary.....	202
<b>Course Data .....</b>	<b>203</b>
Sample XML Requests .....	203
Entity Relations Diagram (ERD).....	204



Solutions .....	205
Solution Lesson 4 – Resources .....	207
Solution Lesson 5 – Scan for New Requests .....	208
Solution Lesson 6 – Flow Orchestration .....	209
Solution Lesson 7 – Checking Customer Existence .....	211
Solution Lesson 10 – Item Validity Check .....	215
Solution Lesson 11 – Services .....	221
Solution Lesson 12 – Checking Request Status .....	222
Solution Lesson 13 – Error Handling .....	225
Solution Lesson 14 – Adding a Customer .....	228
Solution Lesson 15 – Handling Approved Requests .....	232
Solution Lesson 16 – Automatic Item Check .....	236
Solution Lesson 17 – More About Magic xpi .....	240

## Introduction

**W**elcome to Magic Software University's **Getting Started with Magic xpi 4.5** self-paced tutorial. 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.

The self-paced tutorial is constructed with the same information that has proven successful and useful in the classroom setting. The self-paced tutorial that you are currently reading is provided in addition to the classroom course. It contains the same information but has more step-by-step information. The purpose of this self-paced tutorial is so that you can refresh your memory after sitting in on the classroom sessions. This tutorial does not replace the classroom lessons, but instead complements the classroom studies.

### About the Course

The course is intended for people who want to know how to successfully use Magic Software Enterprises' Magic xpi Integration Platform, and to be able use Magic xpi to integrate with external resources.

During the course you will learn about Magic xpi and how it works. You will use some of the flow components that Magic xpi provides to create flows, and you will learn how to connect to some external resources.



Before you install Magic xpi 4.5, please ensure that your computer meets the hardware requirements described on the next page.

We also recommend that you consult the *Magic xpi Installation Guide*.

## Course Prerequisites

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

Development knowledge	<p>Familiar with:</p> <ul style="list-style-type: none"> <li>• Databases: tables, rows, fields and indexes</li> <li>• XML</li> <li>• HTML technology, such as HTML and HTML tags</li> </ul>
-----------------------	---

Your computer must also meet some basic requirements:

Hardware	<ul style="list-style-type: none"> <li>• Windows 7 and later. The course was tested on Windows 7.</li> <li>• Pentium processor 1.8GHz and upwards.</li> <li>• 4Gb RAM or greater</li> <li>• At least 2Gb free space</li> <li>• Your screen should be set at a resolution of at least 1024x768 pixels.</li> </ul>
MSSQL Server	Access to MSSQL version 2005 and above. You will need the supervisor password for the installation.
Web Server	A personal IIS Web server must be installed on your computer. Some exercises that you will develop use the Web server.
License	The course uses the standard license. Please obtain a Magic xpi 4.5 IBNPSRV evaluation license from your local Magic Software Enterprises representative.
Email Server	You need access to an email server so that you can send emails. You can use your Gmail, Yahoo or Hotmail accounts as well. Check the internet for instructions on how to configure those mail servers for POP3, IMAP and SMTP.
Email Client	<p>You will need an email client to view emails that were sent by the system. You can use an email client such as Mozilla Thunderbird.</p> <p>If you are using a web-based email such as Gmail, you can use the internet browser to check the mail.</p>



## How to Use This Tutorial

To get the most out of this tutorial, follow the classroom lesson and ask any questions that you have. You can then review the self-paced tutorial for that lesson, and if you have further questions you can ask the instructor before the next lesson.

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

## Exercises

At the end of most lessons, there is an exercise. There are solutions provided for most of the exercises. These are provided as a complete project. Try to do the exercise on your own before looking at the solution.

Note that your solution may differ from the solution offered. This does not mean your solution is incorrect, as there are many ways to solve a problem. The solution provided with the course shows one method of solving the problem.

## Course Material

The course material contains the following:

- Course data – This is data that you will need during the course exercises. This includes the database scripts and the XML schemas that you will be using. During the course you will need to copy this directory to your Magic xpi project folder once you start your project.
- Student Guide – This is the course classroom handout.
- Self-Paced Tutorial – This is the book you are reading. It contains the course, but as a self-paced guide.

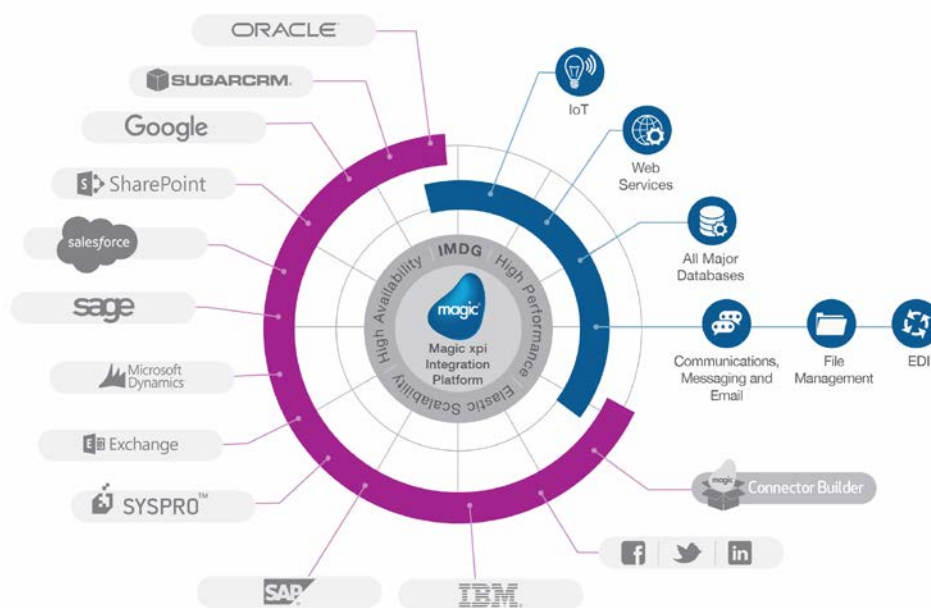


Magic®  
University

# Lesson 1

## Magic xpi Overview

Magic xpi Integration Platform is a comprehensive integration tool delivering fast and simple integration and orchestration of business processes and applications. Magic xpi enables the development and implementation of true enterprise application integration (EAI), business process management (BPM), and service-oriented architecture (SOA) infrastructure.

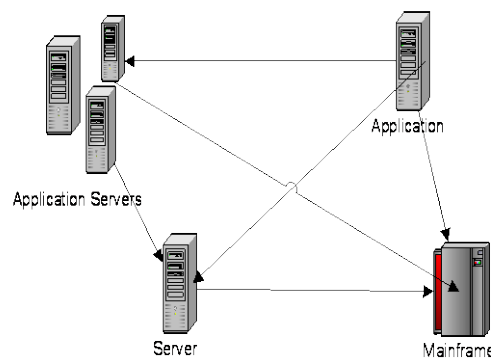


In this lesson, you will learn about integration projects, and get a first glance of the Magic xpi development environment. You will be able to identify the main components of Magic xpi Integration Platform. This lesson covers various aspects of Magic xpi Integration Platform, including:

- The Magic xpi Studio
- Deploying your project
- Tools

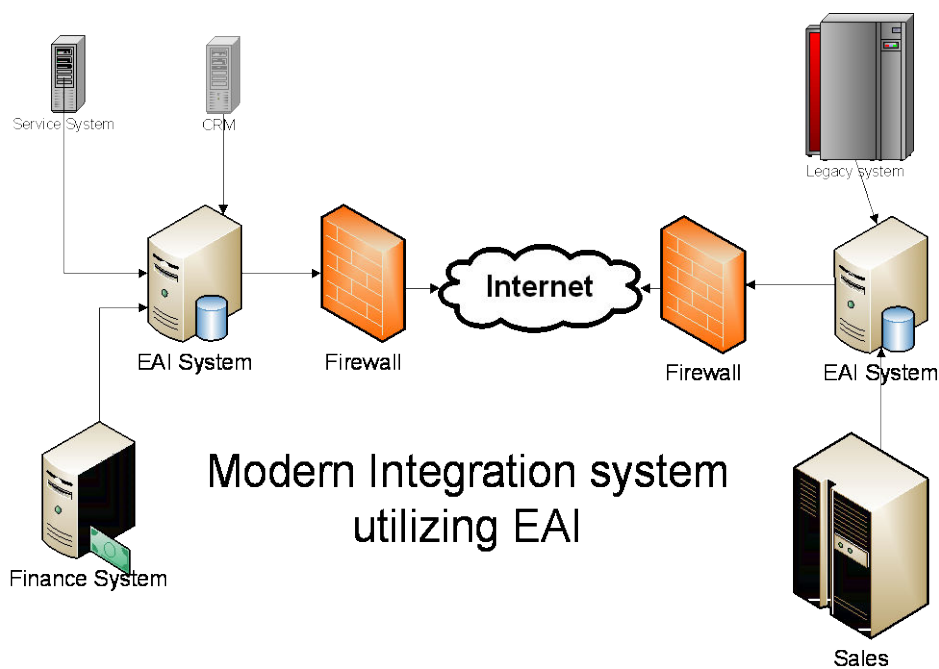
## Introduction

In the early days of application development, developers were concerned with solving specific business problems. When a data connection was needed between applications, a specific interface was developed between those specific applications. Applications were connected with no clear indication of the flow process and, as the pace of business change accelerated, connecting the systems became a major roadblock.



Enterprise Application Integration (EAI) emerged as a method to integrate the wide array of IT applications.

EAI became a top priority in many enterprises. With the internet and the creation of the extended enterprise, Business to Business (B2B) integration became possible.



## Magic xpi Studio

Magic xpi Integration Platform includes the Magic xpi Studio, which is a graphic design tool that lets you design and create integration projects for your organization. It contains the development tools for the Magic xpi project.

The Studio's design features allow you to develop your integration project in an intuitive and methodical way.

### Integration Flows

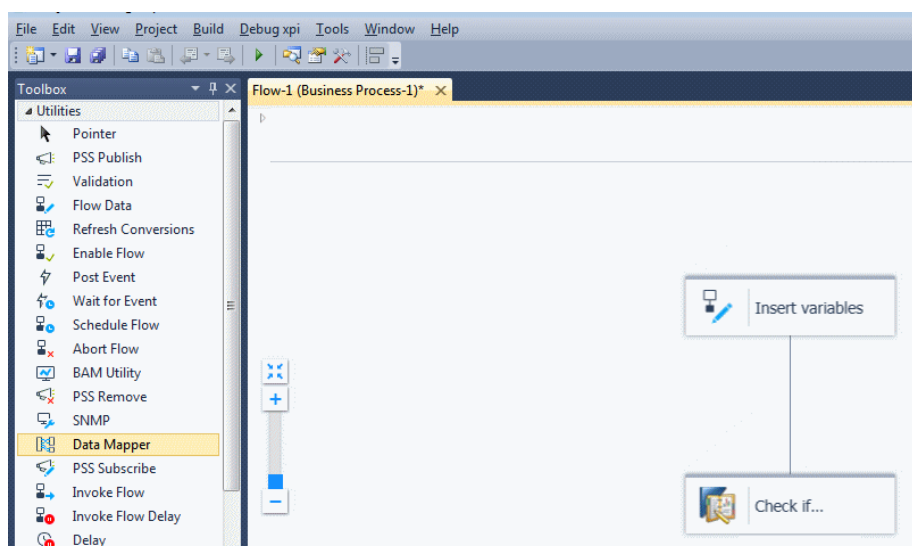
The flows are the backbone of your Magic xpi project. An integration flow is a set of steps, comprising their own logic that controls the execution order of the steps.

A step can either call a Magic xpi flow component or use one of the Magic xpi Server services.

The integration flows are executed by the Magic xpi Server during deployment. You can use a *flow variable* to define the flow's logic.

An integration flow can be comprised of several objects, such as:

- Flow components
- Utilities
- Flow variables, context variables, global variables, and system variables
- Environment variables
- Flow logic
- Error handling



## Components

A flow component is used within the integration project as a step in a Magic xpi flow.

Flow components can have connecting, adapting, converting, and processing capabilities. Magic xpi flow components can also be used to connect to third-party products and enable integration within the organization. Flow components provide connectivity functionality and are highly customizable to achieve the required integration functionality. Components are provided for use in the following ways:

- **Out-of-the-box:** These are components supplied with Magic xpi. You need to configure these components for the specific task at hand.
- **Self-developed:** These are components that you can develop for use in your integration project. Magic xpi provides a Connector Builder to create your own component.

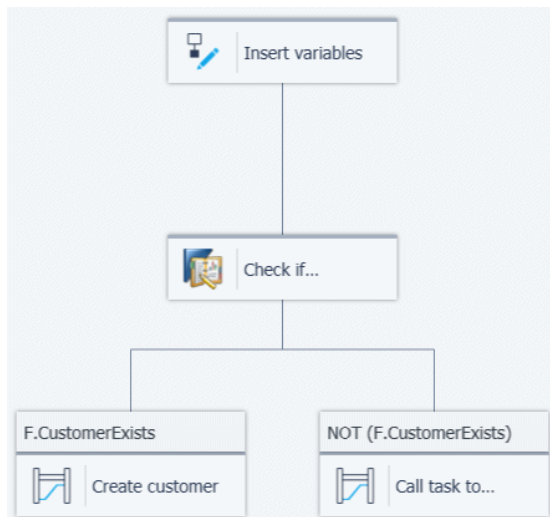
A Magic xpi flow component must provide a specific interface, which is used by the Magic xpi Studio and the Magic xpi Server to incorporate the component in the Magic xpi project.

This provides transparency between the flow components, and allows you to create your flow components in a standard way.

## Flow Orchestration

You can set conditions or rules on how the execution of the flow will perform.

Within the flow you can place conditions on the different flow components. This tells Magic xpi which branch to follow, based on which condition evaluates to 'True'. This mechanism enables you to create branches, loops, and any other flow logic that is necessary for the integration project.



## Data Management

Most integration projects deal with data, so this is something that has to be handled.

With Magic xpi, data can be handled in multiple ways. It can be:

- Transferred to a secondary destination, such as another application, or a messaging queue
- Converted to another format, such as an XML file or text file
- Saved in one or more tables (such as SAP and Salesforce) and databases
- Used as conditions for other activities

To accomplish the above scenarios, Magic xpi provides:

- The Flow Data utility
- The Data Mapper
- Flow orchestration mechanics

## Deploying Your Project

There are four elements involved in running a project:

### Magic xpi Server

The Magic xpi Server is a scalable, high performance enterprise server for deploying, running, managing and maintaining integration projects. The Magic xpi Server provides runtime containers and enterprise services that run integration processes, as defined in the integration project.

The Magic xpi Server is based on In-Memory Data Grid technology as the underlying messaging and context infrastructure, and on Magic xpi runtime processes (workers) to execute integration flows. The Server is started as a process in background mode and has no user interface.

The Magic xpi Server runs on various operating systems, such as Windows and Linux.

### In-Memory Data Grid

The in-memory data grid is middleware software composed of multiple server processes running on multiple machine instances (physical or virtual) that work together to store large amounts of data in memory.

### Magic Monitor

Since the Magic xpi Server has no user interface, the Magic Monitor is provided to monitor the Server activity.

The Magic Monitor displays information about an integration project's deployment performance and accuracy. The information provided by the Monitor enables you to examine the project (flow by flow) after the Server has processed it. This enables you to see if there are any errors that need to be fixed, and to see if any modifications need to be done to improve performance.

You can track your project's execution in greater detail through the **Dashboard**, **Messages**, **Flows**, **Triggers**, **Servers**, **Locking**, **Subscription**, **Scheduler**, **Summary**, **Activity Log**, **ODS**, and **BAM** tabs.



## Magic xpi Debugger

The Magic xpi Debugger enables you to debug the project for any errors.

The Debugger displays extra information about the current execution, such as flow variables, user parameters, flow logic, and the activity log.

## Version Control

Version Control, which is sometimes referred to as Source Management, provides the following functionality:

- Revision management of code
- Multi-user access to the same project

Magic xpi provides Version Control support for third-party version control products, such as Microsoft Visual SourceSafe.

Magic xpi Version Control functionality provides the following benefits:

- **Save versions** – You can save versions of various aspects of the project, enabling you to retrieve a previous version of any part of the project.
- **Team development** – You can develop your project in a multi-developer environment. Each developer works on a copy of the project's source files, and is responsible for synchronizing their own copy with the Version Control system.
- **File protection** – To edit a file, you first need to check out the file. This protects the source files from being accidentally overwritten by another user.

Version Control functionality is available in Magic xpi for various objects, including:

- Projects
- Business processes
- Flows – Each individual flow can be checked out
- Repositories
- Variables – Global and Context variables can be checked out

When creating a new project, you define whether the project will be implemented using Version Control. You can also add an existing Magic xpi project to a Version Control system.

You will not be learning about Version Control in this course.

## Tools

Magic xpi gives you some useful tools to create and maintain user defined components, and to check the syntax of the current project or flow.

### Connector Builder

There are certain occasions when you will need customized code that Magic xpi does not provide out of the box. Magic xpi provides a Connector Builder that allows you to create a component for your specific needs and connect to the Magic xpi Studio for use within the integration project.

### Checker

Magic xpi provides a tool to check the syntax within the integration project. The Checker tool will find things like unconfigured steps and incompatible types when scanning for problems. With Magic xpi, you have two different options to process the syntax checker, either on the specific flow or on the entire project. If the checker finds errors, it will then show you where the errors were found.

### Text Search

Magic xpi enables you to search for specific text within the integration project. You can search either on a specific flow or the entire project. If the text is found, Magic xpi will then show you where that text is located.

## Summary

In this lesson, you learned about the Magic xpi Studio. You should now be familiar with:

- Integration projects.
- The main components of Magic xpi Integration Platform.
- Flow components, and the difference between out-of-box components and self-developed components, for inclusion in your integration project.
- The built-in debugger, where you can monitor and debug your flows.
- Magic xpi tools.



Magic®  
University

# Lesson 2

## Magic xpi Methodology

When embarking on a development project, it is good practice to define a development methodology. When working according to a defined methodology, you can save valuable time by finding problems during the early stages of the project.

Magic xpi Integration Platform is a comprehensive integration tool designed to provide you with maximum integration capabilities for your organization.

When embarking on any development project, there are various stages:

Analysis ➡ System definition ➡ Development ➡ Debugging ➡ Deploying

If you work according to a defined methodology, you can prevent future errors and minimize problems that you may be faced with during each stage of the project.

This lesson introduces the recommended methodology when working with Magic xpi. The course's lessons follow this methodology.

In this lesson, you will learn about the recommended Magic xpi development methodology and you will be introduced to some of the terminology that you will be learning about during this course.

## Identifying Business Processes

A business process is the business logic that leads to a specific workflow in the organizational structure. The business process is a set of activities that are performed by people or machines.

Each business process can be broken down into smaller business logic cycles.

Example:

- You may have a business procedure called Recruitment, which begins with a search for a candidate and ends with the candidate starting to work.
- This can then be broken down into a number of separate processes, such as accepting candidates' resumes and inviting them for an interview.
- Another process could be the new employee's first day at work, in which the employee receives a car, a desk, a chair, a computer, and an email address.

## Identifying Participating Applications

The business process defines specific workflows within the organization.

The workflow is a logical flow of activities that are executed by people or machines. Some of these machines include computer applications and databases.

These applications form the backbone of the flow, since they are responsible for storing and handling information needed in order for the flow to be successful.

An application can be on a proprietary standalone server or on an application with a database that each member of the organization has access to, such as the mail server.

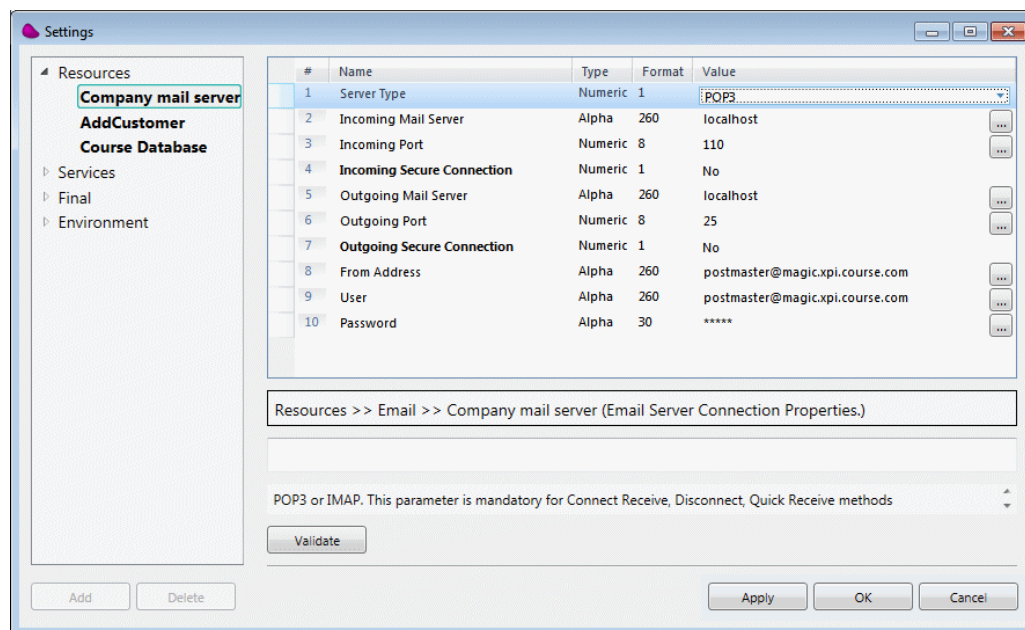
Each application that takes part in the workflow needs to be identified, including the physical location of the servers and their stress load.

## Identifying Project Resources

Once you have identified the organization's business processes, participating applications, and IT infrastructure elements, you will need to define how the Magic xpi project should interact with them.

The Magic xpi Server can interact with these applications using various methods, such as SQL, emails, messaging queues, and many others.

These external entities are resources that the Magic xpi project interacts with to provide a solution for the integration project. After you have decided how Magic xpi will interact with the external applications, you can then define the Magic xpi resources.



## Identifying Project Services

During the deployment of a Magic xpi project, the project will use the external resources that were defined in Magic xpi.

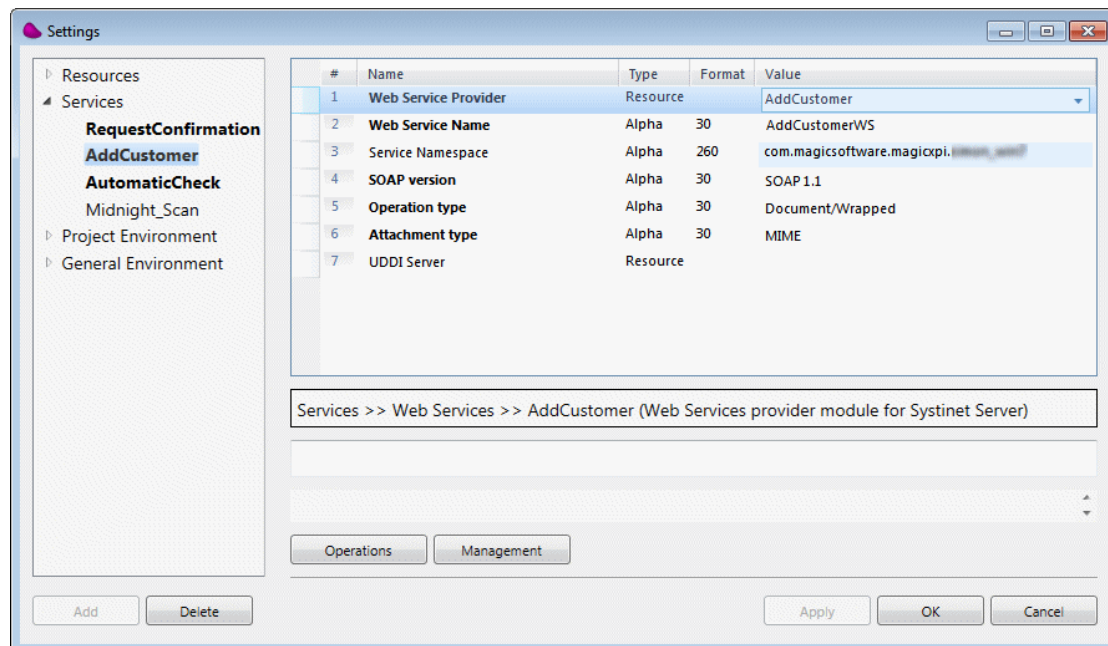
However, when defining the business process, you may find that some of these resources need to activate Magic xpi externally. To handle these external requests, Magic xpi exposes various services that an external application can invoke when required.

Services are the application units that the project exposes externally, enabling them to be invoked from external applications. The service definition holds the interface information with which applications can call a Magic xpi project.

The Magic xpi Server offers services using various methods, such as:

- Web services
- HTTP
- Messaging queues
- Email triggers
- And much more...

Once you understand how other applications need to interact with Magic xpi, you can define the Magic xpi services that are needed for the project. Services are defined in a central location for definition and management purposes. These defined services can be used when you add triggers to a flow.





## Designing Flows

Flows are the backbone of the integration project.

Each individual flow is broken down into steps that need to be implemented when developing the integration project. Each step may branch off into further steps, depending on whether a specified condition is met.

When you are in the design phase, you should give considerable thought to how each flow will be invoked. Flows can be invoked in numerous ways:

- When the project loads
- At a specified time or at polling intervals
- As a result of an external procedure, such as being triggered by an external HTTP request
- By another flow

## Testing and Deploying

As with any development project, when the integration project is developed, it needs to be tested. Your project can be tested and debugged (using the built in Debugger). Going through this process before deployment will reduce problematic issues that can occur at a later stage.

After the project has been deployed, it will work with real data and implement the business logic that was defined in the analysis stage of the implementation.

## Summary

In this lesson you learned about the recommended Magic xpi development methodology. You should now be familiar with the following terminology:

- The recommended Magic xpi development methodology
- Business processes
- Magic xpi resources
- Magic xpi services
- Integration flows

During this course you will learn more about the subjects that were touched on during this lesson.



# Lesson 3

## Magic xpi Project

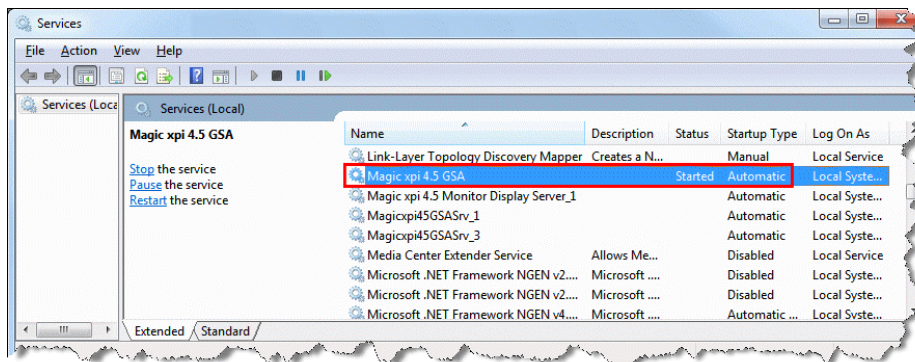
In the previous lesson, you learned about the recommended methodology for approaching a Magic xpi project.

To implement this methodology, you first need to create a new project in Magic xpi using the Magic xpi Studio. This is the first application in Magic xpi Integration Platform that you need for your integration project.

## Starting the Magic xpi GSA Service

If you selected the **Install the Grid Service Agent (GSA) as a service** check box when installing Magic xpi, the **Magic xpi GSA** service should be running on your machine. A Magic xpi 4.5 project cannot run if the service is not also running. This is the default when installing Magic xpi, so you should already have it on your machine. However, let's first check that it's up and running:


1. On your machine, from the **Start** menu, click **Run**.
2. In the **Run** dialog box, enter **services.msc**.
3. In the **Services** dialog box, look for **Magic xpi 4.5 GSA**. If it has a **Started** status and an **Automatic** startup type, then it's running.
4. If not, double-click on it and in the **Magic xpi 4.5 GSA Properties (Local Computer)** dialog box's **Startup type** parameter, select **Automatic**.
5. Then, run the service.
6. Click **OK** to finish.



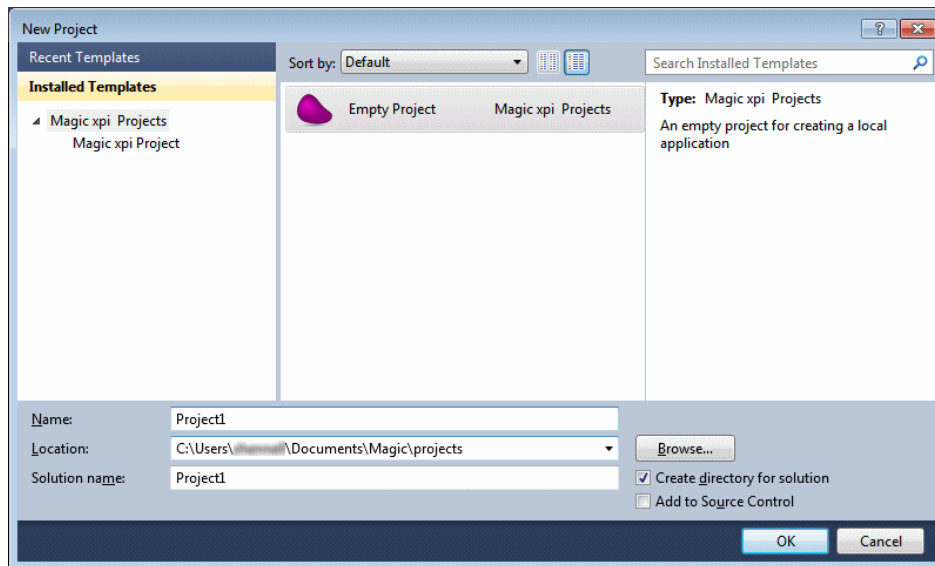
## Creating a Project

To create a new project, you need to be in the Magic xpi Studio. The Magic xpi Studio can be executed from the **Magic xpi Studio** shortcut in the Windows **Start** menu.

To create a new Magic xpi project:

- Click the **File** menu, select **New**, then **Project**. The **New Project** dialog box opens.
- Alternatively, click the **New Project** icon , or press **CTRL+Shift+N**.

Once you have opened the **New Project** dialog box (see figure below), there are three properties that you can fill in:



- **Name** – This name identifies your integration project and is used as an index in various tables. You will learn more about this later.
- **Location** – This defines the location where your project will be created.
- **Solution Name** – By default, this is the same as the name that you assigned to the project in the **Name** property (above).
- **Create directory for solution** – Select this check box if you want to change the solution name to something else.
- **Add to Source Control** – Select this check box if you want your new project to work with Version Control.

You will now get hands-on practice with the Magic xpi Studio:

1. Create a new project called: **Magic\_xpi\_course**.
2. Click **OK** to save your new project. By default, Magic xpi will save the project in the **Users\<your user name>\My Documents\Magic\projects** folder.
3. Open Windows Explorer, and go to the Windows directory where you saved the project (**Users\<your user name>\My Documents\Magic\projects**).

## Created Project Files

Once you have saved the new project, Magic xpi creates various files and directories in the **projects** directory. As mentioned earlier, the project files were created in the following path:

**My Documents\Magic\projects\<<project name>**

For example, your project was created in:

**c:\Users\<<your user name>\My Documents\Magic\projects\Magic\_xpi \_course**

Please take note of the following files. Some of these files will be discussed in more detail during this course:

Directory / File	Description
Source Directory	This is the location where Magic xpi creates all the source files that are needed for the project. All the files in this directory are in XML format, and should not be changed.
Magic_xpi _course.mgxpiproj	This is the project metadata and contains all the project definitions.
Resources.xml	This file contains the settings for the external applications that are needed for the project. You will learn about resources in a later lesson.
Services.xml	This file contains the settings for the application units that the project exposes externally, so that they may be invoked from external applications. You will learn about services in a later lesson.

## Course Data

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

- Copy the **course\_data** folder into the project's directory.

## Environment Variables

Environment variables are configuration variables that you can define for the project. They are useful for defining the environment properties within a Magic xpi project.

These variables enable the smooth portability of projects between environments:

Development Environment  Production Environment

To achieve portability, Magic xpi translates these variables at runtime, according to the values entered in the **Settings** dialog box's **General Environment** section, under **Internal Environment Variables**.

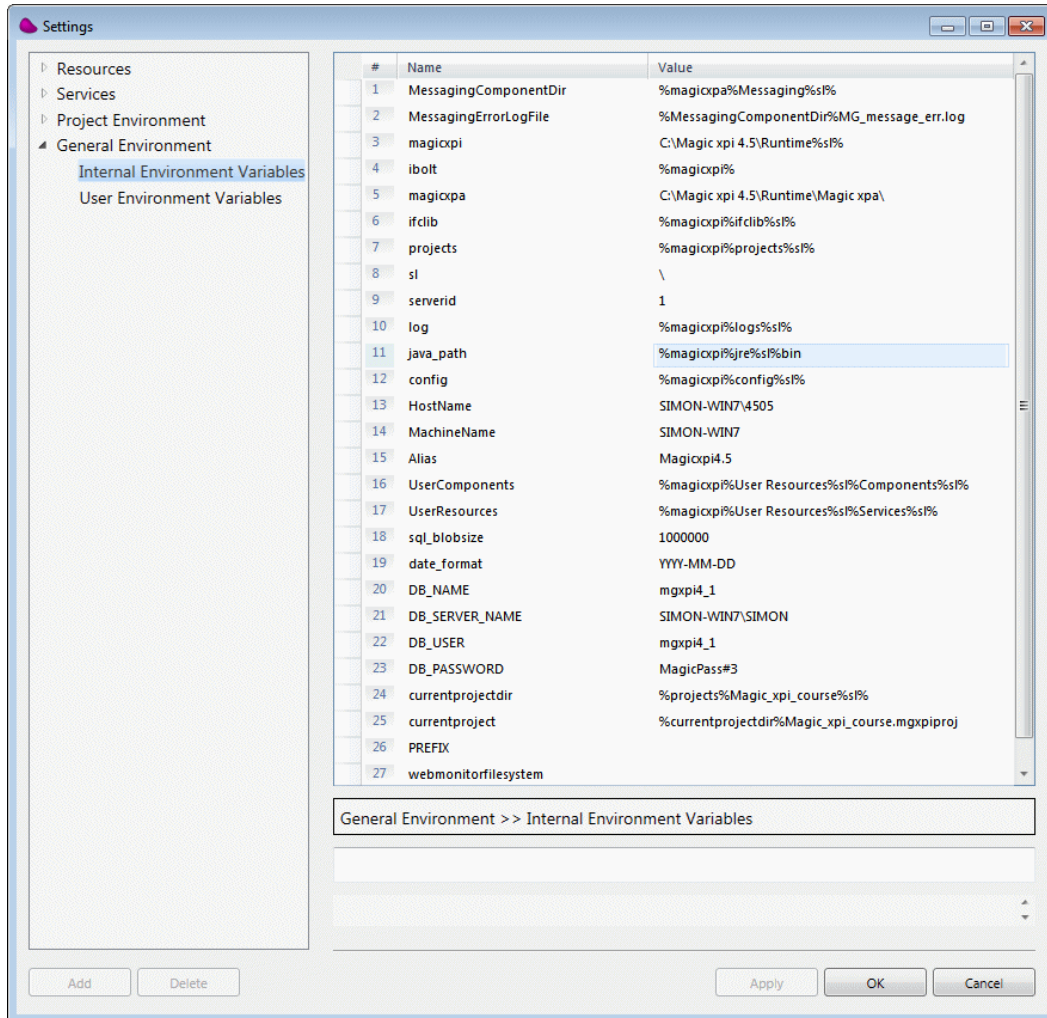
You will learn more about environment variables during the course, and you will use them in your examples.

When creating a new Magic xpi project, a number of predefined environment variables are created for the project.

There are also directory-related variables:

currentprojectdir	<p>%projects%Magic_xpi_course%sl%</p> <p>This environment variable contains the path to the directory where the current project resides. By using this environment variable within the project, hard-coding directory paths can be eliminated.</p>
currentproject	<p>%currentprojectdir%Magic_xpi_course.mgxpiproj</p> <p>This environment variable contains the full location of the project that is currently loaded. This variable is updated by Magic xpi when the project is loaded.</p>

The screen below shows an example of the environment variables in the **Settings** dialog box's **Environment** section.

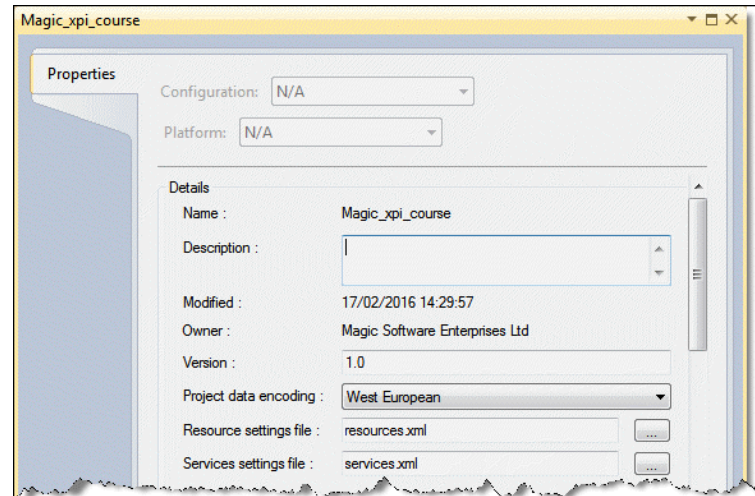




## Project Properties

A Magic xpi project has various properties, which can be configured to your requirements in the **Project Properties** window.

To open the **Project Properties** window from an existing project, select the **Project** menu and click **<Project Name> Properties**.



Property	Description
Name	The name of the project. This property cannot be modified in this window.
Description	A description of the integration project.
Modified	The last time the project was modified. This property cannot be modified.
Owner	The project owner (as specified in the Owner parameter of the INI configuration file). This property cannot be modified in this window.
Version	This property is maintained by the developer and is for information purposes only.
Project data encoding	The language encoding (the language and character set) that is used in the project file and in any destination XML file created in the Data Mapper.
Resource settings file	This property points to the location of the file holding the resource settings. This file can be edited externally to port the project to different environments.
Services settings file	This property points to the location of the file holding the service settings. This file can be edited externally to port the project to different environments.

## Course Project

During this course, you will build your own integration project with Magic xpi and will find solutions for the some of the common challenges that you are likely to face.

### Company Description

**MSU Computers Ltd.** sells computer hardware products.

There are several departments:

- **Purchasing** – This department purchases stock from vendors and manages the stock in the warehouse.
- **Sales** – This department handles customer relations and the approval of the customer orders.
- **Distribution** – After an order is approved, this department handles the delivery of orders to the customer.

### The Current Status

Currently, the company works with different systems and a lot of the work is manual:

- The Sales department receives a request for items in XML format.
- Each request is approved by the Sales department at two levels: financial and stock availability.
- The Logistics department verifies the stock availability by phone or email.
- After the request is approved, it is manually entered into the Distribution department's system and the items are delivered to the customer.

### Defining the “Challenge”

Recently, the company's management decided to integrate the systems and defined new goals. The goals are to:

- Enable a customer to register and to enter requests via the web.
- Decrease the cost of the order handling process and the post-sale process.

## The Proposed Solution

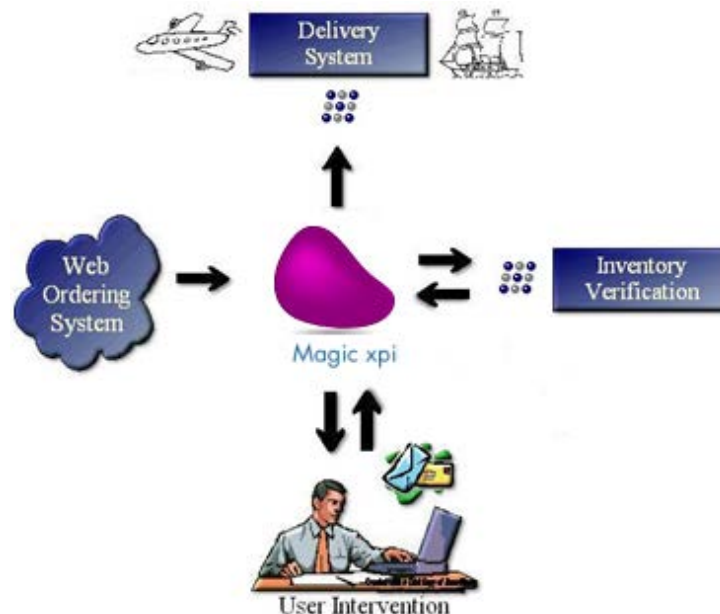
The company management investigated and considered several technologies and solution approaches, and decided on the following course of action:

- Connect all of the company's systems.
- Receive requests through the Website, and make them available on the company's system. This will prevent errors that result from duplicate information in both the system and the Website.
- Decrease the request handling duration, such as less typing, less human errors, and information available anywhere.
- Define an "Exception Approval" process for the Sales department. This process will approve orders when there is insufficient stock.
- Add an automatic email for notifications and approval requests.

## Project Description

The course project handles a customer request in the following manner:

- Accept/check for new customer requests.
- Check for stock availability within the local database.
- Have a human intervention step (if required).
- Process the request and update the delivery process.





## Summary

In this lesson, you learned about the Magic xpi project, including:

- Opening the Magic xpi Studio
- Creating a new integration project
- Environment variables
- Project properties
- The challenge of the course project



# Lesson 4

## Resources

Before you can actually start and define an integration flow, you need to identify and define the different resources that will be communicating with the project. Magic xpi provides a central location to define all the external resources referenced in the project. This is the **Resources** section of the **Settings** dialog box. The **Settings** dialog box can also be opened from the **Start** menu in stand-alone mode.

The **Resources** section defines all the external systems that Magic xpi needs to access during project execution.

Each resource contains the configuration details needed for a successful connection to an external application. For example, if you need to connect to a specific database, you only need to configure the connection details once.

## Advantages

Magic xpi provides a central location to define all the external resources referenced in the project. This approach has many advantages.

In the code itself, you refer to the resource name. You can refer to the same resource many times in your project, while the configuration details are stored only once.

The resource configuration details are stored separately from the integration flow logic. This allows easier administration, deployment, and migration between environments without changing the code.

Resources can be defined once and shared among multiple integration projects. This is particularly common in situations where resources such as Email or databases are defined.



Before continuing, take a step back and think which resources you will be using in the course project.

## Resource Types

Before you discuss the resources themselves, you must know about resource types.

Resource types are a predefined category of resources that share the same type of properties. For example, **Database** is a resource type. You may have multiple databases defined in your project, using different DBMSs such as Oracle or MySQL, but all share the same or similar configuration properties, depending on the particular DBMS.

Resource types are predefined in Magic xpi, and the properties associated with each type are stored in the <Magic xpi 4.5 installation>\Runtime\resource\_types.xml file.

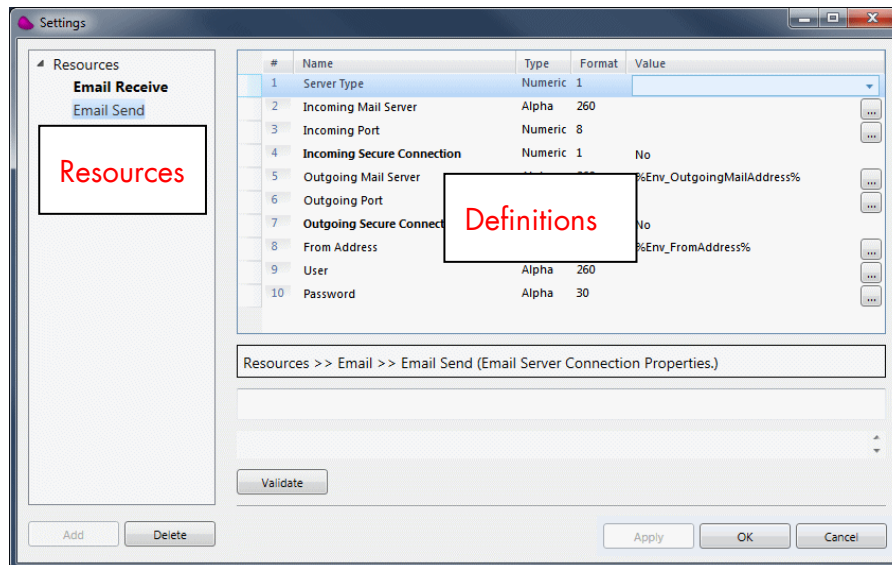


Although it is possible to modify these definitions, it is not recommended and should be done only by advanced developers.

## Adding a New Resource

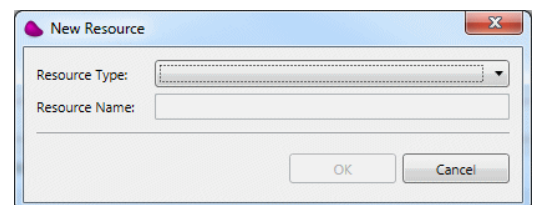
To access the **Resources** section of the **Settings** dialog box, click the **Project** menu and select **Settings**.

The **Resources** section is divided into two panes. In the left pane, there is a list of all defined resources grouped by resource type. In the right pane, you can find the configuration properties of the currently highlighted resource.



To add a new resource:

1. Click **Add**.
2. The **New Resource** dialog box appears.
3. Select **Email** from the **Resource Type** dropdown list. You will use this resource later on to send notifications to the salesperson.
4. Set the resource name to **Company mail server**. This name is the internal name to reference this resource throughout the project. It is recommended to use a meaningful name.
5. When you are done, click **OK**.



You can change the resource name at any stage by parking on the resource and selecting **Rename** from the context menu.

Once the resource has been added, the resource name will show in the left pane and a list of properties will appear in the right pane. As mentioned before, the number and type of properties are unique for each resource type.

You can now set the properties for the resource, in this case the **Email** resource. The properties that may be set for an **Email** resource are:

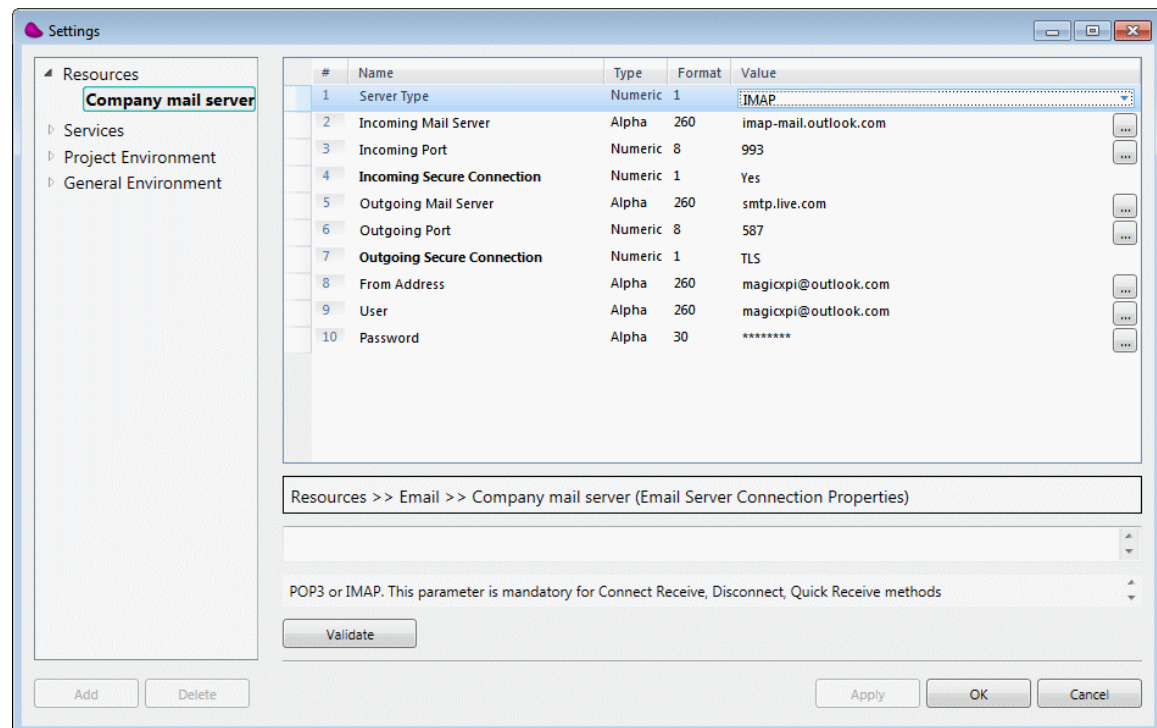
Property	Description
Server Type	This could be either POP3 or IMAP for incoming mail. Leave the property blank for outgoing mail. For the purpose of this course, use <b>IMAP</b> .
Incoming Mail Server	This is the name of the mail server, such as <b>mail.example.com</b> .
Incoming port	The port that the mail server is listening to. For secured connection, the default is <b>110</b> .
Incoming Secure Connection (SSL)	Specifies whether the communication with the incoming mail server will be established over a secured connection.
Outgoing Mail Server	The name of the SMTP server, such as <b>smtp.example.com</b> . This is the only protocol supported for sending email.
Outgoing Port	The port that the SMTP server is listening to. Typically <b>25</b> .
Outgoing Secured Connection (SSL)	Specifies whether the communication with the SMTP server will be established over a secured connection.
From Address	Enter a valid email address. This is the address that emails are sent from. For example, <b>magicxpi@outlook.com</b> .
User	User name for the mail server authentication, such as <b>magicxpi@outlook.com</b> .
Password	Password for mail server authentication. The characters entered are masked for security reasons.



If a Resource property is in **bold** font, it is a mandatory property.



When you have defined the **Company Mail Server** resource, your **Resources** section should look similar to the image below.



After entering all the relevant information, it is highly recommended that you check the accuracy of the information you entered by clicking **Validate**. This is very important, as it verifies the validity of your newly-created connection.

When you click **Validate**, Magic xpi connects to the resource using all the parameters that you have supplied. If the connection is valid, you will get a dialog box showing that the connection was successful. However, if the connection fails, you will get an error message that explains why it failed.

## Exercise

Here you will practice what you learned in this lesson. You are going to create a resource that you will need in later lessons.

- **Database Resource** – This resource will connect to the course data in MSSQL.
  - The resource name is **Course Database**. You will refer to this during later lessons.
  - Configure all details according to your MSSQL setup.
  - The course database is **Magic\_xpi\_course**.



To be able to complete the course project successfully, you need to load the supplied database tables into the MSSQL Server that is installed on your machine. To do this:

1. In the <My Documents>\Magic\projects\Magic\_xpi\_course\Magic\_xpi\_course\course\_data\DB\MSSQL folder, open the **runsql.bat** file for editing. Edit the file as follows:
  - Replace %1 with your database user name.
  - Replace %2 with the database server name.
  - Replace %3 with your database user password.
2. Save and run the **runsql.bat** file.
3. In the <My Documents>\Magic\projects\Magic\_xpi\_course\Magic\_xpi\_course\course\_data\DB\MSSQL folder, double-click the **logsql.txt** file to verify that the database tables have been loaded properly.

## Summary

In this lesson you learned about the **Resources** section of the **Settings** dialog box.

The **Resources** section provides you with a single location to configure the external resources, making it easier to maintain them within the project.

You created a number of resources that you will use during the course.

# Lesson 5

## Scan for New Requests

Now that the analysis has been completed, and you know how the business processes will flow in the project, it is time to start writing the integration flows. You will be taking the business process model and turning the logical flows into actual flows.

The Flow Editor is the main editor that you will be using in the Magic xpi Studio. With the Flow Editor, you create the actual business processes. These consist of the flow components and the flow logic.

In this lesson, you will learn about the integration flows. You will also be introduced to the components that make up the flows.

This lesson introduces various topics, including:

- The Flow Editor
- Creating a flow
- Flow properties
- Flow components
- The Directory Scanner component
- The Email component

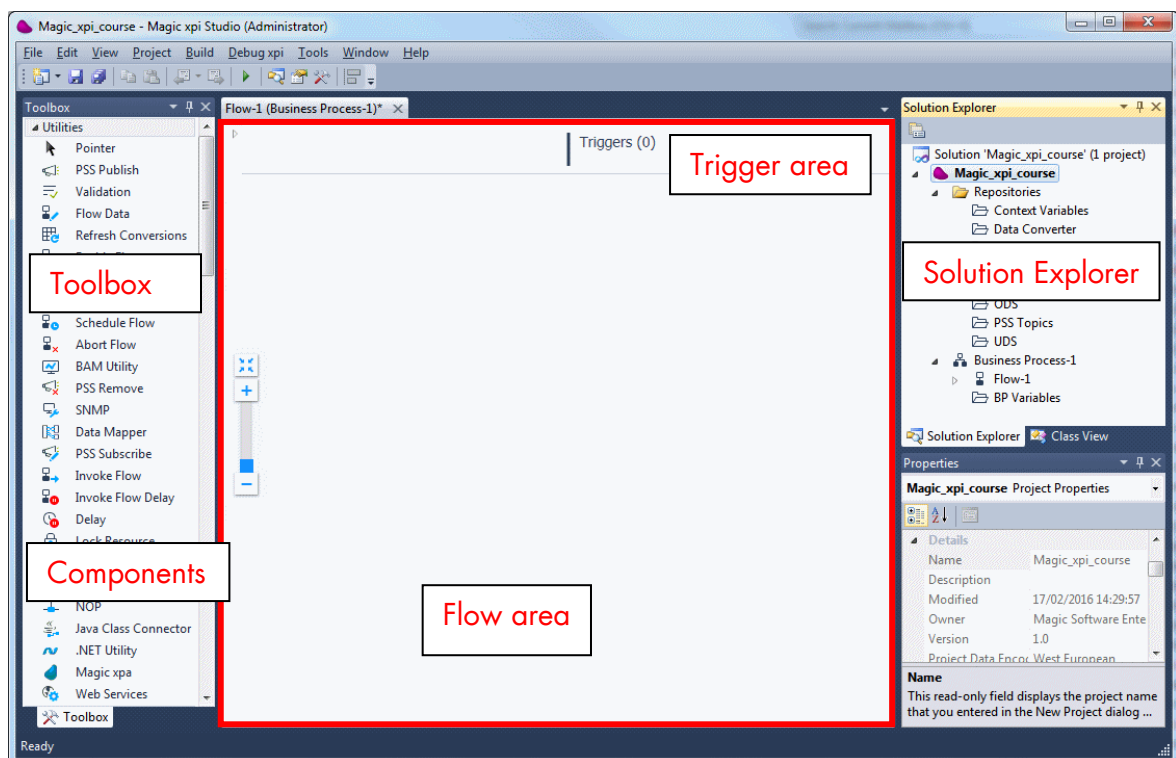
## Flow Editor

The Magic xpi Studio consists of three panes (see image below):

- **Solution Explorer** – This pane displays all of the projects, repositories, business processes and flows that have been defined.
- **Toolbox pane** – This pane contains a list of components, grouped together under various categories, that are available to be used in the integration flow.
- **Main pane** – This is the Flow Editor, and holds all the logic to process the integration project.

The Flow Editor consists of two different areas (see image below):

- **Trigger Area** – This is an area that defines when and what flow is called. You will learn more about this later in this course.
- **Flow Area** – This area contains all the logic that connects the components together. You will be doing most of your development here.



## Defining a Magic xpi Flow

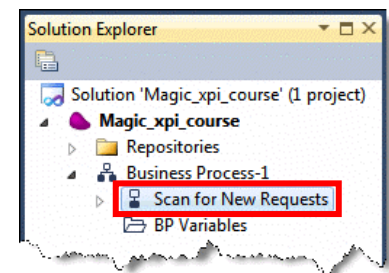
When you create a new project, Magic xpi automatically builds an initial default flow, named **Flow – 1**. You can use this flow to suit the needs of your integration project.

### Renaming Flows

The Magic xpi flow naming convention is provided as a default. It is good practice to give the flow a meaningful name.

To rename the flow:

1. In the Solution Explorer, park on **Flow-1**.
2. Right-click to access the context menu, and select **Rename**.
3. Type **Scan for New Requests** as the new name for the flow.



### Creating a New Flow

You may add as many flows as needed to the business process, with each flow showing the physical representation of the business logic. You will add more flows later in this course.

To add a new flow to the project:

1. Right-click on the **Business Process** where you want to add the flow.
2. From the context menu, select **Add Flow**. A new flow will be created with the initial name **Flow-2**.

## Flow Properties

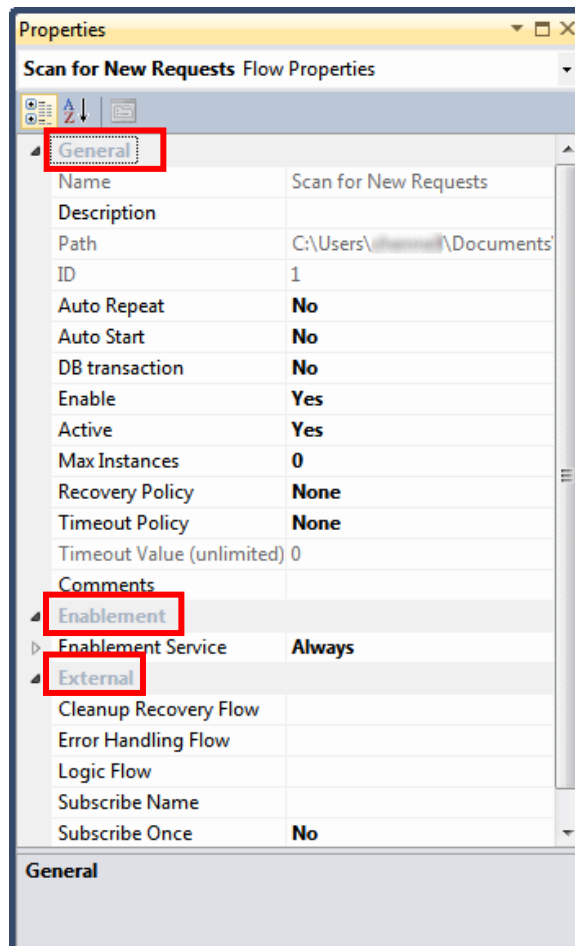
Each flow that is created has a set of properties that can be set for it.

To access the flow's **Properties** pane:

1. Park on the **Scan for New Requests** flow.
2. Right-click to access the context menu and then select **Properties**.



The flow's **Properties** pane is displayed. It contains the following sections:

- **General** – You can set general information about the flow.
- **Enablement** – You set information about when this flow is available. You will learn more about this in a later lesson.
- **External** – You set information about what happens during certain scenarios. You will learn more about this in a later lesson.



## General Section

In the **General** section, there are various properties that can be set:

Property	Description
Name	This is the name of the flow.
Description	A detailed description of the flow for documentation.
Path	A read-only parameter showing the location of the flow file.
ID	This is an internal identifier for the Server.
Auto Repeat	<p>Determines whether the Magic xpi Server will process the flow again when the last flow component is completed. The options are <b>Yes</b> or <b>No</b>. This can create a loop effect.</p> <p><b>Note:</b> When marking a flow as Auto Repeat, the  icon will appear in the Trigger area.</p>
Auto Start	<p>This determines whether Magic xpi will invoke the flow when the Magic xpi Server loads this project. The options are <b>Yes</b> or <b>No</b>.</p> <p><b>Note:</b> When marking a flow as Auto Start, the  icon will appear in the Trigger area.</p>
DB Transaction	Transaction support according to the availability of the component. Here, you select the resources that will take part in the transaction.
Enable	<p>Select <b>Yes</b> or <b>No</b> to enable or disable the flow.</p> <p>When you disable a flow, the flow is not executed when the project is deployed. You can enable a flow during runtime either by using the Enable Flow utility or from the Monitor.</p>
Active	<p>Select <b>Yes</b> or <b>No</b> to activate or deactivate the flow.</p> <p>You can improve the Checker's performance by marking a flow inactive. The flow is then omitted from the Checker process, and also from the deployment process.</p>
Max Instances	<p>Enter the maximum number of flows that Magic xpi can process simultaneously, including the first flow.</p> <p>Leaving the default as zero means that the number of flow</p>

Property	Description
	instances is unlimited.
Recovery Policy	Determines the flow's recovery policy when an error occurs.
Timeout Policy	This property lets you define what will happen once the timeout is reached. The options for the Timeout policy are: <b>Abort</b> , <b>Restart</b> , or <b>None</b> .
Timeout Value (Unlimited)	This property lets you define how long the flow can run before Magic xpi will end it. If you are connecting to an external system, this policy should be set to prevent the system waiting indefinitely for an answer.

## Introduction to Variables

During the invocation of a flow, it is often required to have a temporary storage place for the information. This means that the information can be referred to, and used by, the integration flow, or it can be sent to another integration flow.

Magic xpi predefines some variables that you can use, or you can create your own. You will learn more about the different types of variables in the next lesson.

For the purpose of this exercise, you will be introduced to a few of the Magic xpi predefined **context** variables. You will learn more about variables in later lessons.



The minimum you need to understand about context variables at this stage is that they are created when a new context is launched; in this case the flow is invoked. They are valid as long as the context is alive. They are defined once for all the flows in the project.

The following Magic xpi-generated context variables that you will be using for this exercise are:

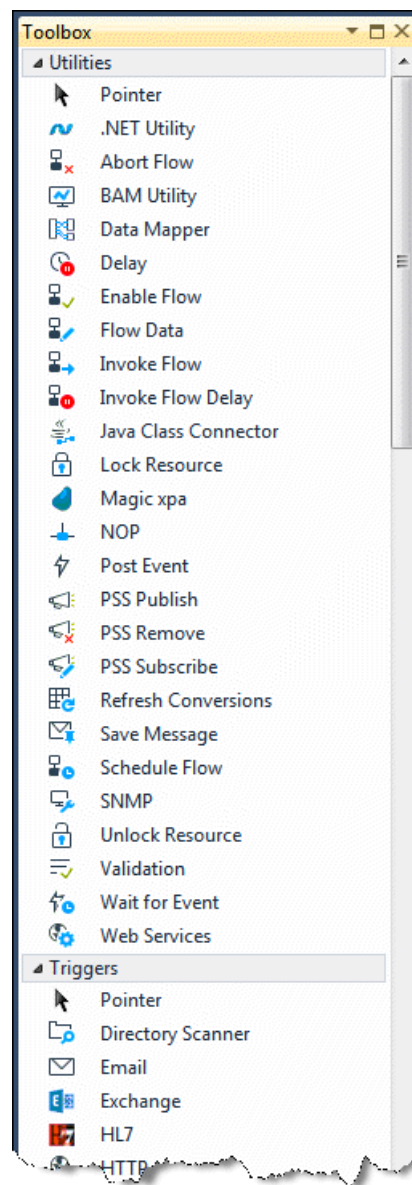
- **C.UserString** – String information up to 1000 characters
- **C.UserCode** – Numeric information up to 12 digits
- **C.UserBlob** – Binary information
- **C.UserXML** – Text information



## Flow Components

Magic xpi provides many components that you can use in your integration project. Components have many different functionalities, including, but not limited to connecting, adapting, converting, and processing.

When you park on a flow, you can view the available components in the **Toolbox** pane, grouped under the relevant headings for their type.



## Directory Scanner Component

According to the business process logic, requests were sent to the system and were placed in a certain directory on the Server. Then, a process would scan to see whether there are any requests in the directory and if there were, it would continue with the flow.

To meet this requirement, you will use the Directory Scanner as the first step in the **Scan for New Requests** flow.

The Directory Scanner component checks local area network (LAN) and/or FTP directories to see whether files are located in the scanned directory or subdirectory.

The component can also automatically perform specific actions on the specified files in the directories scanned by the component. You can also exclude a group of files or a specific file from being watched.

You can use the component to perform various actions, such as:

- Move
- Delete
- Rename

The component can work in two modes: **Step** mode or **Trigger** mode.

- **Step** – Once the first file satisfying the configuration criteria is found in the specified time period, or once the time period has elapsed, the required action is performed.  
For example, you can set a timeout for the component of one second. The Directory Scanner will scan the directory for the relevant file until the file is found or until it exceeds the defined timeout (one second, in this example).
- **Trigger** – The component is activated by the Server, and keeps scanning the directory for new files as long as the activating Server is running. You will learn more about triggers in a later lesson.



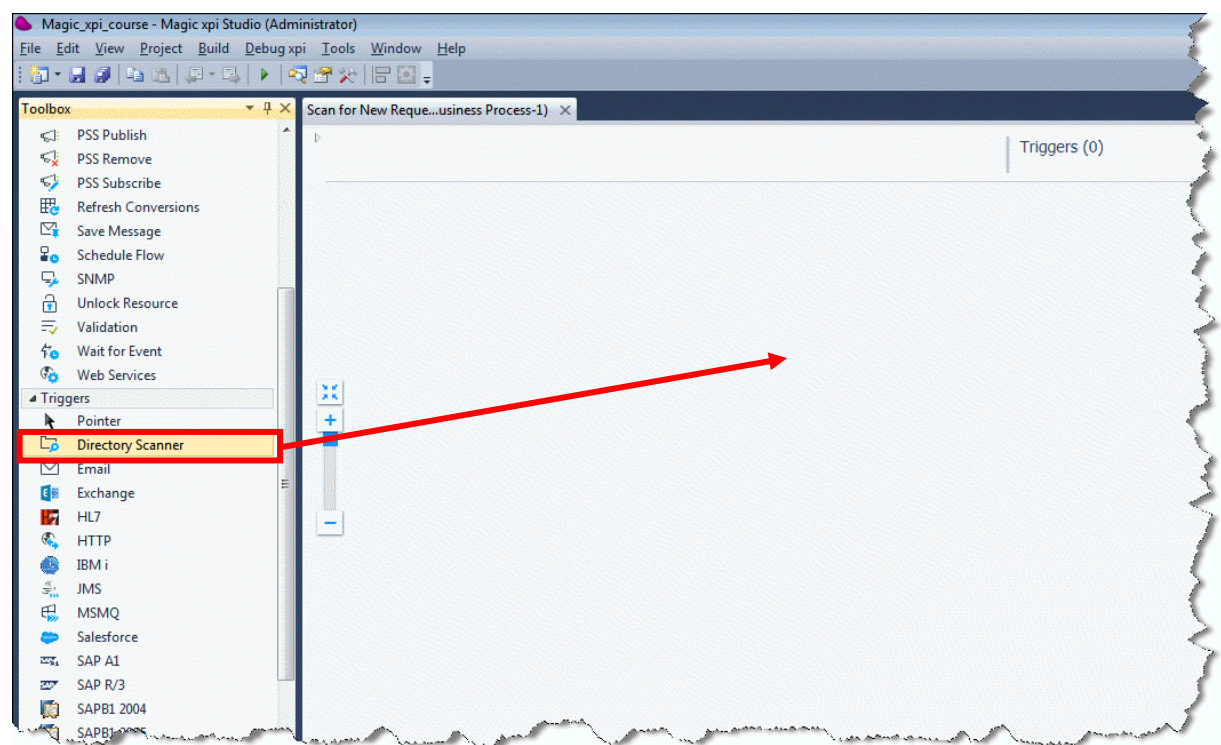
You will be using the Directory Scanner in step mode in this example. However, you should note that in the case of this example it would be better to use the Directory Scanner in Trigger mode.

## Using a Component

To use a component from the Toolbox pane, you will need to drag the component you wish to use and drop it into the Flow Editor pane.

You will add the Directory Scanner as the first step in the **Scan for New Requests** flow. To do this:

1. Click on the Directory Scanner component.
2. Hold the left mouse button down, move the cursor to the Flow Editor pane, and release the mouse button.



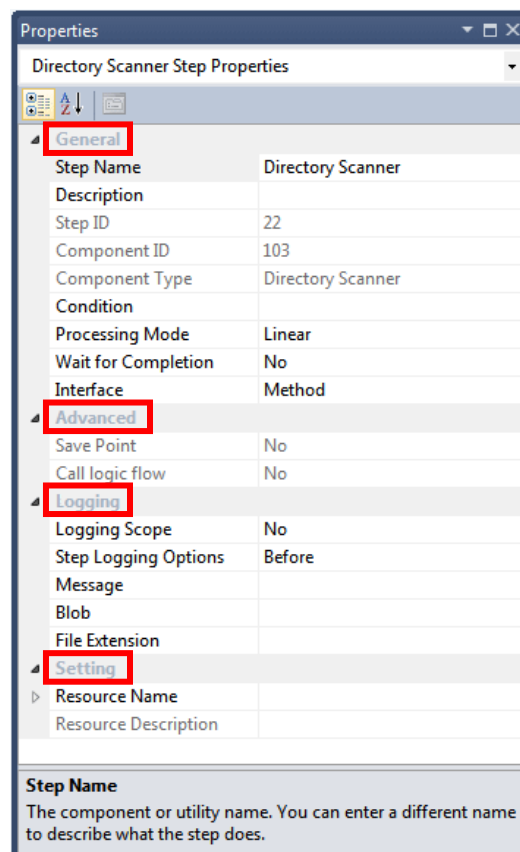
If components already exist in the Flow Editor, then you need to drop the component **on** the component that will be executed **before** this component. Magic xpi will automatically place the new step as the next step in the flow.

## Component Properties

Once you have placed the Directory Scanner component in the Flow Editor, its properties are displayed in the **Properties** pane. For each component, you will need to configure the component to suit the needs of the project. Magic xpi enables you to use the same component multiple times within the flow. You simply need to configure each component for that particular step.

The component's **Properties** pane has four sections:

- **General** – Contains general information about this step, such as the name of the step.
- **Advanced** – This covers features such as two-phase processing, and other more advanced properties. You will not learn about this section during this course.
- **Logging** – This is useful for debugging purposes. You will learn more in a later lesson.
- **Setting** – Contains the connection information as fetched from the resource.



In the **General** tab, there are various properties:

Property	Description
Step Name	The name of the component. This is the name that will appear on the Flow Editor and in the logs. It is good practice to provide a meaningful name.
Description	Detailed description of the component, describing what the component does.
Step ID	A read-only field displaying the ID of the step or trigger in the flow.
Component ID	This is an internal identifier for the component.
Component Type	A read-only field displaying the name of the step type.
Condition	A condition that specifies the flow's execution behavior.
Processing Mode	This will determine how Magic xpi will execute the step. The options are: <ul style="list-style-type: none"> <li>• Linear</li> <li>• Parallel</li> <li>• Stand Alone</li> </ul>
Wait for Completion	This will determine if Magic xpi will wait before continuing to the next step, if there are parallel steps defined.
Interface	This is the interface that will be used with the component: <ul style="list-style-type: none"> <li>• XML – Configured using a pre-created XML file.</li> <li>• Method – Configured using the <b>Direct Access Method</b> dialog box.</li> </ul>

Set the following properties for the **Directory Scanner** component you dragged:

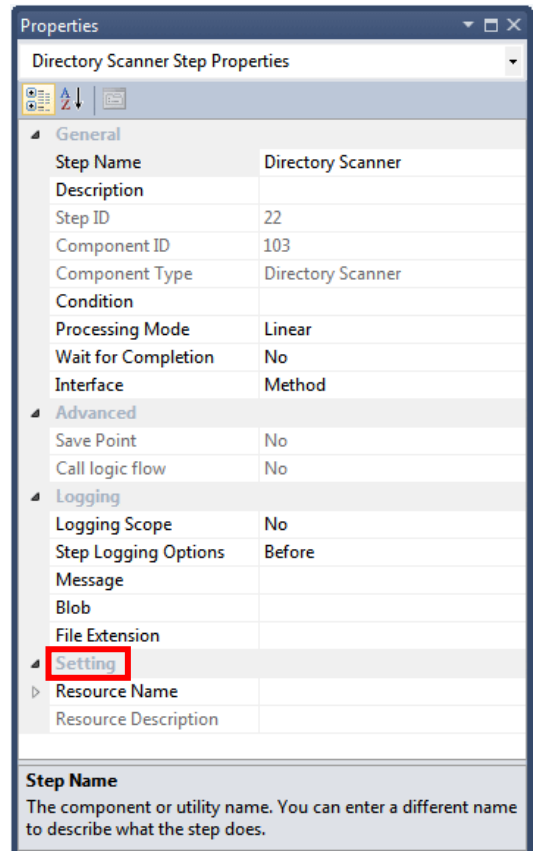
3. Set the **Step Name** property to **Scan Directory**.
4. Set the **Interface** property to **Method**.

In the **Setting** section, you define which resource contains the connection settings.

**Resource Name** – Displays the list of resources that you defined in the **Resources** section of the **Settings** dialog box. You can define a **Dynamic** resource by using an expression. You will learn more about expressions in a later lesson.

If no resources exist, you can add one by selecting **New** from the **Resource Name** property's dropdown list.

**Note:** For a Directory Scanner component, you will only need to configure a resource when working with FTP.



## Component Configuration

Every flow component needs to be configured to let Magic xpi know what needs to be done for the current step.

If the component has not yet been configured, you will see a red wheel symbol in the top right corner of the component.



To configure a component, either double-click on the component or right-click on it and select **Configuration** from the context menu.

In this lesson, you will learn how to configure the **Directory Scanner** component using the Method interface, also known as the Direct Access Method.

There are two sections to the method's configuration dialog box:

- **Method List** – This is a list of all the methods that will be processed during the step, in the order that they appear. You can use the arrow keys to the right to move the method either up or down in the list. You can also enter an expression to set a condition as to whether the step will be executed or not.
- **Method Parameters** – There are various parameters that are necessary to perform the highlighted method. The parameters that are in bold are required. Each entry in the parameter list has the following properties:

Property	Description
Parameter Name	This is the name of the parameter.
Type	The attribute of the parameter, such as Alpha or Numeric.
Picture	This is the size and format of the parameter.
In/Out	This defines whether the specific parameter receives a value to pass to the method, or whether the method returns a value via this parameter.
Value	This is the value itself. A string value will be surrounded by single apostrophes ( ' ).

## Configuring the Directory Scanner Example

In this example, the Directory Scanner will scan for new a request in a particular directory, and move the file to another directory to be processed. This example will use the **LAN to LAN** method, which means move from one directory to another directory. To do this the following configuration must be made in the method's parameter section of the component.

1. In the **Direct Access Method** dialog box, click **Add** to add a method.
2. Select **LAN to LAN** from the combo box.

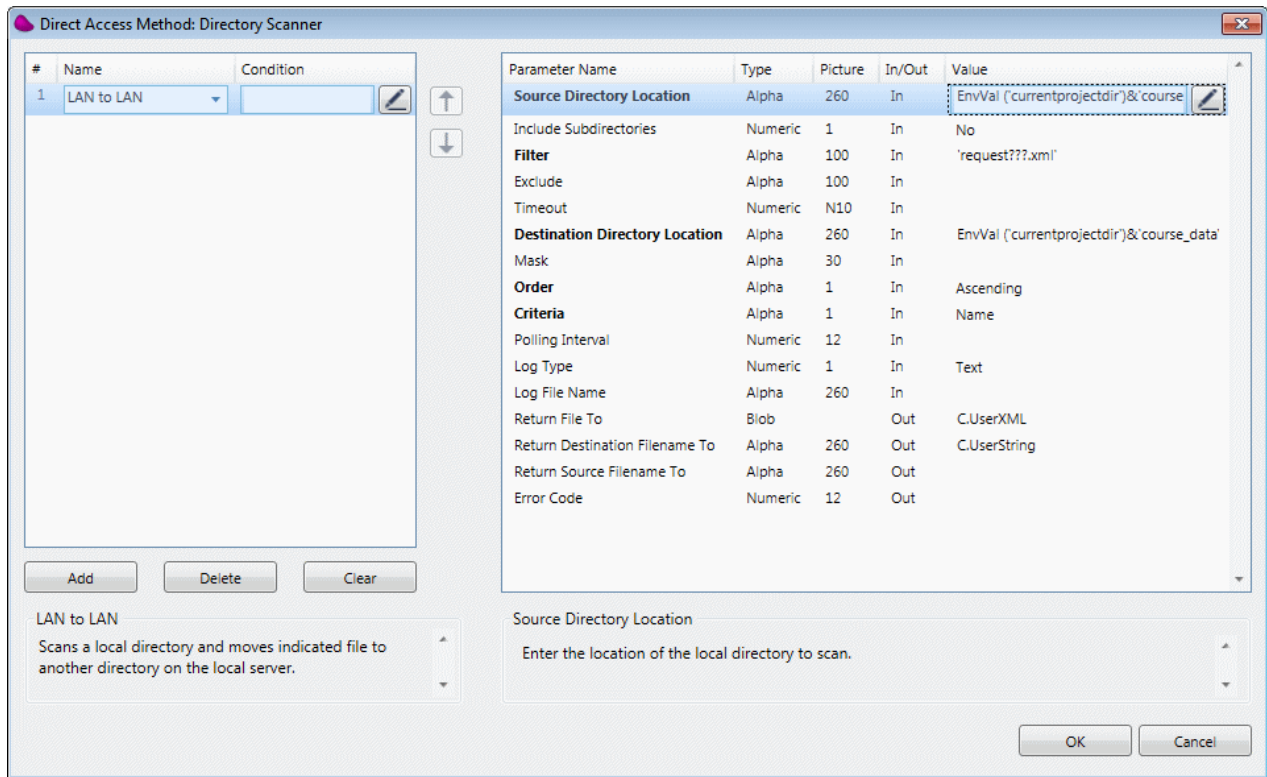
In Lesson 3, *Magic xpi Project*, you were introduced to environment variables and you learned about the **currentprojectdir** environment variable. This variable contains the full path of the current project. You will now learn how to use it.

3. Set the following properties:

Parameter	Value	Description
Source Directory Location	EnvVal ('currentprojectdir')&'course_data\in\ You will learn more about this expression in the next lesson.	The directory that will be scanned looking for new requests.
Filter	'request???.xml'	The name of the file that will be scanned for. The ??? represents wildcard characters such as: request1.xml, request999.xml
Destination Directory Location	EnvVal ('currentprojectdir')&'course_data\out\ '	This is the directory where the file will be moved to.
Order	Ascending	Defines the order in which the directory will be scanned.
Criteria	Name	Defines the search criteria. The options are Name and Size. For this example it will be looking by Name.
Return File To	C.UserXML	This is the name of the variable that the Directory Scanner will return the content of the file to.
Return Destination Filename To	C.UserString	This is the name of the variable that the Directory Scanner will return the name of the file to.



If you configured the component correctly, it will look similar to the image below:



## Exercise

Before you begin the exercise, there are some things you need to know. You will need to know what the email component is and you will need to know how to test your work.

As you have not yet been introduced to the Email component, it will be introduced here. For testing purposes, put the 'To' email as your own so that you will get the email when it is sent. This will be changed later in the course.

## Email Component

From within Magic xpi, the Email component can send and receive emails with attachments. The attachments can then be saved on the local machine.

The Email component can:

- Send and receive multiple messages at the same time.
- Selectively filter emails received based on different criteria.
- View the received email messages on the Server.

For the Email component, you need to define an Email resource in the **Resources** section of the **Settings** dialog box. This information will be used by the component. You defined this resource in Lesson 4.

There are numerous methods that Magic xpi supplies for the Email component. In this lesson, you will be using the **Quick Send** method. This method:

- Opens the connection
- Sends an email message
- Closes the connection

The lesson exercise is:

1. Add an **Email** step that will email a person that a new request has arrived. Call the step **Send Email to Sales**. Use the **Quick Send** method.
2. Enter the following information for the email:
  - **To:** postmaster@magicxpi.com
  - **Subject:** A new request has arrived
  - **Body:** A new request has arrived.
  - Attach the original request to the email (Hint – look at the properties you defined in the Directory Scanner step.

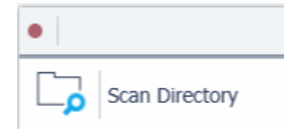
## Testing Your Project

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

Here you will get a very general overview of the Debugger. You will learn about the Debugger and the Checker in a later lesson.

To run your project with the Debugger, you will need to do the following:

1. Click on the **Scan Directory** step.
2. From the context menu, select **Breakpoint**.
3. A red dot will appear next to the step. A breakpoint means that processing will halt at that point.
4. Park on the **Scan for New Requests** flow and select **Debug** from the context menu.

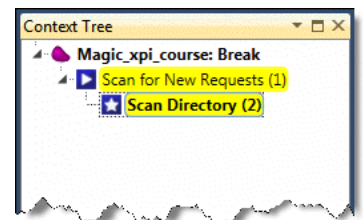


Magic xpi checks the project, in this case the flow, for any syntax errors. If there are syntax errors, you will not be able to continue. There are various types of syntax errors, such as a mandatory property that was not defined or was incorrectly defined.

After a few seconds, the Components pane will become the **Context Tree**.

5. Click the **Scan Directory (2)** option.

From the context menu, select **Step**. This will run each step. First the Directory Scanner step will run, followed by the Email step. You will have to repeat this step until Magic xpi finishes processing each step in the project.



6. Once you have finished processing all the steps, click the **Debug xpi** menu and select **Stop Debugging** to go back to development mode.

Now you can check your email inbox to see if the mail was correctly sent. As you already know, the Directory Scanner component updated two variables, **C.UserXML** and **C.UserString**. To view the data in these variables you can use the Context View. You will learn about this view in a later lesson.

## Summary

In this lesson, you learned about creating flows for the integration project, and how to assign properties to them. You then added two flow components. You also learned about:

- Adding a new flow to the project.
- Setting the properties of the flow.
- Adding components as a step in the flow, and configuring the step.
- Using the Directory Scanner component.
- Using the Email component.
- The Debugger.



# Lesson 6

## Flow Orchestration

The essence of any integration project is the transformation of information between systems and processes. When performing such transformation, data is retrieved, manipulated, and rearranged. During that process, it is often necessary to have a temporary storage place for the information. This means that it can be referred to, and used, by the integration flow.

You already learned about flows and components - the building blocks of the integration project. Now you will discover how Magic xpi decides which step to run within the flow, based on dynamic conditions.

This lesson covers various topics, including:

- Variables
- Flow logic
- The Expression Editor

## Variables

There are many different reasons for the information to be stored, such as:

- Calculating a new value based on one or more values.
- Changing the execution path of a process based on a value.
- Transferring values between different flow components.
- Providing information about the execution environment.
- Storing execution status information, user messages, and errors.

There are two categories of variables:

- **Environment variables** – These are maintained externally.
- **Project variables** – These are different types of variables to serve different purposes. In the previous lesson, you used context variables.

## Environment Variables

Environment variables, as their name implies, are primarily used to describe the physical operating environment of the project. This includes file locations, database names, and any other values that may change as the operating environment changes. As an example, for most projects the development environment is different from the testing and the production environments, and each may have its own server, database and other related configuration information.

An environment variable is a named placeholder for an alphanumeric value. The runtime engine will look for the value of an environment variable during execution. Whenever it encounters a name surrounded by the percent symbol (%), it replaces the value with its defined content.

For example, if you have a variable name **drive** with the assigned value **d:\**, then the string **%drive%FTP\IN** will be interpreted during runtime as **d:\FTP\IN**.

You can use environment variables recursively, meaning that you can use one variable name when defining another. Consider the following example:

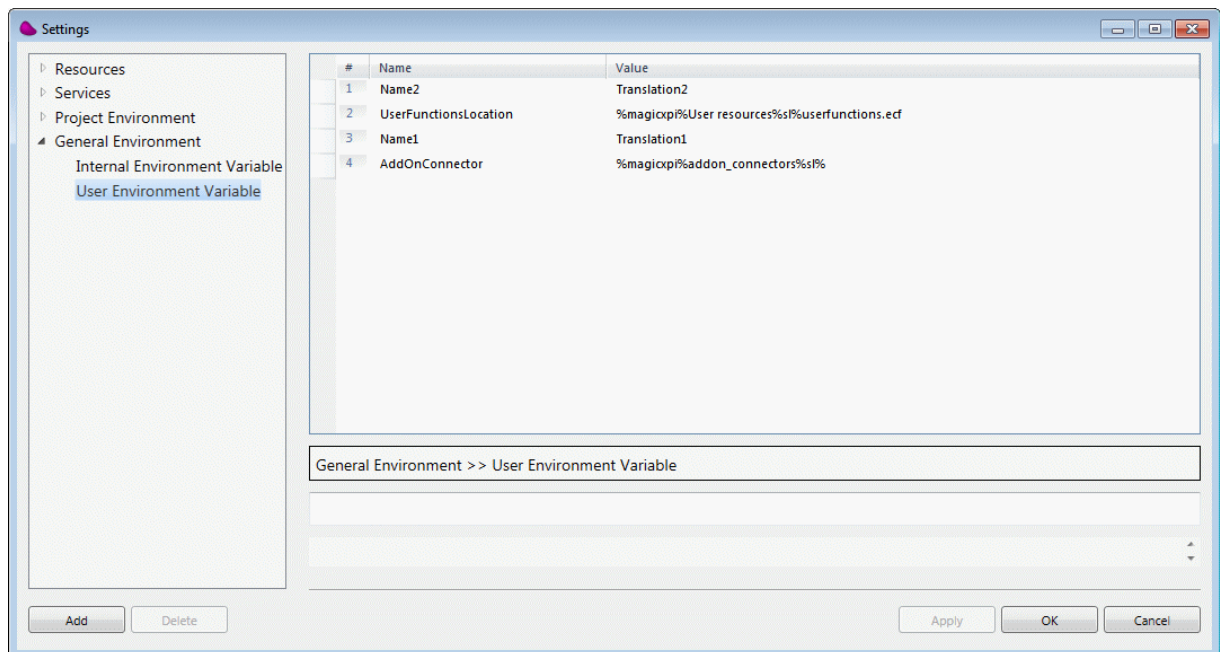
Variable Name	Defined Value	Interpreted value
sl	\	\
drive	C:	C:
datapath	%drive%%sl%data%sl%	C:\data\
Inbound	%datapath%in%sl%	C:\data\in\
Outbound	%datapath%out%sl%	C:\data\out\



Environment variables are case sensitive.

There are two types of environment variables:

- **User** – These variables are defined by the developer. You can add, delete, or define the translation value of those variables. When deleting or renaming variables, Magic xpi does not check whether the name is used in your code, and does not change your code to reflect the change.
- **Internal** – Magic xpi provides several predefined environment variables. You cannot delete these variables, but you can modify their value. One of the most commonly used predefined variables is **currentprojectdir**, which points to the physical folder in which your current Magic xpi project resides.



To create an environment variable called **source\_directory**:

1. Click the **Project** menu and select **Settings**.
2. In the **Settings** dialog box, click **Environment**, **General Environment** and then **User Environment Variables**.
3. Click **Add**.
4. In the **Name** column, type **source\_directory**.
5. In the **Value** column, type the directory that you want to point to.
6. Click **OK** to save.

**Note:** General Environment variables are stored in the **MAGIC\_LOGICAL\_NAMES** section of the **Magic.ini** file in your Magic xpi installation directory. Project Environment variables are stored in the project's **ifs.ini** file.

To modify the translation value of environment variables, you can open the **Settings** dialog box from the **Project** menu, or you can change the values directly in the configuration files mentioned above.



## Project-Specific Variables

Magic xpi provides four types of variables, each one valid in different areas: global variables, BP variables, flow variables, and context variables. These variables all have their own dedicated repository.

Each repository contains the following information:

Field Name	Description
Name	Variable name. Use meaningful names without spaces or special characters. Magic xpi adds a prefix based on the variable type.
Description	Free text describing what this variable is used for.
Type	Defines the type of information stored in the variable. Possible options are: Alpha, Numeric, Logical, Date, Time, BLOB.
Length	The maximum number of characters for Alpha variables. The maximum number of digits and decimal points for Numeric variables. Not applicable to other variable types.
Default Value	Define the initial value for this variable, if no other value is assigned to it. This can be a specific value, or an expression. Note: If an expression is used, only global variables are allowed with the expression text.

### Global Variables

The Global Variables repository is accessed by double-clicking **Global Variables** under the Solution Explorer's **Repositories** folder.

Global variables are available to all flows in the same project that share the same memory address. Changes made to global variables in one flow are reflected immediately anywhere this variable is referenced, throughout the project.

Magic xpi prefixes these variables with the letter **G** and a separating period. So if you named your variable **MyGlobalVariable**, for example, Magic xpi will call this variable **G.MyGlobalVariable**.

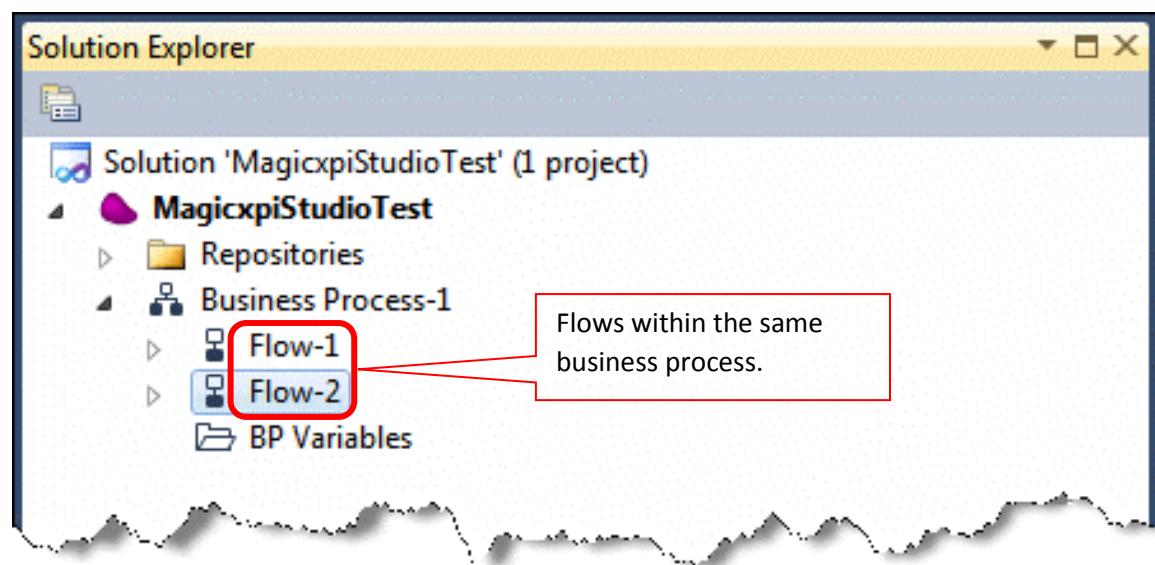
Magic xpi provides two predefined global variables.

- The **G.sys.ServerInstance** variable holds an identifier of the Magic xpi runtime engine, and is assigned a sequential number during runtime (i.e. 1 for the first engine, 2 for the second, etc.).
- The **G.sys.ComponentLogging** variable handles the control of the components' global logging process. You can use this variable to dynamically control the component logging. You will learn more about this variable in a later lesson.

## BP Variables

The BP Variables repository is accessed by double-clicking **BP Variables** under the relevant business process in the Solution Explorer.

BP variables are similar to global variables, since they can share the definition and the values among different flows. But, unlike global variables, BP variables are visible and can be used only within flows of the same business process.



Magic xpi prefixes these variables with the letter **B** and a separating period.

## Flow Variables

The Flow Variables repository is accessed by:

- Double-clicking **Flow Variables** under the relevant flow in the Solution Explorer.
- Parking on the relevant flow and pressing **Ctrl+L**.

Flow variables can be updated and referenced only within the flow they are defined in.

Magic xpi prefixes these variables with the letter **F** and a separating period.

## Context Variables

The Context Variables repository is accessed by double-clicking **Context Variables** under the Solution Explorer's **Repositories** folder.

Like global variables, context variables are defined globally for the entire project. But, as opposed to global variables, the value of the context variable is stored separately for each context. This means that when referring to the same variable name, you can have two different values if the reference is made from different flows. However a called flow runs in the same context, so the value of the variable in this case will be the same.

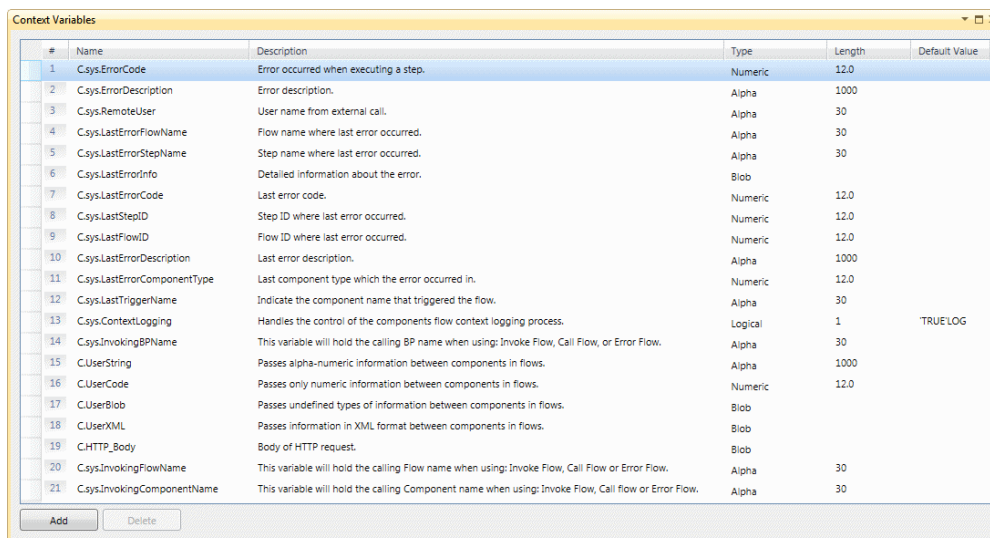
When you define a parallel step, this creates a new context. Therefore the same variable can have different values even within the same flow. You will learn more about parallel steps in a later lesson.

Updates made to a context variable are reflected only within the same flow that the update was made from.

Magic xpi prefixes these variables with the letter **C** and a separating period.

Magic xpi provides some predefined context variables that can be divided into the following categories:

- **User Variables** – Used to transfer information between components and flows. Many components update these variables with a special value. You used some of these variables in previous lessons. Avoid using these variables to store information, as their content may be overwritten by other steps.
- **Error Variables** – Updated automatically by the different components, and provide information about the last error occurred.
- **Flow Invocation Variables** – These variables hold the information about the invoking details, when a flow is invoked during runtime.
- **External Call Variables** – Used in external calls.



#	Name	Description	Type	Length	Default Value
1	C.Sys.ErrorCode	Error occurred when executing a step.	Numeric	12.0	
2	C.Sys.ErrorDescription	Error description.	Alpha	1000	
3	C.Sys.RemoteUser	User name from external call.	Alpha	30	
4	C.Sys.LastErrorFlowName	Flow name where last error occurred.	Alpha	30	
5	C.Sys.LastErrorStepName	Step name where last error occurred.	Alpha	30	
6	C.Sys.LastErrorInfo	Detailed information about the error.	Blob		
7	C.Sys.LastErrorCode	Last error code.	Numeric	12.0	
8	C.Sys.LastStepID	Step ID where last error occurred.	Numeric	12.0	
9	C.Sys.LastFlowID	Flow ID where last error occurred.	Numeric	12.0	
10	C.Sys.LastErrorDescription	Last error description.	Alpha	1000	
11	C.Sys.LastErrorComponentType	Last component type which the error occurred in.	Numeric	12.0	
12	C.Sys.LastTriggerName	Indicate the component name that triggered the flow.	Alpha	30	
13	C.Sys.ContextLogging	Handles the control of the components flow context logging process.	Logical	1	TRUELOG
14	C.Sys.InvokingBPName	This variable will hold the calling BP name when using: Invoke Flow, Call Flow, or Error Flow.	Alpha	30	
15	C.UserString	Passes alpha-numeric information between components in flows.	Alpha	1000	
16	C.UserCode	Passes only numeric information between components in flows.	Numeric	12.0	
17	C.UserBlob	Passes undefined types of information between components in flows.	Blob		
18	C.UserXML	Passes information in XML format between components in flows.	Blob		
19	C.HTTP_Body	Body of HTTP request.	Blob		
20	C.Sys.InvokingFlowName	This variable will hold the calling Flow name when using: Invoke Flow, Call Flow or Error Flow.	Alpha	30	
21	C.Sys.InvokingComponentName	This variable will hold the calling Component name when using: Invoke Flow, Call Flow or Error Flow.	Alpha	30	

## Flow Logic

Flow logic is a broad term referring to the set of conditions and rules that determines the execution path of your project.

Execution path can be determined by using:

- Conditions
- Logic flow
- The GoTo command
- Call Flow component
- Component properties (linear, wait for completion)
- Flow invocation (triggers, publish and subscribe, Scheduler)

Using these tools, you can create complex logic including branches, loops, and other flow logic that you may need. You will learn about these during this course.

### Steps and Branches

In the last lesson, you scanned a directory for a file and retrieved the file in one step. In a different, subsequent, step, you sent an email to the salesperson. This flow had two steps, one following the other.

Now assume that the requirement was to send a reply email to the customer to say that his request was received by the system and would be handled shortly, what would you do? You could add a second email step after the **Send to Salesperson** step. However, this will be inefficient as these two steps are independent of one another. Why can these steps not be executed in parallel? You can do this by creating a branch.

To create a branch:

- Select the Email component and drop it on the Directory Scanner component.
- You will now have two steps under the Directory Scanner step.

The Directory Scanner step is now the head of a branch.



You now have two steps. In which order will they be executed?

## Parallel vs. Linear Execution

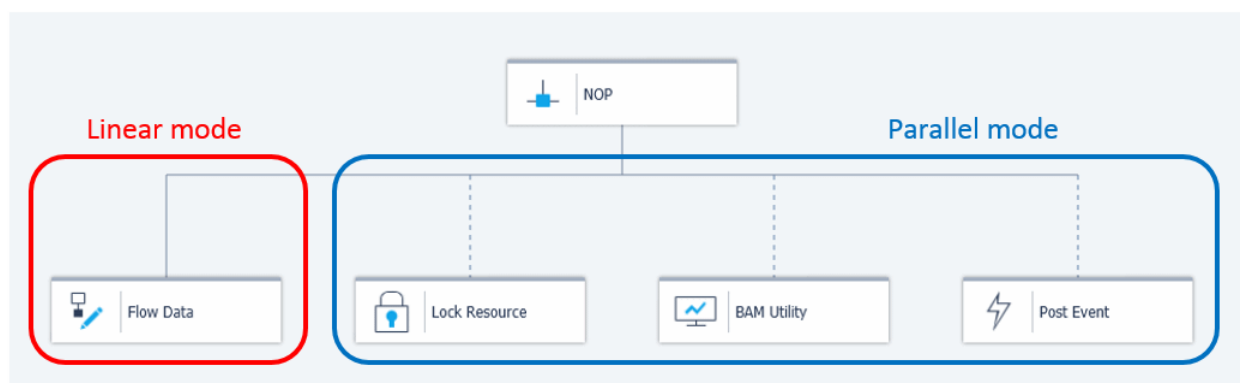
In Magic xpi, each component could have one of three possible processing modes:

- Linear
- Parallel
- Stand Alone

When steps are being executed in a linear mode, the Magic xpi Server executes those steps sequentially, one after the other. The execution of the next step does not start until the completion of the previous step.

There are situations when you will decide to execute more than one step at a time. There are several reasons for concurrent execution such as:

- **Performance** – If each step is executed only after the previous step has been completed, the flow will take a long time to execute.
- **Independent steps** – If a step is independent of a previous step, you can run these two steps concurrently.
- **Long executing step** – When a step takes a long time, such as a step that fetches data from a remote database, it will hold up other steps. You can, therefore, run other steps concurrently so that this step will not hold up execution.



In the Flow Editor, as you can see in the diagram, a linear connection is marked with a solid line, while a parallel connection is marked with a dotted line.



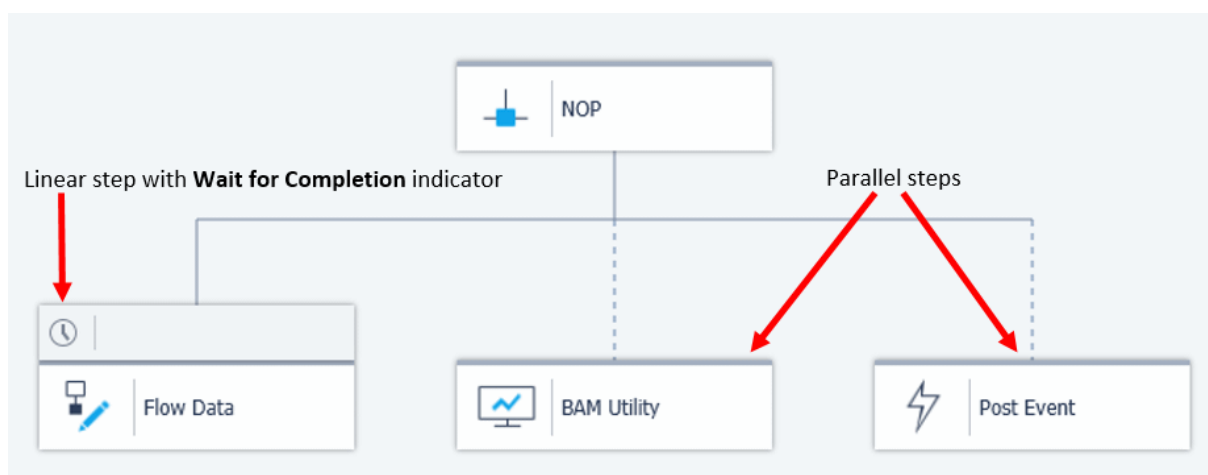
You can have more than one parallel step at the same level, and all parallel steps without a condition, or with a condition that when evaluated to TRUE will be executed. You may also have multiple linear components at the same level, but only one will be executed.

## Wait for Completion

When there are both linear and parallel steps on the same level, there are situations where the process needs to wait for the completion of the execution of some of the parallel steps before continuing to the next linear steps. This may be for various reasons, such as waiting for a returned variable from a running step.

To instruct the Magic xpi engine to wait for the completion of all other steps at the same level, there is a special **Wait for Completion** flag that you can set in the flow's properties. This flag can be specified only on steps with linear execution mode.

When the Magic xpi Server determines the next linear step to execute, it checks if the selected step has **Wait for Completion** set to **Yes**. If so, the Magic xpi Server will wait for the completion of all parallel steps that will process at this level. This excludes those steps with a condition that evaluates to FALSE. Also, parallel steps with an execution mode set to **Stand Alone** are also excluded from the **Wait for Completion** process.



When marked with **Wait for Completion**, a special icon will show just above the step on the Flow Editor (see image above).

## Determining the Next Step to Execute

When Magic xpi completes the execution of a step, it needs to determine which steps will be invoked next.

The Magic xpi Server scans all the non-linear (parallel or stand alone) components in the next level, and evaluates their conditions. The Magic xpi Server then executes all those non-linear components where the condition evaluates to TRUE, or no condition is defined.

Among the linear components, only one component will be executed. The Magic xpi Server scans all the linear components in the next level that have a condition defined. The scanning order is from left to right. If the condition of a linear component evaluates to TRUE, Magic xpi will execute this component.

If no linear component with a condition is defined, Magic xpi will look for a linear component without a condition. If one is found, Magic xpi will execute this component.

If Magic xpi does not find any linear component to execute, the flow is terminated.

If there is more than one component at the same level where the condition evaluates to TRUE, the condition with the leftmost component will be selected as this is the first that will be found.

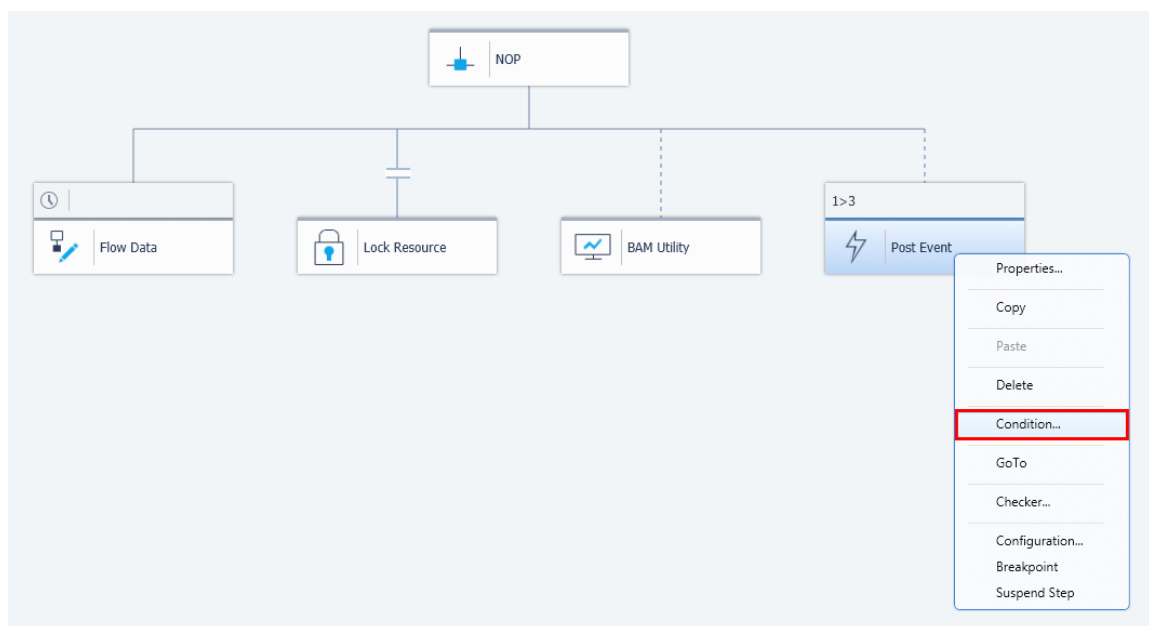
## Expression Editor

The Expression Editor is where you define expressions that return a specific value to the reference point where the Expression Editor was invoked from.

An expression in Magic xpi is comprised of variables, constants, functions, and operators.

When you want to specify a condition for the execution of the step, you need to define a Boolean expression that will evaluate to TRUE or FALSE at runtime.








To define an execution condition, right-click on the desired step, and select **Condition** from the context menu.



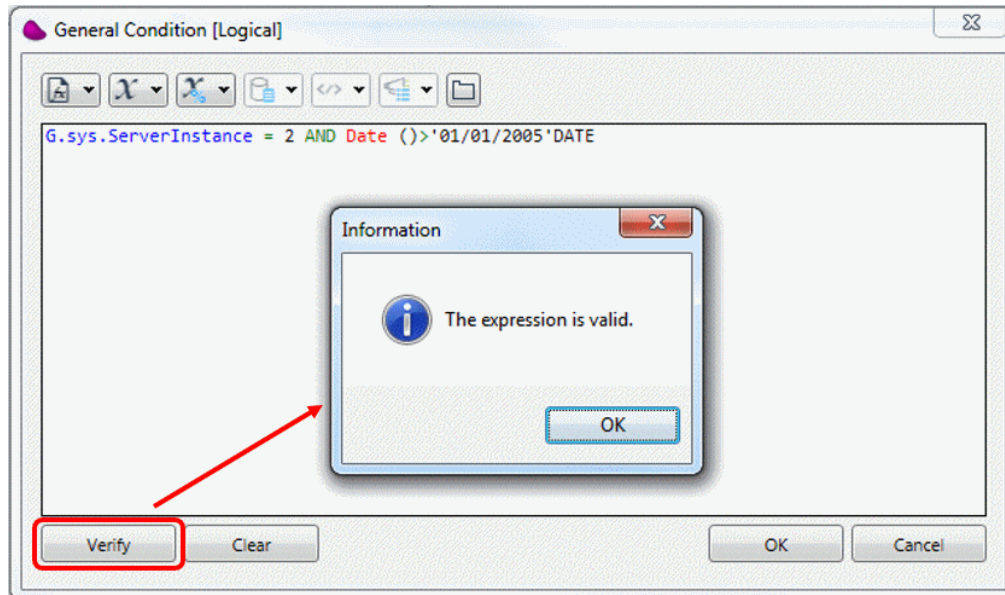
The **Expression Editor** dialog box will open and provide you with the tools to define the condition expression. Note that the dialog box's title will show in parentheses the expected type of the return value (in our example – Logical).



Just above the editing area, there is an icon toolbar. These are tools that can help you when constructing an expression:

-  **Functions** – Displays a list of functions with a short description of the highlighted function. Enables you to paste the function into the editing area.
-  **Variables** – Displays a list of variables, with the variable description and type. Enables you to paste the variable name into the cursor position in the editing area.
-  **Environment Variables** – Displays a list of environment variables with their current translation value. Enables you to paste the name into the cursor position in the editing area. Magic xpi will automatically add the **EnvVal** function to the selected name. This function returns the environment variable value.
-  **ODS** – Displays a list of all ODS entries defined. You will learn about ODS later in this course.
-  **Source Nodes** – This option is enabled only when you access the Expression Editor from the Data Mapper. This topic will be covered in the next lesson.
-  **PSS Topics** – Displays a list of the PSS variables that you defined in the PSS Topics Repository. These variables are used for working with Publish and Subscribe (PSS) utilities. PSS lets you define an event so that when the event occurs in your integration project, the event is published.
-  **Insert Filename** – Opens the standard Windows **Open** dialog box and enables you to select a file. The selected file is pasted to the editing area at the cursor point as a string literal.

When typing in the Expression Editor, you can type the first few letters of a function and press **CTRL+Space**. This will open a list of functions and the selected value in the list will be the one closest to the string you have typed. Like the assistants mentioned above, you can select the highlighted function and paste it into the editing area at the cursor position.



It is good practice to verify the validity of the expression before you close the **Expression Editor** dialog box.

## Exercise

1. Add two new variables to be used in the Directory Scanner component in place of the predefined context variables:
  - **F.RequestXML** – a BLOB variable which will hold the request itself.
  - **F.RequestFileName** – Alpha 255 variable.
  - Use these variables in the Directory Scanner.
2. Use a global variable for the Email component for the person who will receive the email. In the previous lesson, you hard-coded the **To** field with an email address. Now you will use a global variable to define the **To** email address. Enter the following information for the variable:
  - Name – **EmailTo**
  - Type – **Alpha**
  - Length – **100**
  - Default Value – **Salesperson's email address** (for example: **postmaster@magicxpi.com**)
  - Add this global variable to your Email step.
3. You do not want the Email component to process if the moving of the file was unsuccessful. Put a condition on the Email component, so that if the file name is blank, the flow will be terminated.
4. Add the following flow variables (you will need them in the next lesson):
  - **F.CustomerEmail** – Alpha variable, length 100
  - **F.CustomerName** – Alpha variable, length 30
  - **F.ContactName** – Alpha variable, length 30
  - **F.CustomerId** – Numeric, size 9

## Summary

In this lesson, you learned about the different variable types supported by Magic xpi, and about the flow logic that determined the actual execution path at runtime.

After completing this lesson you will be able to:

- Specify the different types of variables in Magic xpi.
- Understand the definition scope and data scope for each variable type.
- Explain how Magic xpi decides which steps to execute.
- Write an expression for a step condition.
- Specify a step's execution mode.



Magic®  
University



# Lesson 7

## Checking Customer Existence

A central part of any integration project is data transformation. In Magic xpi, one of the main tools used for data transformation is the Data Mapper. In this lesson, you will be introduced to the Data Mapper utility. This enables you to create associations between records and fields in different formats from different sources by using visual mapping. This utility is one of the most widely used utilities in the Magic xpi Studio.

Once the system receives a new request and the file is moved to the appropriate directory, the system then needs to validate the input. The first step in this process is to check whether the customer exists in the local database.

## Data Mapper

You can think of the Data Mapper as a tool that reads from a source and writes to a destination, a "Read/Write" scenario. You can have a number of different sources and a number of different destinations.

The Data Mapper utility is used for any of the following:

- Creating or updating files in XML, HTML, or flat file formats.
- Creating, updating, and deleting a set of database records.
- Calling a flow and passing variables.

Magic xpi enables you to map source data to destination data. Source data can be one of the following:

- XML
- Database (Select statements)
- Flat file (a text file or CSV file)
- JSON
- Variables
- ODS or UDS entries

Destination data can be one of the following:

- XML
- Database (Insert/Update/Delete statements)
- Flat file (a text file or CSV file)
- JSON
- Variables
- ODS or UDS entries
- Call flow
- Template (usually HTML or RTF)

The properties of the Data Mapper utility are the same as the other components learned about in the previous lesson.

Do the following:

1. Drop a **Data Mapper** utility as a child step of the **Send Email to Sales** step.
2. Name the step **Extract Details from Request**.
3. Double-click the Data Mapper utility, or right-click on it and select **Configuration** from the context menu to open the **Data Mapper** window.

## Data Mapper Window

The **Data Mapper** window enables you to configure the source objects and the destination objects.

You can have multiple sources and destinations. For each of the entries you will need to define the properties. Each set of properties will be different, depending on the object type. For example, Variable properties will be different from Database properties.

In the image below, you can see an XML file as a source and flow variables as a destination. When the source and the destination are configured and connected, and this step is executed, the Data Mapper will map elements from an XML file to variables.



The Data Mapper must have at least one destination object.

In the previous lesson, you used the **Directory Scanner** component to move requests that are received to a different directory for processing. To process that request, you will need to pull the request information from the XML file and put that information into variables that you can pass to a different step.

For this example, the source type will be **XML** and the destination type will be **Variables**.



Every type in the Data Mapper has a different set of properties. To set the properties for a specific entry, right-click on it and select **Show Properties**.



In the current lesson, you must define properties for the **XML** source and for the **Variables** destination. In most cases, the properties will be the same regardless of whether the specific type is in the **Source Tree** or the **Destination Tree**.

Do the following:

4. From the Toolbox's **Mapper Schemas** section, drag an XML source into the **Data Mapper** window's **Source Tree** area.
5. Right-click on the XML source and select **Show Properties**.
6. Enter **Request** in the **Name** property.





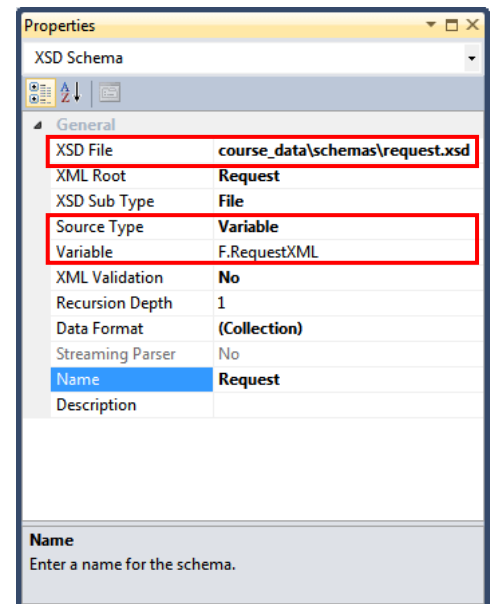
## XML Properties

Property	Description
Name	The XML Source's name.
Description	A detailed description of what the XML file is for.
XSD File	This is the path to the XML schema. You have to specify the exact path and file name for the XSD file.
XML Root	Select the XML root that you want to map data from. Note that you will only be able to park on this field when there is more than one root element in the XSD file. Magic xpi will default to the first root element that it finds.
XSD Sub Type	The Source schema's subtype. Select one of the following options: <ul style="list-style-type: none"> <li>File</li> <li>IFC Model</li> <li>XML Position Forwarding (Source only)</li> </ul>
IFC Model	This property lets you select a predefined XSD to use as XML data. This provides an easy way of using the XML Schema Definition (XSD) for any flow component.
XML Position Forwarding (Source only)	This option enables the XML Source data to start mapping from the saved start position, by calling the flow.  For example, if you have an order entry, where each XML order file includes one order header and multiple order items, the order header is handled in the main flow. The multiple order items are handled in a second (called) flow. If you select XML Position Forwarding, the called flow handles single order items instead of reprocessing the entire XML.
Source Type	There are various options for the source of the data: <ul style="list-style-type: none"> <li>File – This is an external XML file, where the data is being stored. If you select this, you can click on the  button in the <b>File Path</b> field to enter an expression for the file.</li> <li>Variable – This is a BLOB variable that the data is stored in. If you select this, you can click on the  button in the <b>Variable</b> field to get a list of valid variables.</li> </ul>
XML Validation	This validates that the XML is in a valid format according to the schema, before you start mapping. If the XML is not in a valid format, you will get an error.

Property	Description
Recursion Depth	This property will determine how many times the structure of the selected XML file repeats. Example: If a parent node in the XML structure contains three children, enter 3 to repeat the structure three times. For no recursion, leave the default of 1.
Data Format	This property lets you define the defaults that you want Magic xpi to use when presented with a certain data type.
Use Streaming Parser (Source only)	Select this check box to use a streaming XML parser. This allows the Data Mapper to handle large XML documents.

Do the following:

7. In the **XSD File** property, click the  button and select **request.xsd**, which is located under:  
**%currentprojectdir%\course\_data\schemas**
8. Select **Variable** in the **Source Type** property.
9. In the **Variable** field, click the  button and select **F.RequestXML** from the list.




## Variable Configuration

You are able to select which variables will be used in the subsequent Data Mapper screen. In this example, you will be mapping from the Request XML to defined variables.

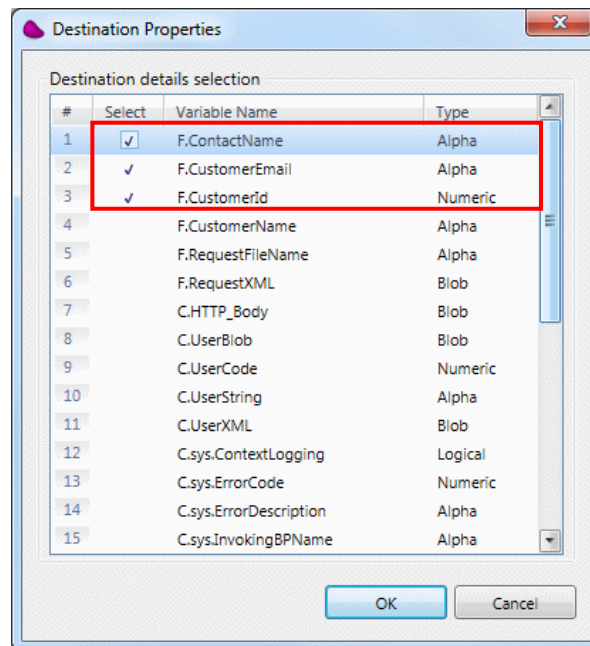
Do the following:

10. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Data Mapper** window's **Destination Tree** area.
11. Right-click on the Variable destination and select **Show Properties**. In the **Properties** pane, enter **Variables** in the **Name** field.

You will now select some variables that you defined in the previous lesson:

12. In the Variable destination's **Properties** pane, click the **Variables** field's  button.
13. In the **Destination Properties** dialog box, select the following variables:

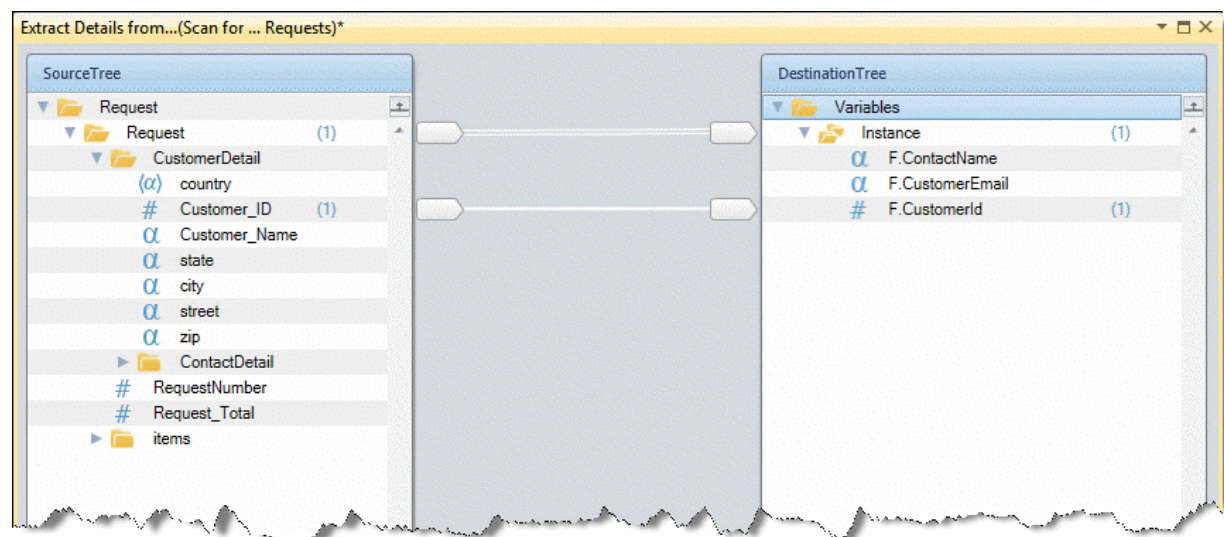
- **F.ContactName**
- **F.CustomerEmail**
- **F.CustomerId**



## Mapping Sources to Destinations

The last step for the Data Mapper is to map the request to the defined variables. To map:

14. Expand the source by clicking **Request**.
15. Expand the **Request** node.
16. Expand the **CustomerDetails** node.
17. Expand the destination by clicking **Variables**.
18. Expand the **Instance** node.
19. In the **Source Tree**, select **Customer\_ID**.
20. Right-click and select **Connect** from the context menu.
21. Drag the line to the **F.CustomerId** entry in the **Destination Tree**.

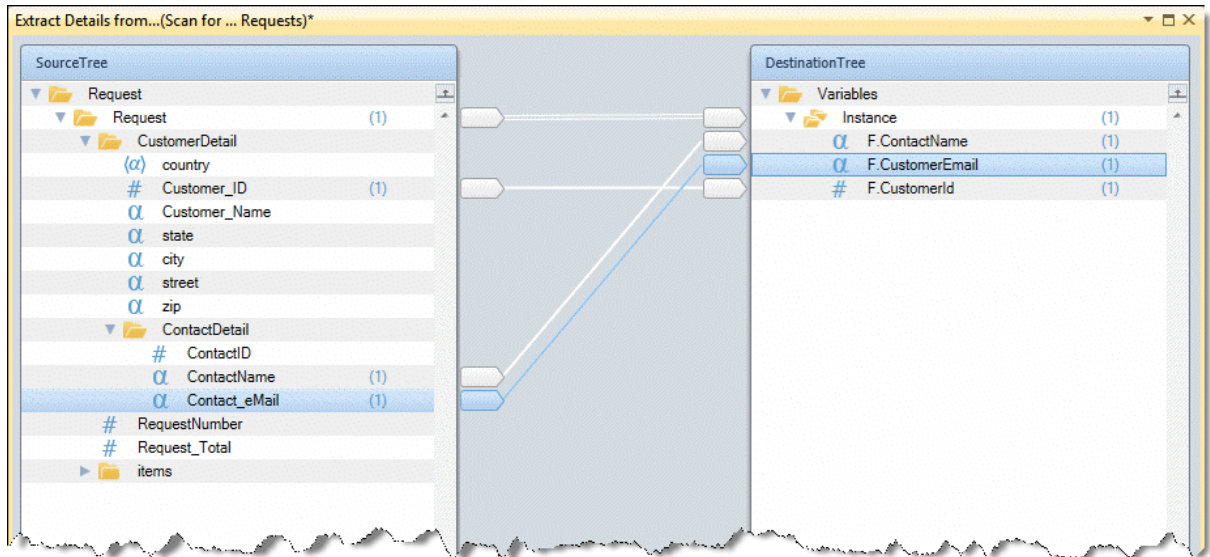


A white line will appear connecting the source to the destination. A double white line will appear connecting the request to the instance. This line denotes that for every occurrence found in the XML, it will be copied to the **F.CustomerID** variable. Note that a blue color for the connecting line simply denotes the connection currently in focus.



For example, there were three occurrences of the **CustomerID** in the XML. Therefore, Magic xpi would update **F.CustomerID** with a loop of the three values. But because this is a variable, only the last one would be saved in the variable.

22. In the **Source Tree**, expand the **ContactDetail** node.
23. Connect **ContactName** to **F.ContactName**.
24. Connect **Contact\_email** to **F.CustomerEmail**.



25. Right-click on the **Scan for New Requests** flow and then select **Properties**. In the **Properties** pane, set the **Auto Start** field to **Yes**.
26. Set a breakpoint on the **Extract Details from Request** step.
27. Run this flow using the Debugger. Once the Debugger reaches the breakpoint, you will see that the flow variables were updated with values fetched from the XML.

## Checking for Customer Existence

Now that you have pulled the customer information from the request XML, you can check whether the customer exists in the local database. You would not want to create an order if the customer does not exist. The next step for the integration project is to validate the customer data.

When working with data, you often need to use database tables. You fetch data from tables according to a set of rules and you write data to tables.

For example, after receiving a request with a Customer ID, you need to check whether that client is a customer in the local database and what the customer's credit rating is. If the client is not a customer, you will need to add the client to your database.

The Data Mapper utility provides the tools needed to work with databases.

A database object may be used as either a source or a destination object in the Data Mapper. However, as discussed earlier, the mapping of a source to a destination is a **Read/Write** scenario; therefore, not all database operations are valid in both trees.

The valid options are:

- **Select** – Source
- **Insert** – Destination
- **Update** – Destination
- **Delete** – Destination



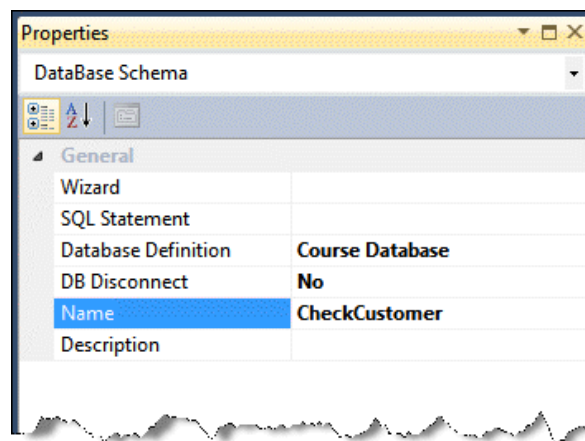
A database is an external resource, so it should be configured in the **Resource Repository** before continuing. You created the database resource in the **Resources** lesson.

Various MSSQL tables are used in this course. For information on the various tables and how they connect to one another, please look at the **Course Data** section, which you can find right before the **Solutions** appendix.

In this example, you want to check whether the customer exists in the database. This is an SQL Select statement. Therefore, the database is defined as the source in the Data Mapper.

To check the existence of the customer:

1. Drag a **Data Mapper** utility as a child step of the **Extract details from request** step. Name this step **Check if the Customer Exists**.
2. Double-click the Data Mapper utility, or right-click on it and select **Configuration** from the context menu to open the **Data Mapper** window.
3. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Data Mapper** window's **Source Tree** area.
4. Right-click on the Database source and select **Show Properties**. Set **CheckCustomer** as the Name.



The following details can be defined:

- **Wizard** – You can create an SQL statement visually with the Database Wizard.
- **SQL Statement** – You can create an SQL statement manually.
- **Dynamic SQL Statement** – You can determine whether you want to write your own INSERT/UPDATE/DELETE statements, which will be executed "as is".
- **Database Definition** – A list of the defined resources for databases. If you have not previously defined a resource or want to add a different one, you can click the **New** option in the dropdown list, which will enable you to add a new resource.
- **DB Disconnect** – Defines whether all open connections to the database will be disconnected at the end of the step.
- **Error Handling Flow** – Enables you to handle exceptions using a specific error flow mechanism. This is only valid for Database operations that are defined as destinations.




When the Database is used as a source, the only DB Operation that can be defined is **Select**.



Magic xpi offers two modes of configuration:

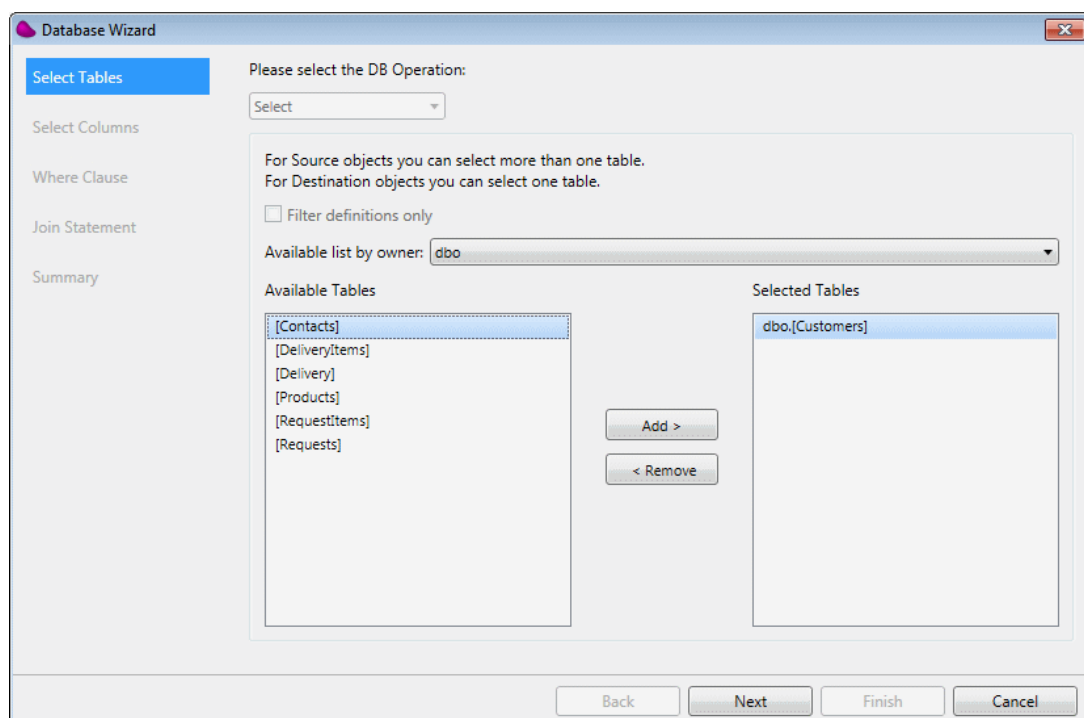
- **Wizard** – This will guide you through the necessary steps to create a valid SQL statement.
- **SQL** – This enables you to enter your own SQL statement.

During this course, you will use the wizard.

5. In the **Properties** pane's **Wizard** field, click the  button. The **Database Wizard** opens.

The wizard shows a list of the tables found in the database. There are two selection panes which enable you to select which tables will be used. You use the buttons to select the tables. In the **Selected Tables** list, there is a list of tables that have been selected. If you want to remove a table from the list, you can park on it and click **Remove**. To select the **Customers** table:

6. Park on the **Customers** entry in the **Available Tables** pane.
7. Click **Add**. The **Customers** entry is removed from the **Available Tables** pane and is added to the **Selected Tables** pane.





There are various modes of filtering and sorting the entries in the **Select Tables** view:

- **Filter Definitions Only** – This check box lets you select the owner and the tables that will be fetched for the Select Table operation. You can select a specific owner and view all of this owner's tables, or you can select only a subset of these tables.
- **Available list by owner** – This enables you to filter the list according to a certain owner or to display all of the tables. Bear in mind that there may also be system tables in the list.

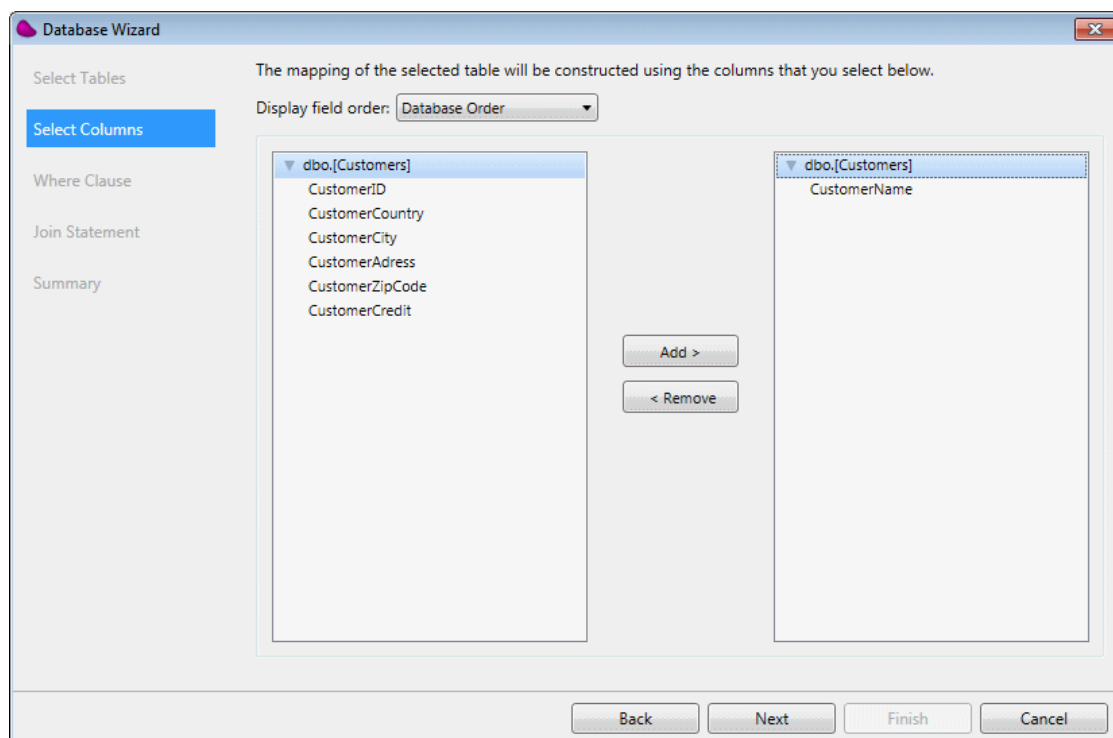
8. Click **Next**. The **Select Columns** dialog box opens.

The **Select Columns** dialog box displays a list of the columns or fields found in the tables that were selected. The **Display Field Order** field lets you define the order in which the database fields will be displayed – either as defined in the table or in alphabetical order.

In the **Select Columns** dialog box, you select which table columns will be fetched from the tables. Remember that the result will be a database SELECT statement. The columns displayed are those that are defined in the tables you selected in the previous step.

9. Park on the **CustomerName** entry in the **Available Columns** pane.

10. Click **Add**. The **CustomerName** entry is removed from the **Available Columns** pane and added to the **Selected Columns** pane.



You have selected the table that you want to use (the **Customers** table) and selected the column you want to display (the **CustomerName**). Now you need to fetch an entry that matches the **CustomerID** from the request. This is performed with the SQL WHERE clause of the SELECT statement. A WHERE clause enables you to filter the number of records that will be fetched.

11. Click **Next**. The **Where Clause** dialog box opens.

You are provided with two lists:

- **Available Columns** – A list of all the columns in the tables you selected.
- **Variables** – The list of variables that you can use.

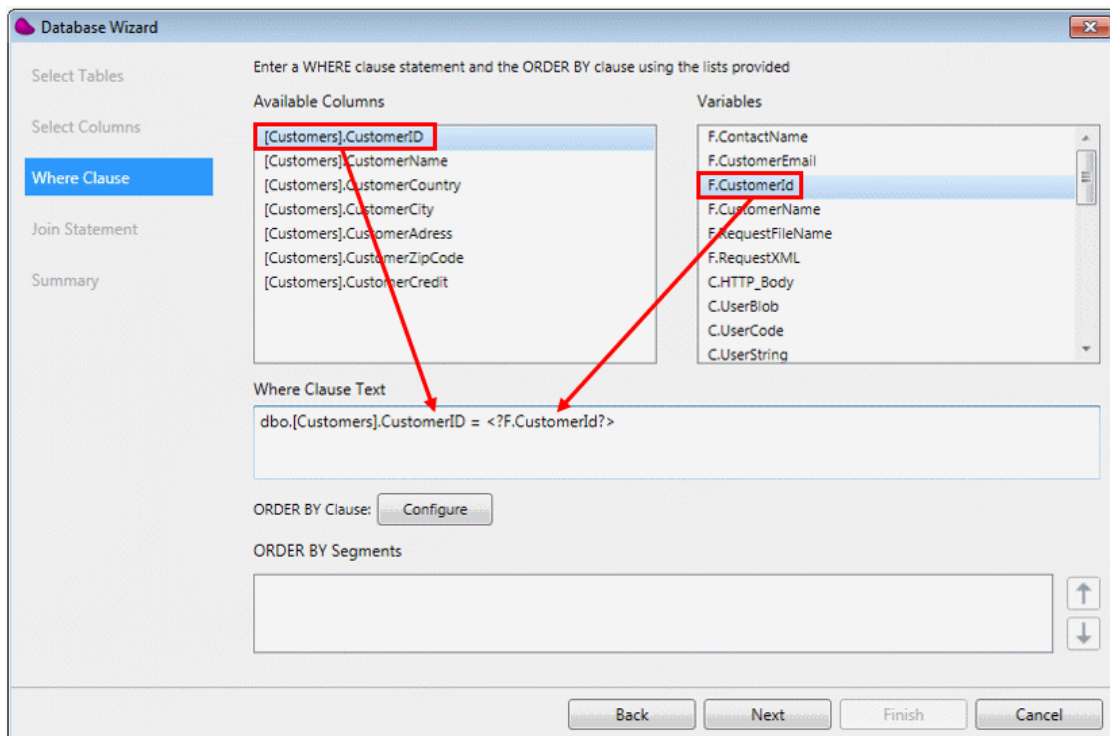
To select an item from the list, you need to park on it and double-click. You may select any entry more than once.

12. Park on **[Customers]CustomerID** in the **Available Columns** pane and double-click. **[Customers]CustomerID** is added to the **Where Clause Text** pane.

13. Park on the **Where Clause Text** pane and type = after **[Customers]CustomerID**. The text should now read: **[Customers]CustomerID =**

14. Park on **F.CustomerId** in the **Variables** pane and double-click. The text should now read: **[Customers]CustomerID = <?F.CustomerId?>**

You may also manually type in **<?F.CustomerId?>**. The name of the variable is enclosed by **<? and ?>**. These are internal Magic xpi symbols that will be replaced in deployment by the value found in **F.CustomerId**.




## Order By

You can click the **ORDER BY Clause** field's **Configuration** button to define the ORDER BY clause that will be sent to the database in the SELECT statement. The ORDER BY clause will then appear in the **ORDER BY Segments** pane. This clause enables you to define how you want the results to be sorted. In the current example you are expecting a single result, if any, and therefore there is no need for an ORDER BY clause.

After clicking **Next**, you are presented with the **Summary** dialog box. Here you see the result of what you previously defined. You can make changes to the SQL statement if necessary.

The next stage is to map the record retrieved by the Data Mapper to a variable. You already learned how to do this earlier in this lesson.

15. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Data Mapper** window's **Destination Tree** area.
16. Right-click on the Variable destination and select **Show Properties**. In the **Properties** pane, enter **CustomerName** in the **Name** field.
17. In the Variable destination's **Properties** pane, click the **Variables** field's  button.
18. In the **Destination Properties** dialog box, select the **F.CustomerName** variable.
19. Expand the source and the destination.
20. Connect the **CustomerName** node on the source to **F.CustomerName** in the destination.



If no record is found, **F.CustomerName** will be blank.

## Exercise

The following steps will only be performed if the customer exists in the database.

1. Pull the Contact ID from the Request XML. You will need to create a flow variable to accomplish this.
2. Now that you have the Contact ID from the Request XML, you need to check that ID against the **Contacts** table to see if the contact exists.
3. If the contact does not exist, add an Email component to send an email back to the customer stating the following '**You are not registered as an official contact for your company. Please approach your representative.** '

## Summary

In this lesson, you learned about the Data Mapper utility. You learned how to:

- Add a Data Mapper to a flow and define the properties.
- Use the Data Mapper to extract information from an XML.
- Use the Data Mapper to fetch information from a Database table.



# Lesson 8

## The Runtime Environment

Up until now, you checked for files and sent emails using the Debugger. Now you will learn more about what goes on behind the scenes. When you eventually deploy your project, you will not be using the Debugger and you will not be running the Magic xpi Studio. The program that actually executes your program is the Magic xpi Server.

As you already know, a Magic xpi project is comprised of business processes containing flows, and within each flow there are several steps. Each step is a representation of a component that performs some actions.

As you develop a project, you provide instruction and configure each component. You also provide flow logic that instructs Magic xpi what component to execute, and in what order.

When you want your project to execute, you start a background process that reads those instructions and executes the components according to the flow logic. This background process is the Magic xpi Server.

## The Executable File

When you installed the Magic xpi Suite, the installation program installed several components. These include the Magic xpi Studio, the Magic xpi Server, the Magic Monitor, and the GigaSpaces middleware.

When you are in development mode and save the project, Magic xpi saves the project definition files in XML format in the `<My Documents>\Magic\projects\<Project name>\current project\Source` directory.

The first step in executing a project with the Magic xpi Server is building an executable file. When you want to deploy your project, you need to save the project as an executable file (**ibp**).

To do this:

1. Open the **Build** menu.
2. Select the **Build Solution** option.

If the file was already built, you can select **Rebuild Solution**, or simply press **CTRL+SHIFT+B**.

- **Build Solution** – Uses a number of optimization rules on the generation process of the modified objects to create the executable file (ibp). Magic xpi saves the project (only the modified objects), loads the project files and runs the Checker on the project level. If the Checker finds any errors, Magic xpi does not build the project. Note that when using the **Build Solution** option, the Magic xpi Checker does not check for variables that are not in use.
- **Rebuild Solution** – Ignores the optimization rules. Magic xpi saves the project, loads the project files and runs the Checker on the project level. If the Checker finds any errors, Magic xpi does not build the project.

When the Magic xpi Server loads, it executes in the background and as such has no user interface. It automatically loads a project and executes it. The project that is executed is determined by two environment variables:

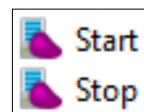
- **currentprojectdir** – This variable holds the path to the folder where all the project files reside, including the project file itself. If you did not change the default setting, this folder contains the name of your projects. It is located in the **projects** subfolder, which is in the Magic xpi installation folder.
- **currentproject** – This variable holds the complete path and file name to the project file itself. This file was created when you built the executable file.

Both of these environment variables are predefined, and are updated automatically by the Magic xpi Studio.

When you build an executable file, the Magic xpi Studio updates the project shortcut on your Windows Start menu. This enables you to control the execution of the project using the Magic xpi Server.

Under the **Projects** folder, you will find an entry with the name of your project, and two shortcuts in a submenu. These are:

- **Start** – Starts the Magic xpi Server, if it is not already running, and runs the project.
- **Stop** – Stops processing the project.



## Executing a Project

When a Magic xpi project starts, the Magic xpi Server loads the project metadata and performs a series of initialization activities. The following list describes the Magic xpi project's initialization phase:

- The Magic xpi Server is assigned a Server ID.
- Magic xpi reads the environment variables and loads the project executable file (**Magic xpi\_course.ibp**).
- If a project is not running yet, the first Server that starts will create the project in the Magic Space.
- The server can now start handling requests.

At this point, the Magic xpi Server executes the project according to the flow logic.

Each activity is logged in the messaging system and can be viewed using the Magic Monitor (see later in this lesson).

The Flow Manager is the part of the Magic xpi Server that is responsible for the actual execution of the flow. After a component is executed, the Flow Manager checks for errors by examining the error return code, and executes the internal error handler or invokes the error flow. If there are no errors, the Flow Manager checks the return status code and executes the internal logic handler or the logic component, if one is defined. Once the internal variables are set, the Flow Manager decides which component will be executed next, based on the flow logic. You will learn more about error handling later in this course

The Flow Manager is also responsible for calling steps in a different Server, using the Magic xpi routing system, if the next step is set to run on a remote Server.

If transactions are defined, the Flow Manager is responsible for managing the transaction by sending **Start Transaction**, **Commit**, and **Rollback** commands as required.

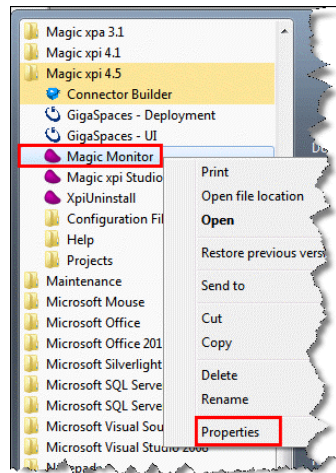


## Magic Monitor

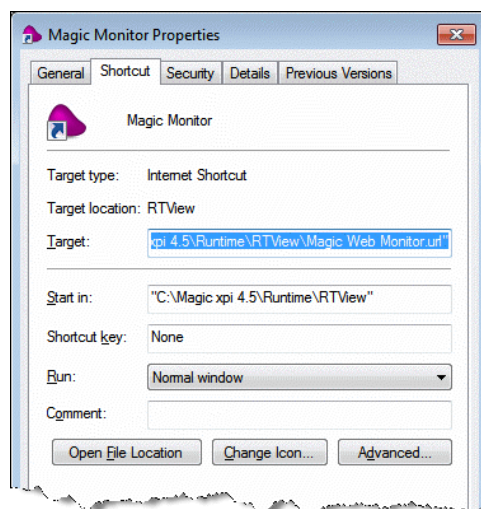
The Magic Monitor is a tool that enables you to track the execution of your project by giving you accurate information about your projects in a single intuitive and easy-to-use dashboard. You can view the information for whole projects or you can select different levels within projects, and you can use filters to display information from specific times. The information displayed is updated regularly. The status of each project is taken from the Space.

You open the Monitor from the Windows **Start** menu's **Magic Monitor** shortcut. By default, this link points to the local host. You should change it to link to the host running the Magic Monitor services:

1. Right-click the **Magic Monitor** shortcut in the **Start** menu.
2. Select **Properties**.



3. The **Magic Monitor Properties** dialog box opens.



4. Change the URL to point to the host running the Monitor services. If, for example, the host that runs the Monitor services is 10.1.3.75, you would change the URL to: <http://10.1.3.75:8068/magicmonitor/panels.jsp>

**Note:** By default, the Magic Monitor Web Server uses port 8068. To change this port, open the <Magic xpi installation>\Runtime\RTView\servers\apache-tomcat-6.0.18-s1\conf\server.xml file, and then change the Connector port parameter to any non-SSL HTTP/1.1 port.

The **Magic Monitor** shortcut will open a link in a browser and the following login screen will appear.



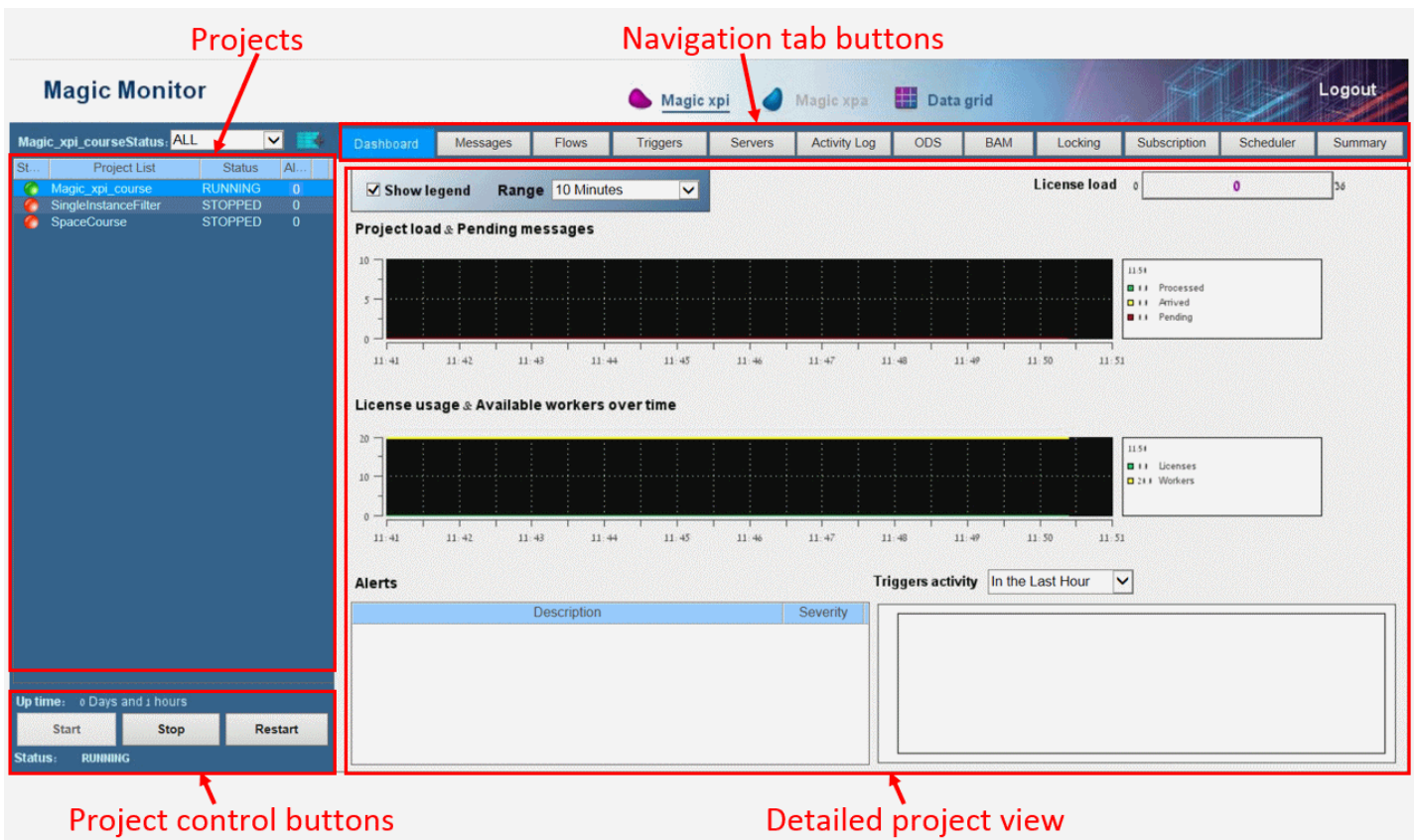
The login details are:

- Username – **admin**
- Initial password – **changeit**

When you open the Magic Monitor, you see the **Magic xpi Dashboard**. This displays top-level information about all the projects that are connected to the server, such as license details, Space status, threads in use, and messages. To get more detailed information about specific projects, you can click one of the links that are grouped together under **Actions**.



This opens the **Magic xpi Projects Dashboard**.



The screenshot shows the Magic Monitor interface. At the top, there are navigation tabs: Dashboard, Messages, Flows, Triggers, Servers, Activity Log, ODS, BAM, Locking, Subscription, Scheduler, and Summary. The 'Servers' tab is selected. On the left, there is a 'Projects' pane with a table of project status. At the bottom left, there are 'Project control buttons' (Start, Stop, Restart) and a 'Status' indicator. The main area shows a 'Detailed project view' with two line graphs: 'Project load & Pending messages' and 'License usage & Available workers over time'. There are also sections for 'Alerts' and 'Triggers activity'.

This consists of two main areas:

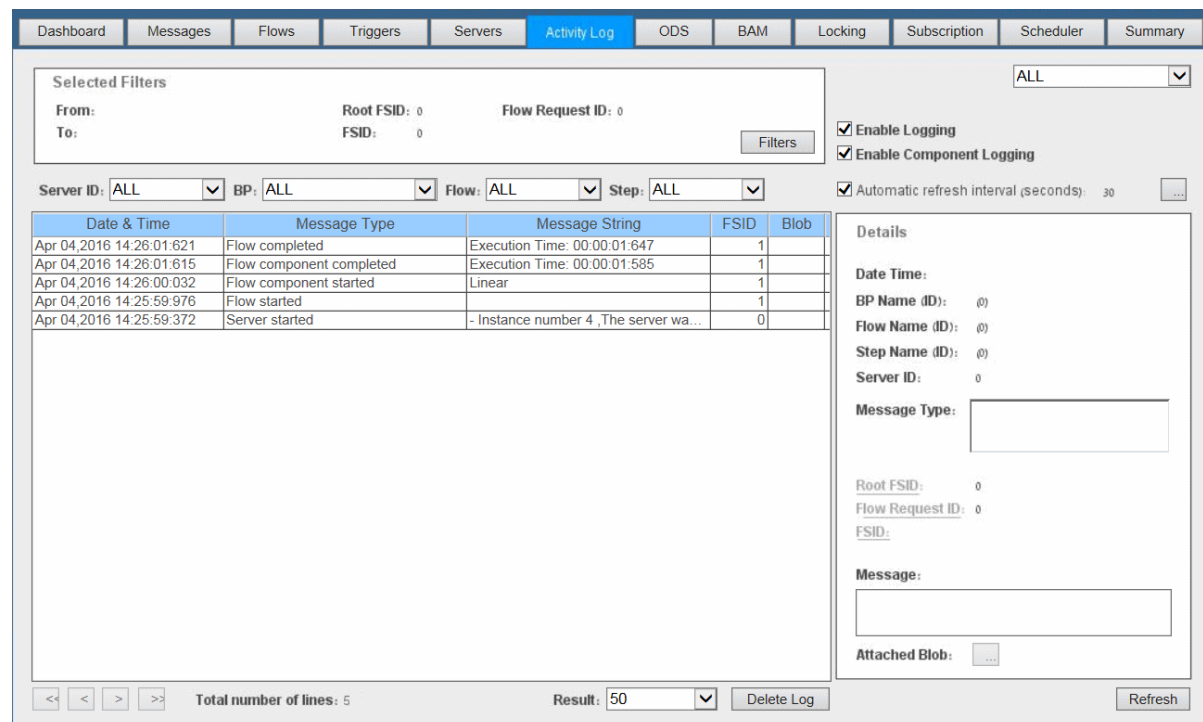
- **Projects** pane – On the left of the screen, this shows a list of available projects. You can control your projects using the **Start**, **Stop**, and **Restart** buttons.
- **Project View** – Covering the rest of the screen, this gives you detailed information about your projects, including messages, license usage, available workers, alerts, and trigger activity. You can adjust the time span of the displayed information.

At the top of the **Magic xpi Projects Dashboard**, you can click the **Messages**, **Flows**, **Triggers**, **Servers**, **Locking**, **Subscription**, **Scheduler**, **Activity Log**, **ODS**, **BAM**, and **Summary** tab buttons to navigate to specific project information.

## Activity Log

The Magic Monitor's **Activity Log** tab displays detailed information about the Magic xpi Server log messages. There are various check boxes and dropdown lists that enable you to filter the information that is displayed. You can add your own messages to this log, as you will learn in the next section.

Name	Description
Date & Time	This column displays the date and time the message was written to the log. The time is displayed in milliseconds.
Message Type	The message that is written to the log. The message describes the action taking place. Messages displayed include: <ul style="list-style-type: none"> <li>When the Server started and when it ended.</li> <li>When the flow started and when it ended.</li> <li>When the component started and when it ended.</li> <li>An error message appears in red.</li> </ul>
Message String	This column displays the text attached to the log process. This text has additional information about the execution process.
FSID	This is the flow sequence ID in the execution process. If the message is general, and not part of a flow, this number is zero.
BLOB	This column holds any BLOB file that is part of the message.



The screenshot shows the Magic Monitor Activity Log interface. At the top, there is a navigation bar with tabs: Dashboard, Messages, Flows, Triggers, Servers, **Activity Log**, ODS, BAM, Locking, Subscription, Scheduler, and Summary. Below the navigation bar, there are filter controls for Selected Filters (From, To, Root FSID, Flow Request ID), a dropdown menu set to 'ALL', and checkboxes for 'Enable Logging', 'Enable Component Logging', and 'Automatic refresh interval (seconds): 30'. Below these are dropdown menus for Server ID, BP, Flow, and Step, all set to 'ALL'. The main area contains a table with columns: Date & Time, Message Type, Message String, FSID, and Blob. The table displays five rows of log entries. To the right of the table is a 'Details' panel with fields for Date Time, BP Name (ID), Flow Name (ID), Step Name (ID), Server ID, Message Type, Root FSID, Flow Request ID, FSID, Message, and Attached Blob. At the bottom, there are navigation buttons (back, forward), 'Total number of lines: 5', 'Result: 50', 'Delete Log', and 'Refresh' buttons.

Date & Time	Message Type	Message String	FSID	Blob
Apr 04, 2016 14:26:01:621	Flow completed	Execution Time: 00:00:01:647	1	
Apr 04, 2016 14:26:01:615	Flow component completed	Execution Time: 00:00:01:585	1	
Apr 04, 2016 14:26:00:032	Flow component started	Linear	1	
Apr 04, 2016 14:25:59:976	Flow started		1	
Apr 04, 2016 14:25:59:372	Server started	- Instance number 4 ,The server wa...	0	

## Exercise

In previous lessons you used the Magic xpi Debugger to see Magic xpi in action. You will now run the project using the Magic xpi Server. You can use the following checklist:

- Save your project.
- Build an executable file.
- Open the Magic Monitor.
- Execute the Magic xpi Server for your project.
- View the project's details in the Magic Monitor.

## Summary

In this lesson, you learned about the Magic xpi Server and the Magic Monitor.

You will now be able to:

- Build execution files from the Magic xpi Studio.
- Start and stop the Magic xpi Server for a specific project.
- Start the Magic Monitor.
- Navigate between Monitor views.
- View specific project information.



Magic®  
University

# Lesson 9

## Testing Your Project

You learned how to develop an integration project, and you learned how to deploy it. Before moving your project to production, you need to test your code and make sure it works correctly.

Magic xpi provides several tools to help you test and debug your project, so that you can deliver it error-free.

In a previous lesson, you were introduced to the Magic xpi Debugger. This enables you to test your flow. You will learn more about the Magic xpi Debugger in this lesson.

This lesson covers:

- The Magic xpi Checker tool
- Additional Magic xpi Debugger functionality

## Magic xpi Checker

The Magic xpi Checker is a utility that resides in the Magic xpi Studio itself. It examines the way that the components and flows are defined, and checks for errors in the configuration and definitions.

The Checker starts automatically when you start the Debugger, or when you save an executable file for deployment.

The Checker will disable part of the project or the whole project, if errors are found, as follows:

- When activated with the Debugger, an executable file will not be created at all.
- When activated as a part of the **Build Executable File** process, flows with errors will be changed to an inactive state. It is therefore possible that the entire project will be inactive.

The Checker can also be started manually:

- From the **Project** menu, select **Checker...** and then **Run** to run the Checker on the whole project (this option can be activated by typing **CTRL+R**).
- From the flow's context menu, select **Checker...** to check the selected flow only.



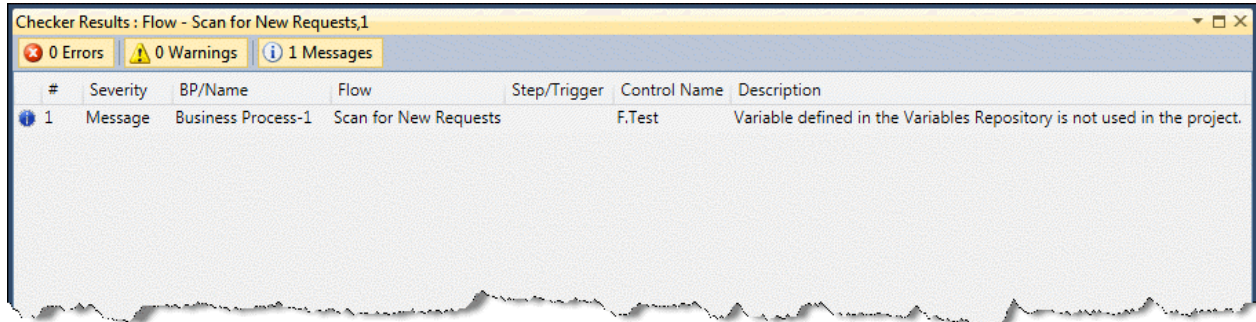
The Checker does not check any logic within the step.



## Checker Results

If errors were found, the Checker will open a dialog box listing all the errors.


The information displayed is as follows:



The **Checker Results** screen lists every item checked and indicates areas where there is an error. The errors are displayed with an indication of their severity on the following levels:

Type	Description
Error	This is the most severe level. It indicates a fatal error, which means that the step will not work. An example of a severe error is: Missing component configuration. This indicates that mandatory configuration information is missing from the component.
Warning	This indicates an error that does not prevent the step from executing. Results at deployment might not be what you expected based on your project definitions.
Information Message	This indicates efficiency-type problems, such as a timeout setting that is not defined, or insufficient Server capabilities for the relevant flows.

The filter buttons at the top of the **Checker Results** screen let you define which types of Checker messages to display.



You can double-click on a line in the **Checker Results** dialog box to go straight to the location that generated the error, warning, or information message.

## Magic xpi Debugger


The Magic xpi Debugger tool is a part of the Magic xpi Studio that enables you to test your integration project during the development stage.

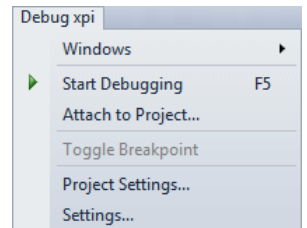
The Debugger runs your project or attaches to a running project. It gives you the ability to control and view execution sequences, flow variables, the calling of Magic xpi services, and breakpoints. You can execute steps one at a time, or you can run the execution automatically until you reach a breakpoint.

Magic xpi executes the project in Debug mode using the running Server engine.

You can view all property-type settings for steps, but only in read-only mode. To make any necessary changes, you will need to stop the Debugger process, go back to the Flow Editor and make the changes, and then return to the Debugger to restart the process.

Starting a debug session can be done by one of the following methods:

- **Open mode** – Opens a project on a local machine. You can do this from the **Debug xpi** menu by selecting **Start Debugging**, by pressing **F5**, or by clicking the Debugger  icon in the toolbar.
- **Attach to Project mode** – Attaches to a running project on a local or remote machine. A project can consist of many Servers. This option lets the Magic xpi Studio debug running projects. You do this by selecting **Attach to Project** from the **Debug xpi** menu.

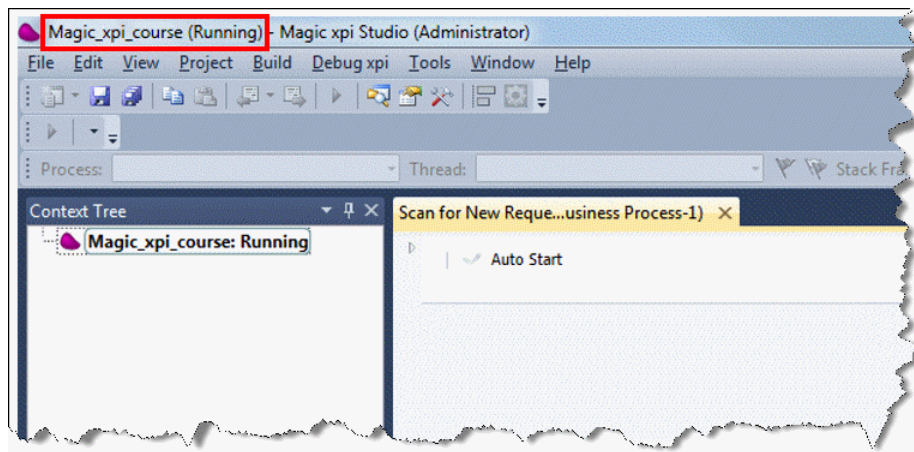



In a previous lesson, you selected **Debug** from the flow's context menu to debug a specific flow. However if you select to debug a specific flow, any other flows in the project will be disabled.

When opened, the Debugger has two different modes:

Option	Description
Stopped	The Server is waiting for a command and will not execute until it receives one.
Running	The Debugger is currently running one flow or more.

The current mode is displayed in the form header, as shown in the image below.



To execute the Debugger, you need to select **Start Debugging** from the **Debug xpi** menu or the  icon from the toolbar.

## Controlling the Debugger

Various additional options are available from the **Debug xpi** menu, which enable you to control the Debugger.

The following options are available when the Debugger's mode is **Stopped**.

Menu Option	Description
Windows	Opens the <b>Breakpoints and Suspend</b> pane and the <b>Output</b> pane.
Start Debugging	Runs the project until a breakpoint is reached, or there are no more steps to execute. ▶
Attach to Project	Lets you start the debug session in Attach to Project mode.
Toggle Breakpoint	Adds or removes a breakpoint from the current step.
Project Settings	Opens the <b>Debugger Settings</b> dialog box. Here, you can define the refresh rate timeout and other debugging options.
Settings	Opens the <b>Options</b> dialog box and takes you directly to the <b>Magic xpi Debugger</b> section's <b>General</b> settings. Here, you can configure the Debugger's timeout settings for your project.

The following options are available when the Debugger's mode is **Running**.

Menu Option	Description
Windows	Opens the <b>Breakpoints and Suspend</b> pane and the <b>Output</b> pane, as well as other various panes.
Continue	Continues the execution of all threads after a break. ▶
Restart	The Debugger is stopped and reset, and project execution is restarted.
Break All	Stops the execution of the project as soon as the execution of all currently running steps is completed.
Step	Execute single step. Active only when the Debugger is in Stopped mode.
Stop Debugging	Stops the debugging process.
Scheduler	Lets you see the upcoming events schedule when you reach a breakpoint in the flow, based on your Scheduler table.
Context View	Opens the Context View for the currently selected step.
Settings	Opens the <b>Debugger Settings</b> dialog box.

## Setting a Breakpoint

Breakpoints are a helpful debugging tool. They enable you to halt execution of the flow on a specific step.

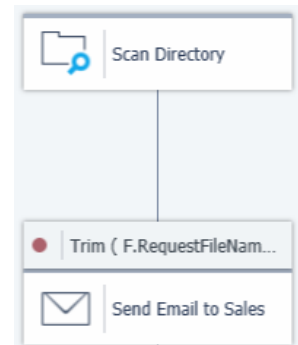
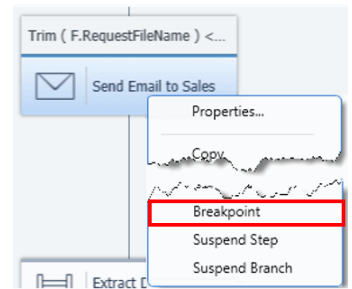
Once a breakpoint is reached, all of the executed contexts, such as threads and branches, are stopped before they execute the next step.

The Debugger then waits until the user selects **Step** or **Continue**.

To activate the breakpoint on a step, select the step and then select **Breakpoint** from the context menu.

When a step has a breakpoint defined, the breakpoint image (●) will appear in its top left corner.

In the image on the right, you can see the step that is marked as a breakpoint.



## Suspending a Step, Flow, or Branch

When debugging, you may want to suspend certain elements to better pinpoint a problem.

A suspended step will be ignored, and the Debugger will continue to the next step as if the step was executed.

A suspended flow will not be executed, and will be considered "stopped" until released from that mode.

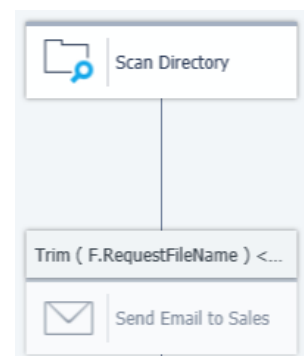
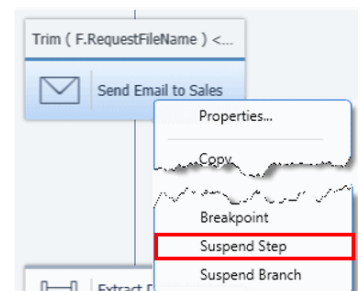
The same method described on this page can be used to suspend a flow or branch.

To suspend a step, select the step and then select **Suspend Step** from the context menu.

When a step is defined as suspended, its icon and text will appear dimmed.

In the image on the right, you can see the step that is suspended since it is dimmed.

The top-most (root) step and the bottom-most steps do not have the **Suspend Branch** option.



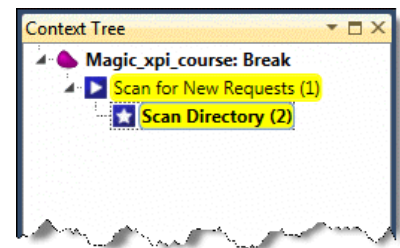
## Context Tree

The Context Tree displays the status of the loaded contexts of a project in the Server. The tree is only refreshed when you select the **Break** option, or when the project reaches a breakpoint. When a breakpoint is reached, the refreshing process is terminated. You can also click **Refresh** to refresh the displayed thread statuses.



There is only one active debug context available at a specific time. The first context that reaches a breakpoint is set as the active debug context. You can set any context as the active debug context by right-clicking and selecting **Set Active Context**. When in Step mode, the focus of the debugging process is always on the active context. You can only step through a context if it is set as the active debug context. You will learn about running other contexts later in this course.

The Context Tree uses various colors and indications to provide you with additional information, as follows:

- **Yellow** – The active debug context. If the active debug context is on the linear path, it will be in bold and highlighted in yellow from top to bottom (including the highest parent). If the active debug context is not on the linear path, only the parallel context will be highlighted in yellow.
- **Gray** – The context path.
- **Green** – Contexts that are still running.
- **Black** – All contexts that have stopped, but not at a breakpoint.
- **Bold** – An active debug context on a linear path, and an actual active step in the active debug context.
- **Regular** – All other contexts, except the one that you are parked on.



The Context Tree uses various icons to enable you to quickly identify each context type. The icons used in the image displayed above are:

	Auto Start
	Linear Step

There are other icons available. You can view them in the *Magic xpi Help*.



## Context View

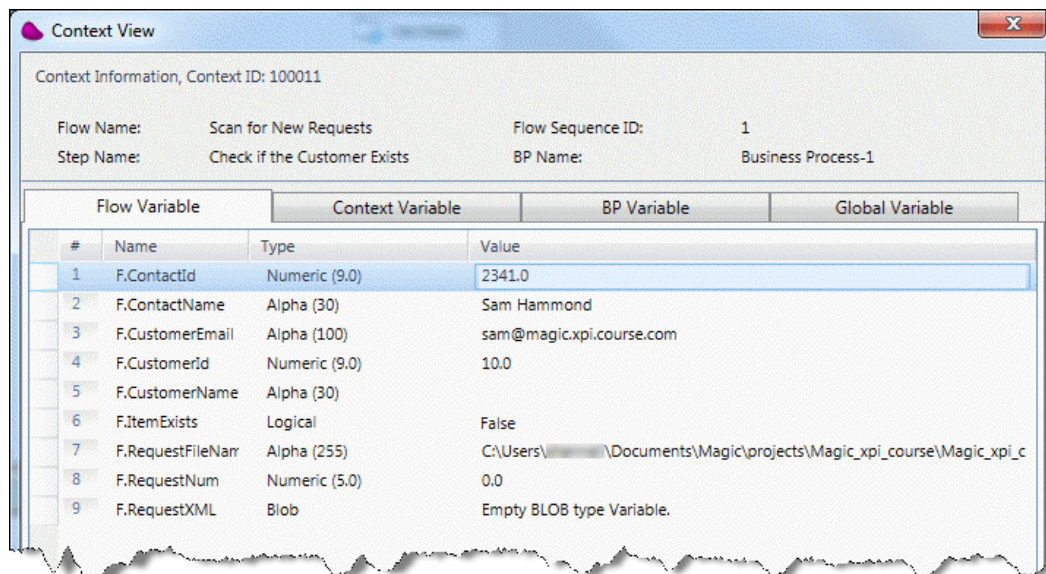
The **Context View** form provides you with all the internal information stored for the currently running view.

This option is available only when the Debugger has stopped; for example, at a breakpoint. It shows the values of all variables in the scope of the current context of the current step, the highlighted step in the Context Tree.

To activate the Context View, make sure you are parked on the correct step and then select **Context View** from the **Debug xpi** menu. Alternatively, you can right-click on the highlighted step in the Context Tree, and then select **Context View**.

The **Context View** form has four tabs, one for each type of Magic xpi variable.

You can change the variables' values from this dialog box. This is very useful when debugging a project.



## Adding User Messages

When you run the project using the Debugger, you can set a breakpoint and the process stops at that point. However, when debugging, you may find that you need to add your own messages to the log for testing purposes, often at a point before the current breakpoint. You can even decide to run the entire flow and view your test messages at the end of the flow.

By adding debug information, you are better able to debug a component.

You can add your own messages by:

- Using the **Logging** properties of the component that you want to debug.
- Adding a **Save Message** utility.

### Logging Section

The **Logging** section is one of the sections in a component's **Properties** pane. You are able to specify any string and/or BLOB information to be sent to the Magic Monitor's **Activity Log** tab.

You have the option to send only a string message, a BLOB variable, or both.

### Logging Scope

Using the **Logging Scope** property, you can specify **when** the information is to be sent to the Magic xpi Activity Log view.

The possible options are:

- **No** – Nothing will be written to the **Activity Log** tab. This is the default.
- **Step** – Magic xpi will write information to the log for the entire step.
- **Method** – Magic xpi will write information to the log for each method in the step. This will be carried out after the method was executed.
- **Full** – Magic xpi will write information for both the **Step** and the **Method** options. The XML interface does not allow **Method** level logging.



## Step Logging Options

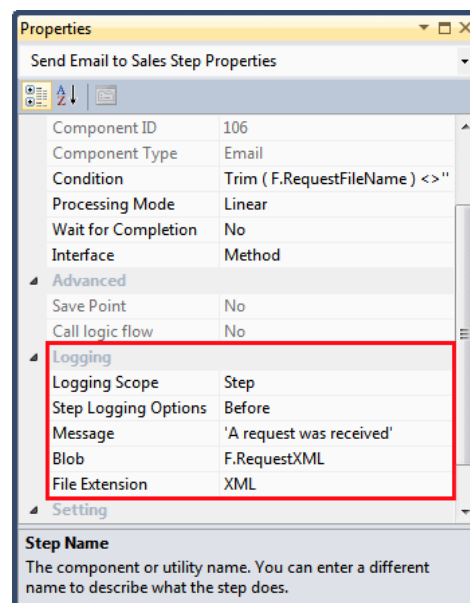
This property is only relevant if either **Step** or **Full** is selected in the **Logging Scope** property. The property defines when the information will be logged to the Monitor.

The available options are:

- **Before** – Information will be logged before starting the step.
- **After** – Information will be logged after the step has finished its execution.
- **Full** – Information will be logged before and after execution of the step.

As an example, add logging details to the **Logging** tab of the **Send email to Sales** component. In this example:

1. Park on the **Send email to Sales** step.
2. Go to the **Logging** section of the **Properties** pane.
3. Define the **Logging Scope** as **Step**.
4. Define the **Step Logging Options** as **Before**, meaning that the information will be written to the log before this component begins executing.
5. Define the **Message**. A string message, **A request was received**, will be written to the log enabling you to easily identify the message.
6. Define the **BLOB**. You want to view the content of the user request that was fetched by the Directory Scanner component and saved in the **F.RequestXML** variable.
7. Define the **File Extension**. Since the request file is in XML format, when you define the extension as **XML**, the file will be opened in the XML viewer defined on your system.



## Exercise

Now that you have learned about the Magic xpi Debugger and Checker, it is time to try them on your project:

- Run the Checker on your project.
- Add a breakpoint on the **Extract details from request** step in your project, and then run the Debugger in Step mode.
- Run the Debugger again, but this time let the Debugger run without breakpoints.

## Summary

In this lesson, you learned how to:

- Check your project for errors using the Checker.
- Debug your project using the Debugger.
- Create breakpoints, and suspend part of the project.
- View and change the values of variables.

# Lesson 10

## Item Validity Check

Mapping data is the most common operation in integration projects. The Magic xpi Data Mapper utility provides an easy-to-use interface to transform data from various source formats to various destination formats.

According to the business scenario, once the system receives a new request and the check for the customer is successful, the system will then need to check that the requested items exist in the database.

In this lesson, you will:

- Use the Flow Data utility to update variables and to store the request data in an ODS.
- Call a flow that checks the stock availability for each of the request's items.

## Flow Data Utility

The Flow Data utility is used to manipulate variables, ODS, and User Parameter data. You can insert, update, clear, reset, and delete the selected data item.

The following table describes the different parameters of the Flow Data utility:

Parameter	Description
Action	<p>The type of action to be performed:</p> <ul style="list-style-type: none"> <li>◦ Update – Adds or inserts data in the target flow data.</li> <li>◦ Clear – Clears all of the data in the target data flow.</li> <li>◦ Reset – Resets a variable to its original value (does not work with ODS or Environment variables).</li> <li>◦ Delete – (ODS only) Deletes an ODS entry from the database.</li> <li>◦ Insert – (ODS only) Inserts data to an ODS array.</li> </ul>
Type	<p>The type of the target variable that the action will be performed on:</p> <ul style="list-style-type: none"> <li>◦ Flow</li> <li>◦ Context</li> <li>◦ Global</li> <li>◦ Business Process</li> <li>◦ ODS Global</li> <li>◦ ODS Local</li> <li>◦ Environment</li> </ul>
Dyn	Dynamic variable name. When checked, the variable name is determined by an expression.
Name	The name of the target variable. When the action is Delete, and the type is ODS, you can leave this field blank in order to delete all ODS variables.
Data Type	The data type of the variable. This type is filled in by the system based on the target variable that you selected. You can select the type only for ODS when the action is not Delete.
Encoding	The encoding type for updated BLOB and Alpha variables.
Index	The index of the ODS record in the ODS data array.
Update Expression	The value of target variable in an Update action will be updated with the return value of this expression.
Condition	The return value of this expression (must be Boolean expression) will determine if the selected action will be performed.

## Using the Flow Data Utility

First, you will need a variable that will contain the request number. This will be a flow variable. Later in the lesson you will be checking to see whether the items requested exist in the database. Therefore, you will need a variable that will indicate whether all items are available in stock. This variable will be used in more than one flow and therefore will need to be a context variable.

1. In the Solution Explorer, under the **Scan for New Requests** flow, double-click **Flow Variables** and add the following variable to the Flow Variables repository:
  - **F.RequestNum** with a type of **Numeric** and size **5**.
2. In the Solution Explorer, under the **Repositories** folder, double-click **Context Variables** and add the following variable to the Context Variables repository:
  - **C.All\_Items\_Exist** with a type of **Logical**.

In the company scenario, the request will be added to the system even if the company is unknown. This will be in two separate threads.

3. Park on the **Check the Contact** step.
4. From the context menu, select **Properties**.
5. Set the **Processing Mode** to **Parallel**.

You need to initialize the **C.All\_Items\_Exist** context variable and save the request number in the **F.RequestNum** flow variable.

6. Add a Flow Data utility as a child step of the **Check if the Customer Exists** step.
7. In the **Properties** pane's **Step Name** field, type: **Get Req number from FileName**
8. In the **Description** field, type: **This service updates the RequestNum variable and initializes the All\_Items\_Exist variable that will be used later on.**
9. Double-click the Flow Data utility, or right-click on it and select **Configuration** from the context menu to open the **Flow Data Configuration** dialog box.
10. Click **Add** to add commands.

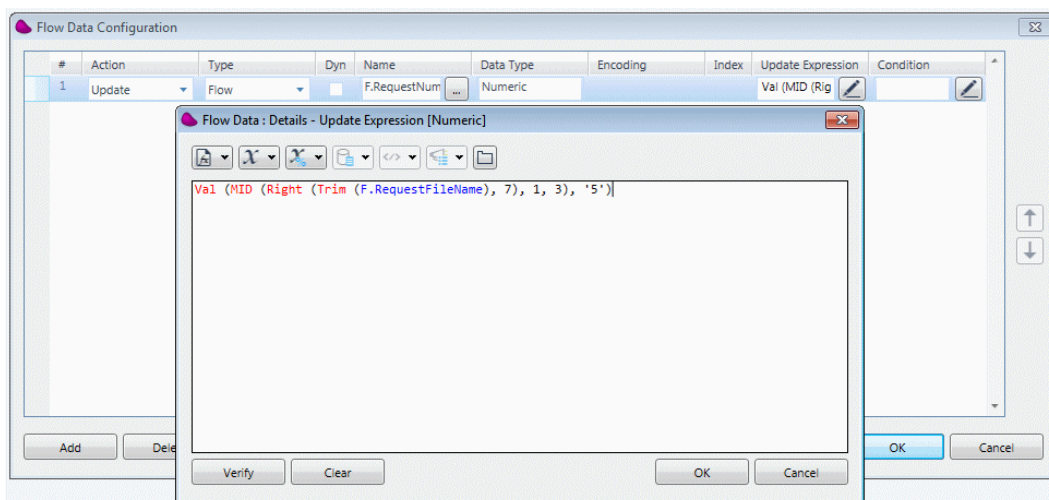
The request number is fetched as part of the physical file name. When the Directory Scanner scanned for requests, it scanned according to the following mask: **request???.xml**. As an example, if the Directory Scanner finds a file named **request003.xml**, then the request number is **003**. This three digit number is the number you must retrieve using Magic xpi functions. The actual file name returned by the Directory Scanner includes the full path, which complicates the final expression.

To retrieve this number you will be using the following functions in the next step:

- **Trim** – This removes trailing blanks from both the left and right sides of the expression, so you will use: **Trim(F.RequestFileName)**
- **Right** – This function fetches a number of characters from the right side of the expression. The idea is to retrieve the value 003.xml, which is 7 characters from the right, so you will use:  
**Right (Trim (F.RequestFileName), 7)**
- **Mid** – This function retrieves a string from within another string. You set the start point and how many characters you want to retrieve. The previous expression left you with 003.xml. All you need is the first three characters, 003, so you will use:  
**Mid (Right ( Trim (F.RequestFileName), 7), 1, 3)**
- **Val** – This function takes a string with digits and converts it to a numeric value. The previous expression returned the string '003'. This needs to be converted to a number, so you will use: **Val (Mid (Right ( Trim (F.RequestFileName), 7), 1, 3), '5')**  
Note that you will see 3 and not 003, this is the expected behavior.

11. Add the following commands to the Flow Data step:

Action	Type	Name	Data Type	Update Expression
Update	Flow	F.RequestNum	Numeric	Val (Mid (Right ( Trim (F.RequestFileName), 7), 1, 3), '5')
Update	Context	C.All_Items_Exist	Logical	'TRUE'log



## Operational Data Storage

Operational data storage (ODS) is a type of database that is often used as an interim area for a data warehouse.

Unlike a data warehouse, which contains static data, the contents of the ODS are updated through the course of business operations.

ODS is designed to quickly perform relatively simple queries on small amounts of data (such as finding the status of a customer order), rather than complex queries on large amounts of data, typical of the data warehouse.

ODS is similar to a person's short term memory in that it stores only very recent information. In comparison, the data warehouse is more like long term memory in that it stores relatively permanent information.

### Magic xpi ODS System

The Magic xpi ODS system provides a way to save data that can be shared by flow components or integration flows.

The ODS system manages a data table that maintains an entry for each item saved in the ODS.

Each item saved in the ODS is identified by its flow sequence ID and UserKey. An ODS data item can be Alpha, Numeric, Date, Time, Logical, or BLOB. The data saved in the ODS system can be retrieved by other Magic xpi Servers.

The Magic xpi ODS system supports the store, retrieve, and save data-retrieval modes.

The ODS stores and retrieves data in the integration flow:

- If, during project development, you use the local flow storage (ODS Local), the data is saved in the ODS database for a particular thread. The saved data can be retrieved by steps in the same thread only. The data is cleared from the ODS database when the particular thread is completed.
- If, during project development, you use the global storage (ODS Global), the data is saved in the ODS database and can be retrieved from within any flow.

## Creating a Dynamic ODS

You will use the **Flow Data** utility to create a dynamic ODS that will hold the Request XML file. The request will be stored in the ODS system until it is retrieved. You will retrieve data from the ODS in a later lesson. This can be added as a separate step or as part of the previous step. Here you will add it as part of the **Get Req number from FileName** step.

1. Double-click the **Get Req number from FileName** step, or right-click on it and select **Configuration** from the context menu.
2. In the **Flow Data Configuration** dialog box, click **Add**.
3. Enter the following information for the command:

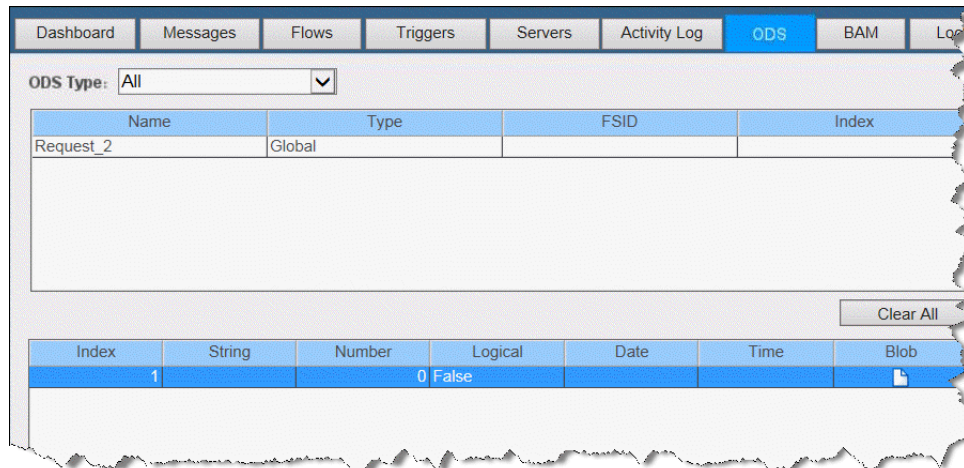
Action	Type	Dyn	Name	Data Type	Update Expression
Insert	ODS Global	<input checked="" type="checkbox"/>	'Request_' & Trim(Str(F.RequestNum,'5'))	BLOB	F.RequestXML



Selecting the **Dynamic** check box indicates that the name will be a dynamic name, meaning that it will be evaluated as an expression. In the example above, the name will be evaluated to **Request\_1**.



When you run the flow and view the project in the Magic Monitor, you are able to see the entries added to the ODS, as shown in the image below:



Name	Type	FSID	Index
Request_2	Global		

Clear All

Index	String	Number	Logical	Date	Time	Blob
1		0	False			

In the case above, the flow was executed twice with the same request number. Each time the flow was run, an entry was added to the ODS. Magic xpi automatically provided an incrementing index in order to differentiate between the instances. At a later stage, the challenge will be how to retrieve the index you are looking for.

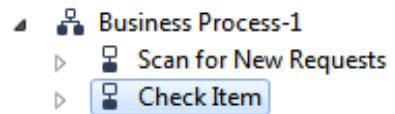
If a BLOB has been added, you can view its contents by clicking the BLOB icon.

## Check Item Flow

At this stage, you have stored the request data in a dynamic ODS. You now need to check the availability of each item in the request in the local database. If there are three items in the request, you need to perform this action three times.

To solve this problem, you will create a flow that checks each item's availability in the database. The flow will check the stock availability of one item and later, you will call this flow for each of the request's items.

1. In the Solution Explorer, right-click on **Business Process-1** and select **Add Flow**.
2. Right-click on the new flow and select **Rename** from the context menu.
3. Change the flow name to **Check Item**.



First, you need to add flow variables that will be used in the flow:

4. Under the **Check Item** flow, double-click on **Flow Variables**.
5. In the Flow Variables repository, add the following variables to your project:

Name	Description	Type	Length	Default Value
F.ItemCode		Alpha	30	
F.RequestNum		Numeric	5	
F.Quantity		Numeric	9.2	



The flow variables defined in this flow will become parameters for the flow when it is called by other flows.

Checking whether the item exists in the database involves using functionality that you already learned about.

6. In the **Check Item** flow, add a **Data Mapper** utility.
7. In the **Step Name** field, type **Report Items Existence**.
8. In the **Description** field, type: **This step checks if the quantity in stock is more than the quantity requested.**
9. Double-click the **Report Items Existence** step, or right-click on it and select **Configuration** from the context menu.

In the **Data Mapper** window:

10. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
11. Right-click on the Database source and select **Show Properties**.
12. Enter **CheckItem** in the **Name** property.
13. Use the Database wizard or the SQL option to:
  - a. Select the **Products** table.
  - b. Select the **StockQuantity** column.
  - c. In the WHERE clause, define: **[Products].ProductCode='<?F.ItemCode?>'**
14. Click **Finish**.
15. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the Data Mapper window's **Destination Tree** area.
16. Right-click on the Variable destination and select **Show Properties**.
17. Enter **Update\_General\_Check\_Var** in the **Name** property.
18. In the **Variables** field, select the **C.All\_Items\_Exist** context variable.




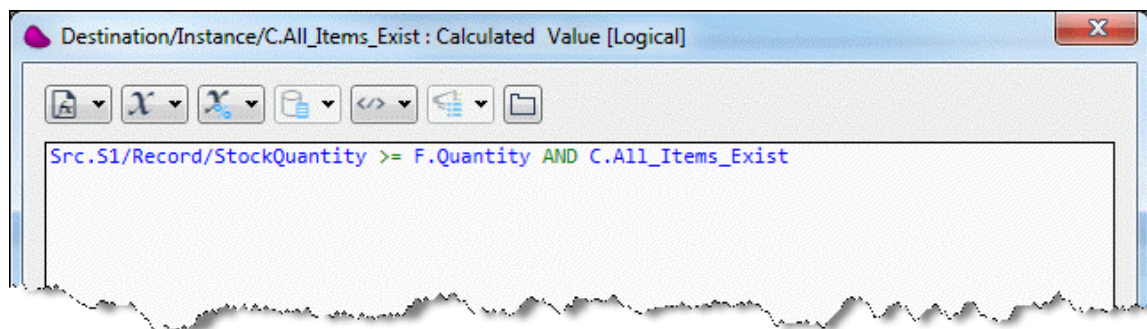
19. Expand the source and the destination.
20. Connect **StockQuantity** to **C.All\_Items\_Exist**.

The **C.All\_Items\_Exist** variable will updated if there is enough of the item in stock. However, if a previous iteration found that there were not enough items in stock, this variable need not be updated. To do this:

21. In the Data Mapper, park on the **C.All\_Items\_Exist** node and select **Show Properties** from the context menu.
22. Park on the **Calculated value** property and open the Expression Editor.

You need to use the value returned from the database SELECT statement to update the check by the **C.All\_Items\_Exist** value.

23. In the Expression Editor, click . This will insert the **Src.S1/Record/StockQuantity** source node into the Expression Editor.
24. Enter the expression:  
**Src.S1/Record/StockQuantity >= F.Quantity AND C.All\_Items\_Exist**



Here is something for you to think about. What will happen if the user requests an item that does not exist?  
This will be part of your exercise at the end of this lesson.

## Calling the Check Item Flow for Each Item

The request may contain one or more items. To check each item's availability, you need to call the **Check Item** flow for each item. The Magic xpi Data Mapper enables you to call a flow for each occurrence of a multi-occurrence element. Of course, if there is only one item, the flow will only be called once.

You will now add a Data Mapper step with a Call Flow destination to the **Check Item** flow. By using the Call Flow, the Data Mapper will invoke the **Check Item** flow for each item in the order.

1. Park on the **Scan for New Requests** flow.
2. Add a Data Mapper utility as a child step of the **Get Req number from FileName** step.
3. In the **Step Name** field, type **Check Items**.
4. In the **Description** field, type: **For each item from the request, call the "Check Item" flow.**
5. Double-click the **Check Items** step, or right-click on it and select **Configuration** from the context menu.

In the **Data Mapper** window:

6. From the Toolbox's **Mapper Schemas** section, drag an XML source into the **Source Tree** area.
7. Right-click on the XML source and select **Show Properties**.
8. Enter **Items\_from\_Request** in the **Name** property.
9. In the **XSD File** field, type: **course\_data\schemas\request.xsd**
10. Set the **Source Type** field to **Variable**.
11. Open the Variables List and select the **F.RequestXML** variable.

To add a Call Flow as a new destination:

12. From the Toolbox's **Mapper Schemas** section, drag a Call Flow destination into the **Data Mapper** window's **Destination Tree** area.
13. Right-click on the Call Flow destination and select **Show Properties**.
14. Enter **CheckEachItem** in the **Name** property.
15. In the **Flow Name** property, open the **Flow List** and select the **Check Item** flow.

Calling a flow for each occurrence of a multi-occurrence element is done by connecting the compounds of the source and destination schemas.

16. Expand the source and the destination.

In the **Destination Tree** area, you will see that all the flow variables defined in the destination flow are used here as parameters of that flow.

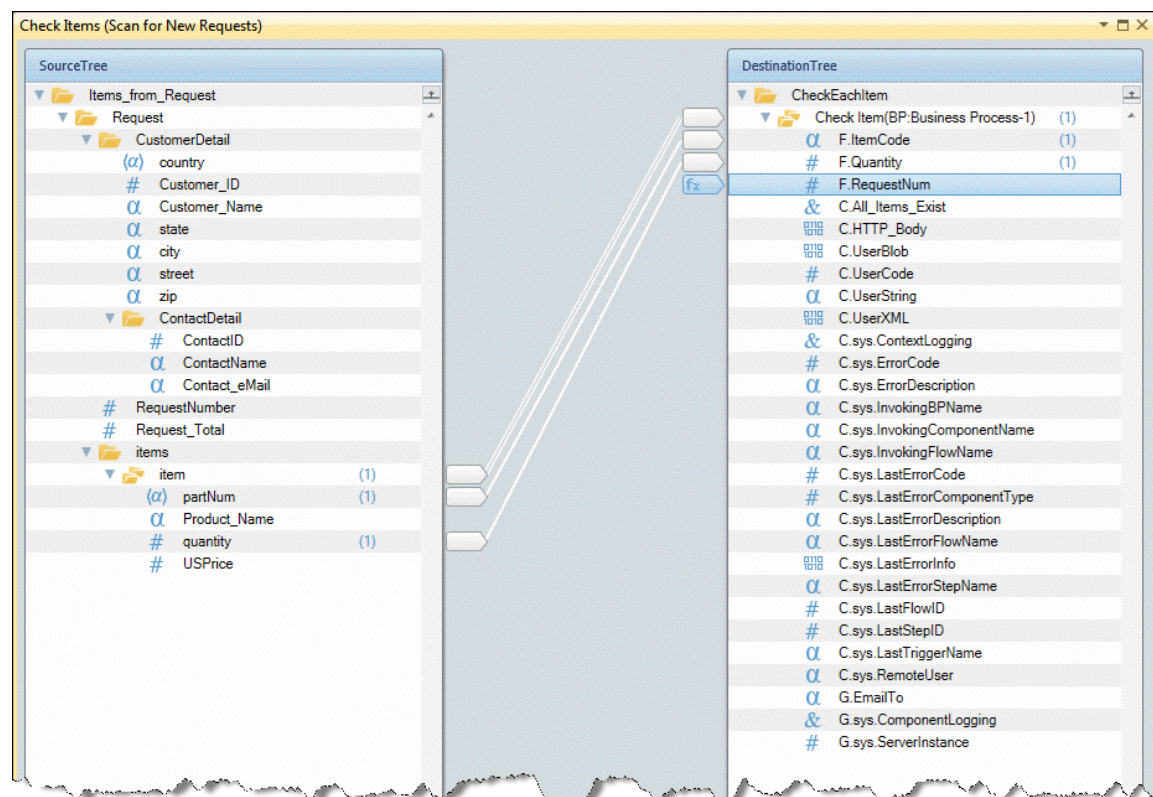
17. Connect the **item** compound node in the **Source Tree** area to the **Check Item** node in the **Destination Tree** area. By doing this you are ensuring that for each item found, you will be invoking the **Call Flow** operation.

18. Open the **Item** node and connect the following nodes:

- **partNum** to **F.ItemCode**
- **quantity** to **F.Quantity**

19. Park on the **F.RequestNum** node in the **Destination Tree** area.

20. In the **Properties** pane, park on the **Calculated Value** property and type **F.RequestNum**. Although it seems that you are updating the **F.RequestNum** variable with itself, this is not the case. The **F.RequestNum** displayed in the **Destination Tree** area is the name of the flow variable in the called flow, **Check Item**.





## Exercise

You will now practice what you learned.

1. The called flow **Check Item** checks whether there is enough stock to supply the required amount. However, two problems arise that you need to handle:
  - The part number in the request may be non-existent. In other words, it does not exist in the database.
  - The catalog price is more than the requested price. In other words, the client wants to purchase the item at a price less than the list price.
2. The request and the items need to be inserted into the local MSSQL databases. This must be executed for each request regardless of whether or not there is a problem with the request.
  - The request header details must be inserted into the **Requests** table. Remember to update the **Requests.CustomerExists** field.  
**Hint:** The customer does not exist if the **F.CustomerName** variable is blank.
  - The request lines must be inserted into the **RequestItems** table. Remember to update the **RequestItems.Status** field. If the current request line is valid, update this with 'TRUE'Log, but if there is a problem update this with 'FALSE'Log.  
**Hint:** Do this in the **Check Item** flow.
3. Create an initialization flow (Auto Start) that deletes all the records from the **Requests** database and **Items** table.

## Summary

In this lesson, you learned how to:

- Use the Flow Data utility.
- Add an entry to the ODS.
- Call a flow from within the Data Mapper.

In the exercise, you used the Data Mapper to insert and delete entries from the database.



Magic®  
University



# Lesson 1 1

## Services

Magic xpi services are a way that Magic xpi provides to enable other applications to invoke Magic xpi externally. These external applications invoke Magic xpi via the Magic xpi service, which then triggers the invocation of a flow.

You learned about resources and the **Resources** section of the **Settings** dialog box in Magic xpi. In a similar way, Magic xpi maintains a central repository defining the external methods that other applications can use to interface with Magic xpi. As with the **Resources** section, the **Services** section is also found in the **Settings** dialog box. The **Settings** dialog box can also be opened from the **Start** menu in stand-alone mode.

This lesson contains various topics, including:

- The Services section of the **Settings** dialog box
- Service types
- Defining a new service

## Services Section

The main purposes of a centralized **Services** section are:

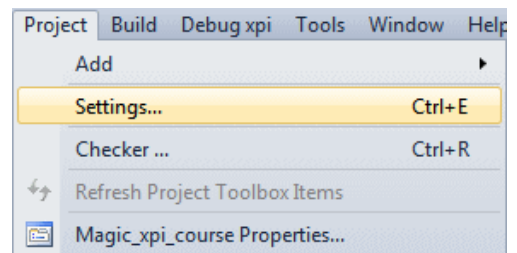
- **Reusability** – A service can be used more than once throughout the project. For example, an external queue manager such as MSMQ can be used in several places. Instead of defining and maintaining the definitions for each instance separately, you can define it once.
- **Flexibility** – The services are defined in a separate file and can be configured outside the project. When migrating the project between different operating environments, no changes are required to the project.
- **Ease of maintenance** – All services are defined in one central location. You do not need to look throughout the project to locate where services are being used.

The **Services** section provides a list of all services that the project provides to external sources. It is good practice to add a brief description to make the list more meaningful.

The **Services** section is accessed by clicking the **Project** menu and selecting **Settings**.

The **Services** section is similar to the **Resources** section, and is divided into two sections:

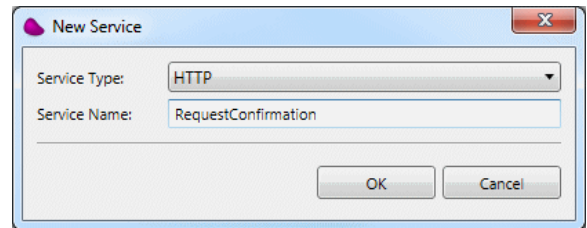
- Service definitions (left pane)
- The specific configuration for that service (right pane)



A service often connects to an external resource. Therefore in many cases you will need to define a resource before the service.

To add a new service:

1. Click **Add**.  
The **New Service** dialog box opens.
2. The **Service Type** dropdown list contains a list of all the predefined service types.



Magic xpi has a predefined a list of service types. For each service type there is a list of configurable properties.

These definitions are per Magic xpi installation and affect all projects.

The definitions are maintained in an XML file, **service\_types.xml**, located in the Magic xpi installation folder.



It is not recommended to make changes to this file.

3. Select **HTTP** from the dropdown list.
4. You are prompted to enter the **Service** name. Enter **RequestConfirmation**. This is the name that will be used throughout the project. It is advisable to provide a meaningful name so that it will be easy for you to identify the service.

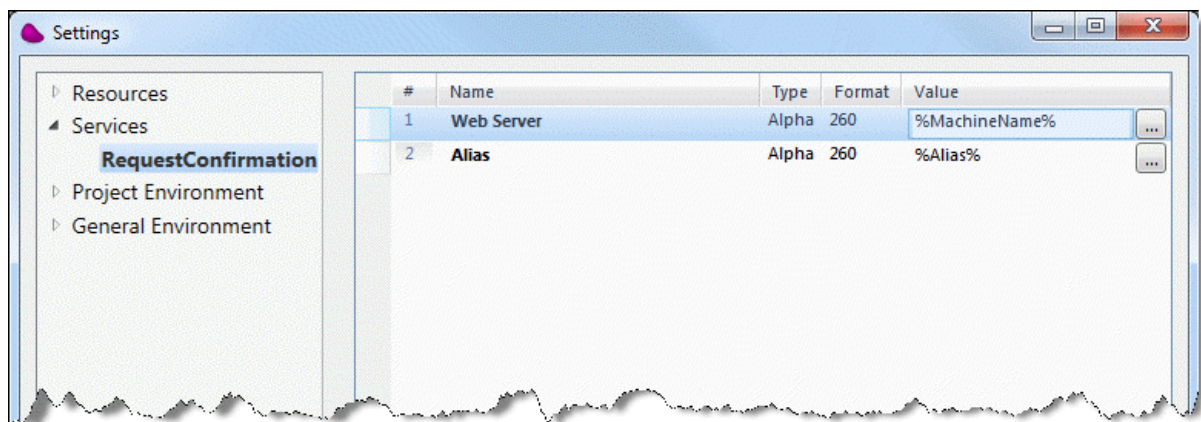
Once the service has been added, you need to configure it. The configurable settings for each service type are different.

As an example, you will continue with the HTTP service that you have added. When you park on the service, you will see the list of properties for that service. As has been discussed, a service is an external resource and is often first defined in the **Resources** section of the **Settings** dialog box. You will use this service in the next lesson, when you define an HTTP trigger.

For HTTP, there are two parameters:

- **Web Server** – This is the name of the website where the Magic xpi requester resides. The default value is **%MachineName%**. This is a predefined environment variable, and its default value is the name of the local machine.
- **Alias** – The website's alias name. The default is **%Alias%**, this is a predefined environment variable. Its default value is **Magicxpi4.5**. You will leave the default values unchanged.

There is no limit to the number of configurations that can be created for a single service type.



## HTTP Endpoints

When using the HTTP service, an external HTML form calls Magic xpi and activates a flow. This form of invoking a flow is called a trigger, and will be discussed in detail in the next lesson.

When calling such a trigger, the calling HTML form may submit additional information required by Magic xpi to service the request. This additional information has to be predefined in Magic xpi. Each such predefined set of passed values, their names and types is called an endpoint.

### The Trigger Activation

The trigger is activated when an HTTP request is received.

To activate the trigger and consequently invoke the flow, the URL must be defined in the following way:

```
http://<ComputerName>/<Magic xpi instance>/MgWebRequester.dll?appname=IFS  
ProjectName&prgname=HTTP&arguments=-AService Name%23Endpoint  
name&Argument_Name = Argument_Value
```

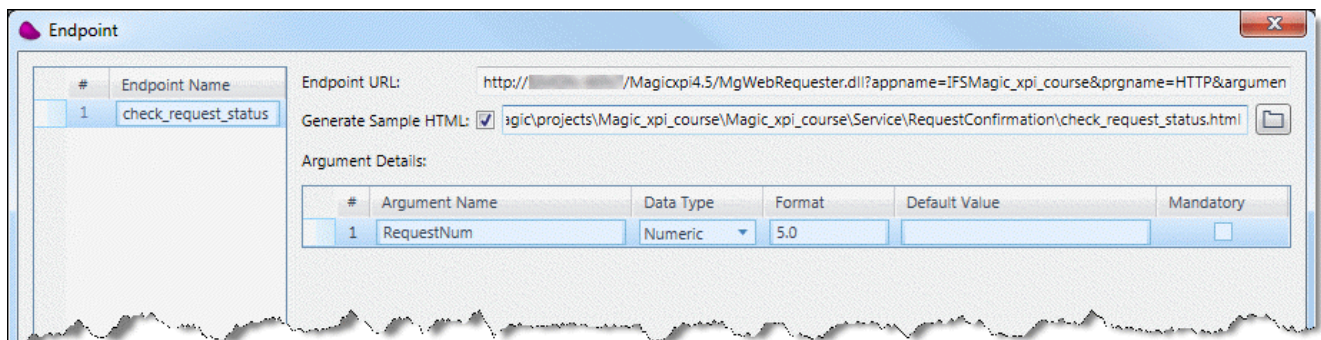
Where:

- **ComputerName** is the name of the local computer where Magic xpi resides.
- **Magic xpi instance** is the Magic xpi alias. This will be **Magicxpi4.5** in the current version.
- **/MgWebRequester.dll?appname=IFS** is constant and must be part of the address.
- **ProjectName** is the name of the current project.
- **&prgname=HTTP&arguments=-A** is constant and must be part of the address.
- **Service Name** is the name of the HTTP service in the **Services** section.
- **Endpoint name** is the name you gave to the endpoint.
- **Argument Name=value** is the name of an argument passed to the trigger with its corresponding value.

For example, the following is a legal address:

**http://localhost/Magicxpi4.5/MgWebRequester.dll?appname=IFSMagic\_xpi\_course  
&prgname=HTTP&arguments=-ARequestConfirmation%23check\_request\_status&  
RequestNum=5**

You can also send a request to the Server by using a form and hidden input tags for the arguments. You can generate a sample form by selecting the **Generate Sample HTML** checkbox in the **Endpoint** dialog box. You will use it in the exercise at the end of this lesson.



The generated sample HTML file will be created in the following directory:  
**%currentprojectdir\Service\<<Service Name>**, where Service Name is the name of the service, in this case, **RequestConfirmation**. You can define your own path.

## Exercise

1. Create an endpoint called **check\_request\_status**.
2. Create an **HTML** page to invoke the service by using the **Generate Sample HTML** option.
3. Pass the request number in a variable called **RequestNum**.

## Summary

In this lesson, you learned about:

- Magic xpi services.
- How to maintain services using the **Settings** dialog box's **Services** section.
- How to create new services.
- The HTTP service and how to add an endpoint.



Magic®  
University



# Lesson 12

## Checking Request Status

Sometimes, in an integration scenario, human intervention is required. The transaction is held in persistent memory, such as a database, until reviewed by a person.

In this lesson, you will learn how to invoke a flow by using an external triggering mechanism and will see an example of this from an Internet browser. You will learn how to create a dynamic HTML file which you will return to the calling Internet browser. While doing this, you will learn a method of implementing a scenario of human intervention.

This lesson contains various topics, including:

- Magic xpi triggers.
- Defining an HTTP trigger.
- Creating an HTML response using HTML merge.

## Intervention Process

A simple human intervention process includes pausing at a certain point. When there is a human response, the process proceeds from the point that it was stopped at, restoring the relevant information at the paused point.

In previous lessons, you learned how to log incoming requests into the database and into the ODS. You also learned how to check if the customer exists in the MSSQL database, and how to check the availability of the items.

If the customer did not exist or not all of the items were in stock, the request was considered to be **unapproved**.

For unapproved requests, human intervention is needed to approve them. Therefore, in this lesson you will learn how to develop a process for human intervention.

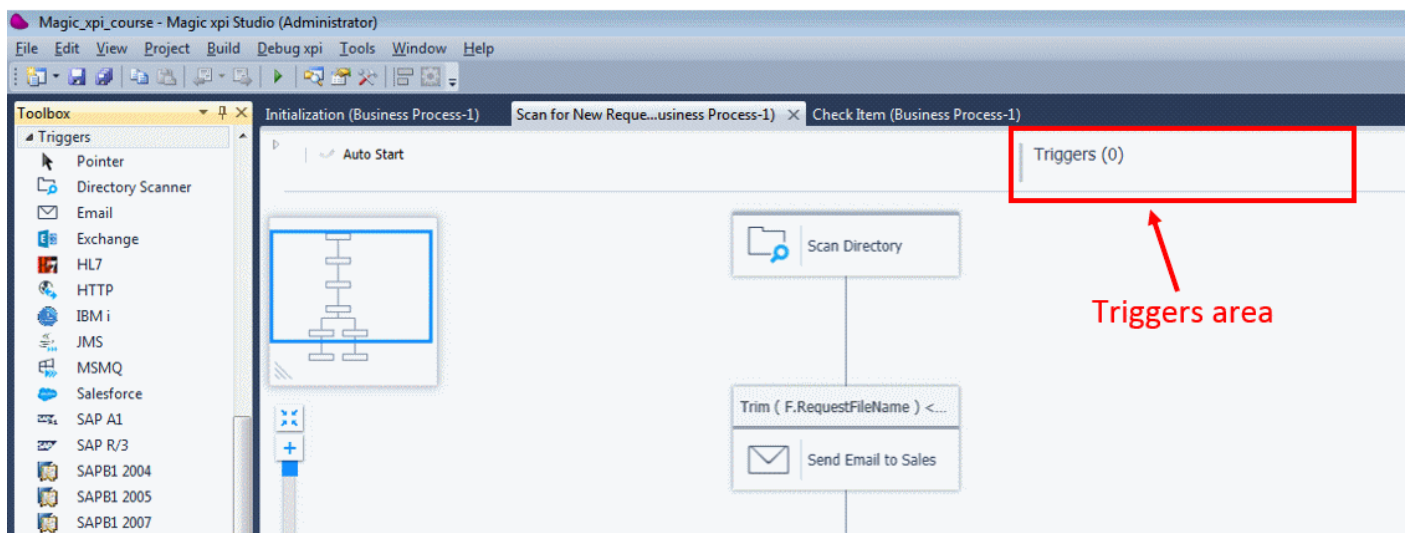
When there is a response from the user, the process is continued from the point it was stopped, the process information is loaded, and the process continues according to the user's response.

## Magic xpi Triggers

A flow in Magic xpi can start in many different ways. For example, a flow can be:

- Set to start automatically when Magic xpi starts.
- Called from another flow.
- Subscribed to a publication. (This will be discussed in a later lesson.)
- Started on a time schedule. (This will be discussed in a later lesson.)
- Triggered by an event.

The Flow Editor provides a special section to define flow triggers. When you include components in the Triggers area section of the Flow Editor, these components serve to activate the flow when the trigger component receives the appropriate information.



## Trigger Types

The Flow Editor's main pane is divided into two areas where you can place components. You can add components in the **Flow** area, or you can add components in the **Triggers** area.

When you include components in the **Triggers** area, these components activate the flow when the trigger component receives the appropriate information.

For example, you can place an HTTP component in the **Triggers** area. When the HTTP request is received, the HTTP trigger is activated and invokes the rest of the flow that it is placed in.

Some of the components that you can use as triggers are:

- Directory Scanner
- JMS
- SAPBI
- Email
- MSMQ
- Web Services
- HTTP
- Salesforce
- WebSphere MQ

Another flow trigger is provided by the **Scheduler** utility, which will be discussed in a later lesson.



You can include up to ten trigger components in your flow.

## HTTP Triggers

Before you can add the HTTP component, there is some preparation work you need to do. You will create a flow that will respond to a user's request for the request status. The response will be a generated HTML page containing the requested data.

The HTML file will contain the customer details, and whether the items are in stock. If the customer does not exist, the customer's name will be contained in the HTML as a hyperlink, requesting the addition of the customer's name to the database. Processing that part of the request will be discussed in a later lesson.

1. In the Solution Explorer, right-click on **Business Process-1** and select **Add Flow**. Name the flow: **Check Request Status**



If you want this flow to be above the **Initialization** flow, you can park on the flow and select **Move Flow Up** from the context menu.

Next, you will set four flow variables that will be used in the flow, and one environment variable that points to the **templates** folder location.

Under the **Check Request Status** flow:

2. Double-click **Flow Variables** to open the **Flow Variables** repository.
3. Add the following flow variables:
  - **F.HTTP\_Request\_Num** – Numeric of size 5.
  - **F.CustomerId** – Numeric of size 12.
  - **F.CustomerExists** – Logical.
  - **F.RequestExists** – Logical. Give this a default value of 'FALSE'LOG.
  - **F.RequestHeader** – Alpha of size 255.
  - **F.Result\_HTML** – BLOB.
4. From the **Project** menu, select **Settings**.
5. Under the **General Environment** section, click **User Environment Variables**. Click **Add** to add an entry named **templates** with a value of `%currentprojectdir%course_data\templates%sl%`.

## The HTTP Component

The HTTP component retrieves information from a specified URL.

The component can work in two modes:

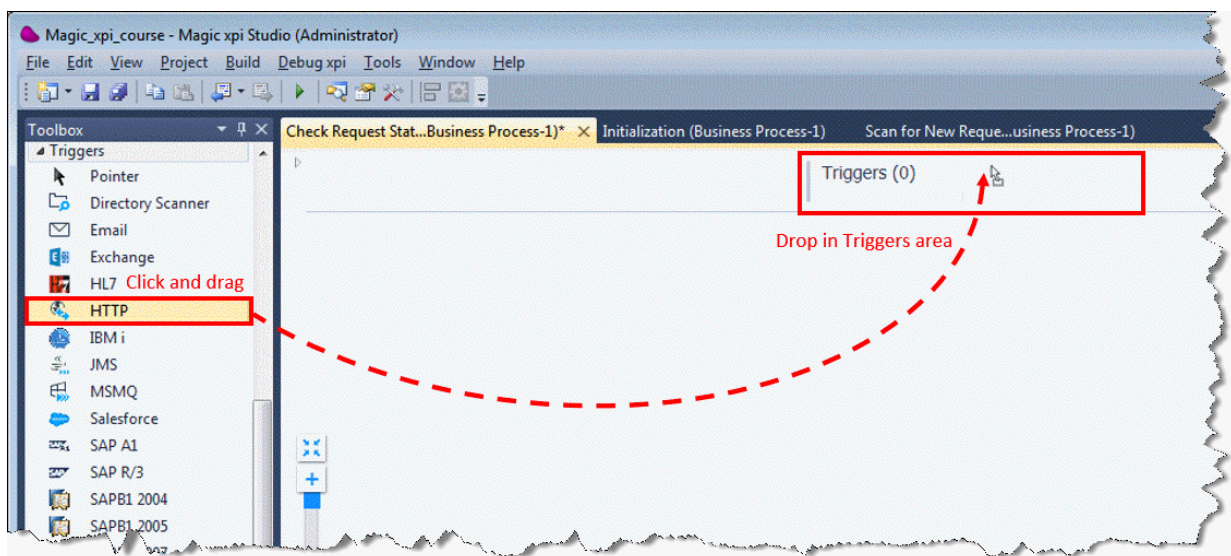
- **Step mode** – The component can:
  - Work in **Get**, **Post**, and **Rest** modes.
  - Retrieve information according to search criteria.
  - Employ multiple search strings, such as **Start** and **End** texts.
- **Trigger mode** – The HTTP component is configured to define the properties that enable you to use the HTTP requester.

As a trigger component, the HTTP component is used to trigger a flow. It also enables the passing of parameters through the Web.

The flow is triggered when the trigger component receives a request from the Web. The request is sent in a specific syntax, and includes the arguments and parameters that are to be updated.

To add the HTTP trigger to the flow:

1. In the Solution Explorer, park on the **Check Request Status** flow.
2. Drag the **HTTP** component from the **Toolbox** pane to the **Triggers** area.



Right-click on the HTTP trigger to open its **Properties** pane.

3. In the **Trigger Name** field, type **Receive HTTP Calls**.
4. In the **Setting** section, select the **RequestConfirmation** service.
5. Double-click the HTTP trigger, or right-click on it and select **Configuration** from the context menu.

The **Component Configuration** dialog box opens. The dialog box has two sections, the **Service Details** and the **Argument Details**.


In the **Service Details** section, you select the endpoint. The argument list will change according to the selected endpoint.

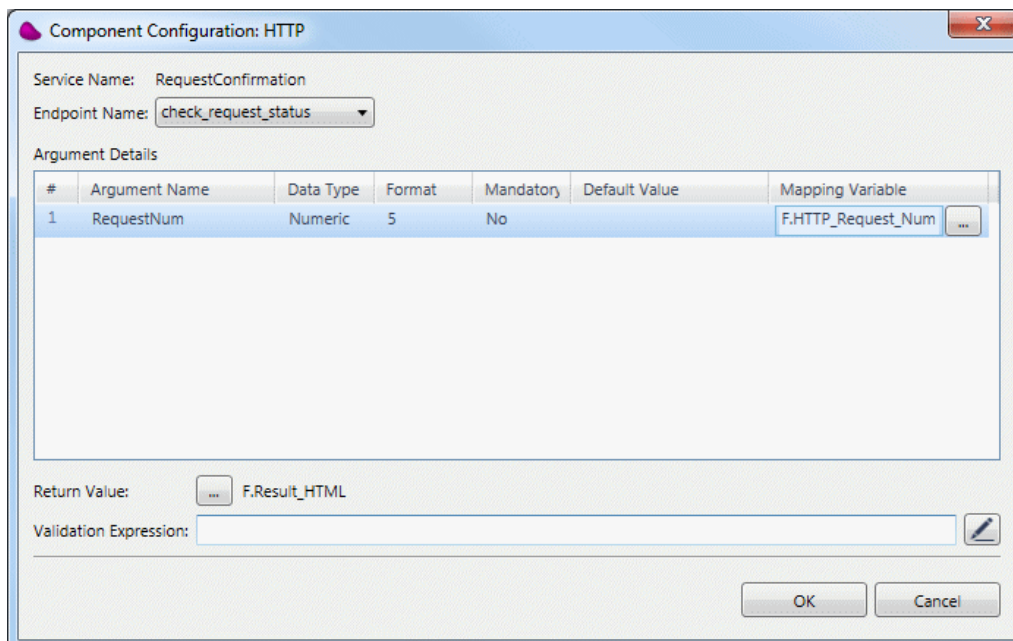
- The arguments will be mapped to an internal Magic xpi variable by selecting the **Variable Selection** box in the **Mapping Variable** column.
- These are the parameters that are sent from the external Internet browser to Magic xpi, and will then be used within the flow. Each mandatory argument has to be mapped to a variable.
- The **Return Value** is a BLOB variable that will be passed back to the Internet browser for display. This is in the form of a valid HTML script.
- The **Validation Expression** defines limits to the trigger. You can open the Expression Editor to create an expression. In this way, you can create conditions for triggering the flow by creating a validation expression.

When you click the variable selection buttons, you will not see any of the variables that you used in the **Scan for New Requests** flow. This is because you defined these variables as flow variables, and flow variables are localized to the specific flow in which they were defined.

6. In the **Endpoint Name** property, select **check\_request\_status**.

In the **Argument Details** section:

7. Park on the **Mapping Variable** column and click the  button. Select **F.HTTP\_Request\_Num** from the selection list.
8. Park on the **Return Value** variable and select the **F.Result\_HTML** variable.
9. Click **OK** to close the dialog box.





Component Configuration: HTTP


Service Name: RequestConfirmation

Endpoint Name: **check\_request\_status**

Argument Details

#	Argument Name	Data Type	Format	Mandatory	Default Value	Mapping Variable
1	RequestNum	Numeric	5	No		F.HTTP_Request_Num 

Return Value:  F.Result\_HTML

Validation Expression:  

OK Cancel

## HTML Response Page

You are going to build the header string for the response page, specifying the customer's details. To accomplish this, you will extract the data from the database using the Data Mapper utility.

1. Park on the **Check Request Status** flow.
2. Add a **Data Mapper** utility as the first step in the flow, and name it **Get Information from DB**.
3. In the **Description** field you can type: **This step generates the header of the request for the returned HTML. It also checks whether the request is a problematic one.**
4. Double-click the step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

5. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
6. Right-click on the Database source and select **Show Properties**.
7. Enter **Scan\_Requests** in the **Name** property.
8. In the **Database Definition** field, select the **Course Database** entry.
9. Click **Wizard**:
  - a. Note that the **DB Operation** property is set to **Select** by default. This is the only option for sources.
  - b. Select the **Requests** table.
  - c. Select the **CustomerID**, and **CustomerExists** columns.
  - d. You need to select only the request entry that the user requested when they passed the Request ID. So, the WHERE clause will be:  
**[Requests].RequestId = <?F.HTTP\_Request\_Num?>**



If the customer exists, you are going to add the customer details to the returned HTML. But, if the customer does not exist, you want to have a hyperlink where the user can click the hyperlink and therefore add the customer.

To do this:

10. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Destination Tree** area.
11. Right-click on the Variable destination and select **Show Properties**.
12. Enter **Create\_HTML\_Header** in the **Name** property.
13. Select the **F.RequestExists**, **F.CustomerExists**, **F.CustomerId** and **F.RequestHeader** flow variables.
14. Expand the source and destination.
15. Map **CustomerId** to **F.CustomerId** and to **F.RequestHeader**.
16. Map **CustomerExists** to **F.CustomerExists** and to **F.RequestHeader**.
17. When the Data Mapper executes, it will fetch only those records where **[Requests].RequestId = <?F.HTTP\_Request\_Num?>**. Therefore, set the **F.RequestExists** variable's **Calculated Value** field to **'TRUE'LOG**.

You can now deal with the **F.RequestHeader**:

18. Right-click on the **F.RequestHeader** entry in the destination pane, select **Show Properties**, and enter the following expression in the **Properties** pane's **Calculated Value** field:  

```
IF (Src.S1/Record/CustomerExists, '', '<a href="http://localhost/Magicxpi4.5/MgWebRequester.dll?appName=IFSmagic_xpi_course&prgname=HTTP&arguments=-ARequestConfirmation%23add_customer&RequestNum=' & Trim (Str(F.HTTP_Request_Num,'5')) & ' "> ' & Trim ( Str (Src.S1/Record/CustomerID, '9') ) & '</a>' )
```

The **Calculated Value** checks whether the customer exists, and then adds a hyperlink under the *Customer ID*, if the customer does not exist.

The expression above seems complex, so a little explanation is necessary. To add a hyperlink under the CustomerID value, you need to use an HTML tag called an anchor. This has a syntax of **<a href="link">text</a>**. The double quotation marks, **"**, are important here. In the example above, the link will be **'<a href="http://localhost/Magicxpi4.5/MgWebRequester.dll?appName=IFSmagic\_xpi\_course&prgname=HTTP&arguments=-ARequestConfirmation%23add\_customer&RequestNum=' & Trim (Str( F.HTTP\_Request\_Num,'5')) & ' ">'**

You are using the HTTP service named **RequestConfirmation**. You can see the addition of the **add\_customer** endpoint to the hyperlink.

You have checked the request information, and you fetched the customer identification from the database. Now you need to fetch the customer information.

1. Add a Data Mapper utility as child step of **Get Information from DB**. Name it **Get Customer Information**.
2. Double-click the step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

3. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
4. Right-click on the Database source and select **Show Properties**.
5. Enter **Fetch\_customer** in the **Name** property.
6. Click **Wizard**:
  - Select the **Customers** table.
  - Select the **CustomerName** column.
  - You need to select a certain customer. You fetched the ID in the previous step. The WHERE clause will be:  
**[Customers].CustomerID=<?F.CustomerId?>**

If the **CustomerId** does not exist, this Data Mapper step will not fetch any details.

7. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Destination Tree** area.
8. Right-click on the Variable destination and select **Show Properties**.
9. Enter **Create\_HTML\_Header** in the **Name** property.
10. Select the **F.RequestHeader** flow variable.
11. Expand the source and destination.

When the Data Mapper executes, it will fetch only those records where **[Customers].CustomerID=<?F.CustomerId?>**.

12. Connect the **CustomerName** to **F.RequestHeader**.
13. Right-click on the **F.RequestHeader** entry in the destination pane, select **Show Properties**, and enter the following expression in the **Calculated Value** field:  
**IF ( F.CustomerExists, Src.S1/Record/CustomerName , F.RequestHeader)**

This is an important expression. If the customer is found in the database, you will update the HTML with the name fetched from the database. But if the customer does not exist, you will use the information that you updated in the previous step.

## Templates

In the previous section, you updated a variable named **F.RequestHeader**. The response to the request is returned in a variable named **F.Result\_HTML**. You need to update the **F.Result\_HTML** variable with the request information. This will be pure HTML data. You will do this by using a pre-defined template.

In the `My Documents\Magic\projects\Magic_xpi_course\Magic_xpi_course\course_data` folder, you will find the **templates** directory, which includes a template of an HTML file:

**RequestStatus.tpl**.

A template is a file with special tags that function as place holders. Magic xpi recognizes these tags, and enables you to map data into them using the Data Mapper. The result is based on the content of the template, with all the place holders being replaced with actual data. The process is similar to the mail merge feature found in most word processors.



The Magic xpi destination for a template file is: **Template**.

Merge tags have the following generic form:

**{Token Prefix}{Token\_name}{Token Suffix}**.

In Magic xpi, the Token Prefix and Suffix are defined as **<!\$** and **>** respectively.

- **<!\$MG\_name>** – The name part of the tag is represented as a node in the destination schema in the Data Mapper utility.
- **<!\$MGREPEAT>** – Defines the beginning of a repeated area that is represented in the Data Mapper utility as a compound. The repeated area is duplicated and processed for each iteration of the connected compound, thereby allowing for an unknown number of data rows.
- **<!\$MGENDREPEAT>** – Defines the end of a repeated area and is not represented in the Data Mapper utility.

You can also define a section of the file that will be displayed in the final output according to a logical expression.

- **<!\$MGIF\_name>** – Defines the start of an IF block. This is logical data. If the data is True, the rest of the IF block is processed. You can have nested IF blocks. The MGIF simply defines the start of a block, and the tag itself will not appear in the output.
- **<!\$MGENDIF>** – Defines the end of an IF block. If there is an IF block, this is mandatory.
- **<!\$MGELSE>** – Defines the start of an ELSE block and the end of an IF block, which must precede the ELSE block. The ELSE block is processed if the name data value of the IF block evaluates to False. This tag is optional. The tag itself is removed from the output.

You will define the template in the lesson exercise.

A Merge Template file will look similar to the image below. Merge Tags are outlined in red:

```
<table border="0">
  <tr>
    <td colspan="4" width="100%">
      <h1 align="center">Getting Started </h1>
    </td>
  </tr>
  <tr>
    <td>
      <h2><!--$MG_Header--> </h2>
    </td>
  </tr>
  <tr>
    <td>Request sent by <b><!--$MG_CustomerName--></b>. <br>
    Total cost of request: <b><!--$MG_RequestCost--></b>.
  </td>
  </tr>
  <tr>
    <td>
      <h3>Items:</h3>
      <table cellpadding="6" border="0" bgcolor="#C0C0C0">
        <tr>
          <td bgcolor="#000080" width="100"><font face="Tahoma" color="FFFFFF"><strong>Code</strong></font></td>
          <td bgcolor="#000080" width="300"><font face="Tahoma" color="FFFFFF"><strong>Name</strong></font></td>
          <td bgcolor="#000080" width="100"><font face="Tahoma" color="FFFFFF"><strong>Price</strong></font></td>
        </tr>
        <!--$MGREPEAT-->
        <tr>
          <td bgcolor="FFFFFF"><!--$MG_ItemCode--></td>
          <td bgcolor="FFFFFF"><!--$MG_ItemName--></td>
          <td bgcolor="FFFFFF"><!--$MG_ItemPrice--></td>
        </tr>
        <!--$MGENDREPEAT-->
      </table>
    </td>
  </tr>
</table>
```

## Exercise

1. Complete the creation of the HTML Response page by defining a **Template** destination.
2. Map the **F.RequestHeader** and request items to the template.
3. Transfer a static HTML response page in case the request was not found.
  - a. Use the File Management component to accomplish this.
  - b. Define a new environment variable named **Files**, which points to:  
`%currentprojectdir%course_data\Files%sl%`  
In this directory you will see a file named **RequestNotFound.htm**.  
This is the file that must be returned if a request was not found in the database.
4. Test your project using the HTML page that you created in a previous lesson. You can find the HTML file in the directory: `%currentprojectdir%Service\RequestConfirmation`.

## Summary

In this lesson, you learned about:

- Magic xpi triggers.
- Implementing a human response situation.
- Templates.



Magic<sup>®</sup>  
University

# Lesson 13

## Error Handling

An integration application, in most cases, has no control over the source systems and input formats, and the availability of consumer systems. Errors can occur in many different areas of the process.

The integration project needs to be able to detect and identify potential errors as soon as they occur, report those errors, and continue to work and stay stable.

The monitoring, processing, and handling of error situations are an integral part of any project.

Magic xpi's error management includes:

- Context error variables
- Error behavior in flows and steps
- Dedicated error flows

## Magic xpi Error Handling

Magic xpi error handling is performed on:

- Step error behavior
- The flow level
- The error flow level

### Context Error Variables

You learned that there are some context variables that are used for error handling. The predefined context variables return error information that you can use to determine the course of action.

The values in the following variables can only be used in the condition following the step that raised the error. They are cleared before entering the next step:

- **C.sys.ErrorCode** – The error code.
- **C.sys.ErrorDescription** – The error description.

The following variables assist you in identifying the origin of the error:

- **C.sys.LastErrorFlowName** – The name of the flow where the last error occurred.
- **C.sys.ErrorStepName** – The name of the step that invoked the last error.

Finally, the following variables hold information about the last error. They are only updated when a new error occurs:

- **C.sys.LastErrorCode** – The last error code. This remains until there is a new error.
- **C.sys.LastErrorDescription** – The last error description. This remains until there is a new error.
- **C.sys.LastErrorInfo** – Contains full information about the error. It is a BLOB and can contain an SQL statement, a Java error, and others.



## Step Error Handling

The Magic xpi step error behavior is **Exit** (except for Data Mapper and SAP B1 steps where you can set the error behavior in the **Error Behavior** property to **Exit** or **Continue**). When an error occurs in the step, the step exits. If there is more than one method in the step, the step will end immediately when an error occurs and will not continue to the next method.

The simplest way of checking for an error is by checking the error code contained in the error variables. You check the value in **C.sys.ErrorCode** before entering the next step.

Follow these steps to see how this works:

1. Park on the **Scan for New Requests** flow. Add an **Email** component as a child step of the **Extract Details from Request** step.
2. Name the step: **Send Thank You Email**
3. Set the **Processing Mode** to **Parallel**.
4. Select the **Method** interface.
5. Double-click the step, or right-click on it and select **Configuration** from the context menu.
6. Click **Add**. Select **Quick Send** and set the following:
  - a. **To:** F.CustomerEmail
  - b. **Subject:** 'We received your request.'
  - c. **Body:** 'Thank you for your request. We are currently checking into it.'&ASCIIChr (10)&'MSU Computers'



**ASCIIChr (10)** enables you to add a line break in the text.

Now what happens if the customer sent an invalid email address?

7. Add an **Email** component as a child step of the **Send Thank You Email** step. Name the step: **Send Error Email**.
8. Select the **Method** interface.
9. Double-click the step, or right-click on it and select **Configuration** from the context menu.
10. Click **Add**. Select **Quick Send** and set the following:
  - a. **To:** postmaster@magic.xpi.course.com
  - b. **Subject:** 'Email to the customer was returned.'
  - c. **Body:** 'This is an automatic notification. Please do not reply.'&ASCIIChr (10)&'An email to a customer failed.'

11. Set the condition for this step as: **C.sys.ErrorCode = 102**

When you run this flow with an invalid email address, you will get an error that is handled by this flow.

## Flow Level Error Handling

With step error handling, you can test for a certain error and then decide whether to continue to the next branch. This can be useful, but you might find that you need to use the same condition often. As an example, if a particular step had three parallel child steps, you would need to use the condition three times. Of course, in this case, you could use the NOP service.

In flow level error handling, Magic xpi enables you to define automatic behavior for certain errors. You can define an error policy for a range of error codes in the Errors Policies repository. This repository is accessed by double-clicking **Error Policies** under the relevant flow.

When an error occurs that falls in that range, the predefined error policy will be invoked.

You can decide between **Abort**, **Ignore**, **Restart Flow**, **Retry**, and **Jump**.



**Jump** enables you to go to a section of the flow that is not normally accessible. You may have a branch in the flow where the condition is false and can only be reached by using **Jump**.

When you use the **Jump** method, the flow continues from that step and does not continue with the step that raised the error.

## Component Type

Each component has an internal component ID and a component type. You can see these values in each component's **Properties** pane.

It is possible to have the same error codes in different components. A specific error code depends on the component type.

The **C.sys.LastErrorComponentType** variable contains the component ID of the last component that returns an error.

Follow these steps to see how this works:

1. Park on the **Scan for New Requests** flow.
2. Park on the **Send Error Email** step and select **Copy** from the context menu.
3. Park on the **Scan Directory** step (the first step in the flow) and select **Paste**. The step is now moved as a new child of the Scan Directory step.
4. Delete the old **Send Error Email** step.

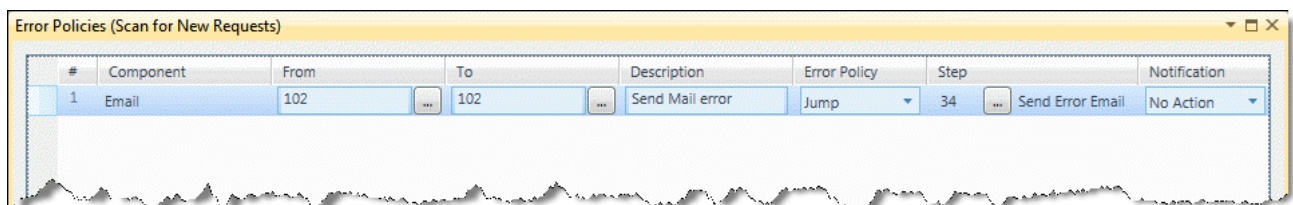
In the last example we added a condition for this step as: **C.sys.ErrorCode = 102**.

5. Replace the **C.sys.ErrorCode = 102** condition with **'FALSE'Log**.

By conditioning this step with FALSE, you are ensuring that this step or branch is unreachable. It is, in a sense, disabled.

6. Under the **Scan for New Requests** flow, double-click **Error Policies**.
7. in the Error Policies repository, click **Add**.
8. Click the zoom button in the **From** column. This loads the **Errors Repository** dialog box, where you can select an error for a specific component.
9. From the dropdown list, select **Email**. Only the error types for the Email component will now be displayed.
10. Select **102 Send Mail Error**.
11. Repeat step 8 for the **To** property.
12. In the **Error Policy** column, select **Jump**.
13. Click the zoom button in the **Step** column. You see only the steps in the current flow. Select the **Send Error Email** step.

This step will not be executed in the normal sequence because the condition is False. However, with the error policy that you defined, you will be able to jump to the step and carry out a sequence of steps.



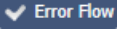
#	Component	From	To	Description	Error Policy	Step	Notification
1	Email	102	102	Send Mail error	Jump	34	Send Error Email

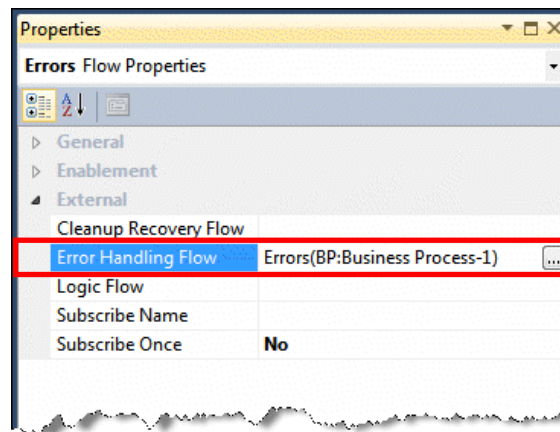
You can test this in the same way that you did the previous example.

## Dedicated Error Flow

You can define a dedicated flow that will handle the error. This can be any Magic xpi flow. An error flow provides a high level of error handling in which you can define your own mechanism for handling the error.

- You first define a flow that will be used as an error flow.
- Then you select the flow that will be using the error flow.

You select a dedicated error flow in the **External** section of the **Properties** pane of the flow which needs the services of an error flow. When you assign a specific flow to handle error codes, the  icon will appear to the left of the Trigger area.



In the error flow, `C.sys.LastErrorCode`, `C.sys.LastErrorDescription`, and `C.sys.LastErrorInfo` hold the values of the last error.

To see an example of this mechanism:

1. Create a flow named: **Errors**
2. Drag the **Email** component as the first step in the flow. Rename the step to: **Send Email to Administrator**
3. Double-click the step, or right-click on it and select **Configuration** from the context menu. To send an email, you need to configure the component using the **Method** interface. Use the **Quick Send** method and set the following:
  - a. **To:** postmaster@magic.xpi.course.com
  - b. **Subject:** 'Email to the customer was returned'
  - c. **Body:** 'This is an automatic notification. Please do not reply.'&ASCIIChr(10)&'An email to a customer failed.'

Since the flow will be invoked each time an error is raised, you need to condition the step according to the error that you want to handle.

4. Set a condition for the **Send Email to Administrator** step, and set the condition with the following expression: **C.sys.LastErrorCode = 102**.

Once the error flow ends, the focus returns to the flow that raised the error. It then continues with the following step, the one after the step where the error occurred. The error flow returns clears the error.



When using the Data Mapper or a component that supports multi-operations, each record can invoke the error flow. When the error flow is completed, the process will either move to the next record or leave the step, depending on the step error behavior, which can be Continue or Exit.



The error flow example provided in this lesson displays the handling of a single error. However, you can create a general error flow that will handle different types of errors.

## Returning the Flow with an Error

Although the flow handled the error, and in essence cleared the error, you may need to return a different error, or even the same error, to the flow that raised the error.

You can decide whether to end without an error (the default), return the same error to the calling flow, or raise a different error.

To raise a different error:

1. Add a Flow Data utility as the last step of the error flow.
2. Update the **C.sys.ErrorCode** variable with **102**. This code must be a code that is compliant with the list of errors for the specific component. The Email component has the following error codes: 100, 101, 102, and 103.

If you need to return the same error code as the one that raised the error, you can update **C.sys.ErrorCode** with the value fetched from **C.sys.LastErrorCode**.

3. When control is returned to the main flow, an error will be raised and handled by the flow error mechanism.



Defining whether the error flow returns with an error or not depends on your own project requirements.

## Exercise

In the previous lesson, you used the File Management component to copy a static HTML file to the BLOB to be returned to the requester. This static file was located in the **Files** folder, and you defined an environment variable that points to that folder.

What happens if that file or folder does not exist?

- If the file or folder does not exist, return an error to the Internet browser.
  - In the **templates** folder there is a **template** named **Error.tpl**. Return this template to the Internet browser.



This is a very specific case and not a general error. Consider establishing an error policy.

## Summary

Magic xpi handles flow errors in the following sequence:

1. Magic xpi calls an error flow if it has been defined.
2. Magic xpi handles the error based on the flow error policy, if the error flow returned with an error.
3. Further steps in the flow logic can be conditioned according to the value returned from the **C.sys.ErrorCode** variable.

# Lesson 14

## Adding a Customer

Web services are described in a special XML file called a WSDL (Web Service Description Language). The WSDL document describes the service, which contains a list of operations including the structure of the input and output.

A Web service can be a commercial Web service or one that is freely distributed.

Magic xpi enables you to use Web services as a consumer and to offer Web services as a provider. Magic xpi's Web service functionality is based on a third party program called Systinet Server for Java (SSJ). This is also referred to as Systinet in this lesson.

In this lesson, you will add a flow that handles the “**add customer**” scenario.

This lesson includes various topics, such as:

- An introduction to Web services.
- The Web service trigger.
- Retrieving information from the ODS.
- The merge HTML response.

## Systinet

Systinet offers strong support for all industry standards, including SOAP 1.1, SOAP 1.2, WSDL 1.1, JAX-RPC, JAXM, SAAJ, JWSDL, WS-Security, WS-Reliable Messaging, and WS-Addressing.

It can handle multiple security mechanisms, including Web Services Security (WS-S). Its authentication support includes HTTP Basic, HTTP Digest, and Secure Socket Layer (SSL).

The authorization model relies on JAAS, which provides maximum integration with existing security frameworks. XML-level security is supported with XML signature, XML encryption, and the Security Assertions Markup Language (SAML).

The Magic xpi installation process also installs a copy of this software. If the installation and configuration of Systinet was successful, Magic xpi will manage all the communication with this software for you.

Systinet creates the following shortcuts in the operating system's **Start** menu:

- The **Start** shortcut executes the Systinet server.
- The **Stop** shortcut halts the Systinet server.
- The **Systinet Enablement Server Administration Console** shortcut brings up the administration program for the Systinet program.

You need to start the Systinet server before using it. You can start the server in one of the following ways:

- The **Start** menu entry
- As a Windows service
- By running: `%WASP_HOME%\bin\serverstart.bat`

Once the server has loaded, you can start the console by doing one of the following:

- Accessing the operating system's **Start > All Programs > Systinet Server for Java** menu, and selecting the **Systinet Enablement Server Administration Console** program entry.
- Entering the following in your browser: <http://localhost:6060/admin/console>

The initial user is **admin** and the password is **changeit**.



## Providing a Web Service

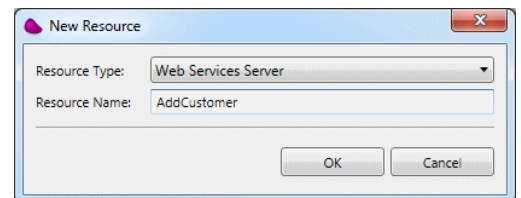
Magic xpi enables two modes when working with Web services:

- **Web Services Client** – Provides the option to use a published Web service. This is also referred to as *consuming* a Web service. You will see this in a later lesson.
- **Web Services Provider** – Provides the option to publish Magic xpi as a Web service, so that other applications can invoke Magic xpi and thereby trigger a flow.

Once you set up Magic xpi as a Web service provider, it will be able to receive requests from any external system, process its request, and provide an answer.

The Systinet Server is the management server where all Magic xpi Web services are deployed. Since this server is external to Magic xpi, the first step is to create a resource containing the connection details to the Systinet server:

1. Create a new **Web Services Server** resource type, named **AddCustomer**.



The resource settings for the Web Services Server resource displays information from your Systinet installation. You can leave the default settings as is.

#	Name	Type	Format	Value
1	Server URL	Alpha	260	http://localhost:6060
2	User Name	Alpha	260	admin
3	Password	Alpha	260	*****

The objective of exposing Magic xpi as a Web Service provider or server is to provide external systems with the ability to request information from a Magic xpi project. You therefore need to create a service.

2. Add a new service and select **Web Services**. You can give it the same name as the resource, **AddCustomer**.

In the settings section:

3. In the **Web Service Provider** setting, select the **AddCustomer** resource that you created.
4. In the **Web Service Name** setting, type: **AddCustomerWS**.
5. Click **Operations** to create the operations or processes that this Web service provides. This is very similar to defining endpoints for the HTTP service.



The service name in the left pane is an internal Magic xpi name used to identify this service in Magic xpi. The **Web Service Name** setting provided in the settings pane is the name of the service that is deployed in external systems, such as Systinet.

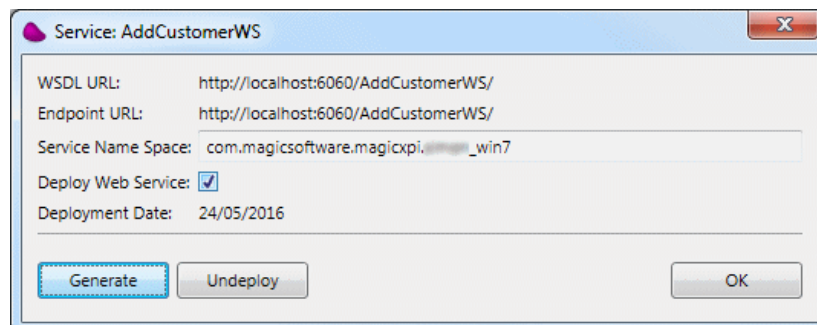
## Operations

You can create as many operations as you want. All the operations will be part of the same Web service that you are setting up and can be called by third party systems.

Each operation can have a single return value and multiple arguments.

To add an operation:

1. In the **Operations** dialog box, click **Add** to add an operation. Name the operation **Add\_customer**.
2. Create an argument, name it **RequestID**, and set the **Direction** to **In**. This must be a Numeric variable of size **5** and an **XSD Type** of **xsd:int**. Remember that Numeric is the internal Magic xpi data type and xsd:int is the data type that is defined in the Web service.
3. Click **OK** to exit the **Operations** dialog box.
4. Click **Management**. This dialog box enables you to generate the specified Web service inside the Systinet program.
5. Click **Generate**.




## Web Service Trigger

You are required to respond to the user's manual request to add a customer. This request is the result of the HTML page containing the hyperlink that you returned in a previous lesson when the customer was not found. The response will be a generated HTML confirmation page of successfully adding the customer.

1. Create a new flow named **Add Customer**. You can move the flow up so that it appears before the **Check Item** flow. This is not a necessity, but by doing so you group all the main flows together at the top of the business process. To move a flow upwards, select **Move Flow Up** from the context menu.
2. Under the **Add Customer** flow, double-click **Flow Variables** to open the **Flow Variables** repository.
3. Add the following variables:
  - **F.RequestNum** – Numeric of size 5
  - **F.RequestXML** – BLOB

When you move a Web Services component into the Triggers area, you are able to define which Web Service operation will invoke this flow.

4. Drag the **Web Services** utility and drop it into the Triggers area of the **Add Customer** flow. Name the trigger **Add Customer by WS**.
5. In the **Setting** section of the **Properties** pane, ensure that the **AddCustomer** service was automatically selected.
6. Double-click the trigger, or right-click on it and select **Configuration** from the context menu so that you can define the parameters. You will see that the Service name, **AddCustomerWS** was automatically defined.
7. Select the **Add\_customer** operation from the dropdown list.
8. You will get the list of arguments that you defined in the service. Park on the **RequestID** parameter and then from the **Mapping Variable** column, click . Select the **F.RequestNum** variable.

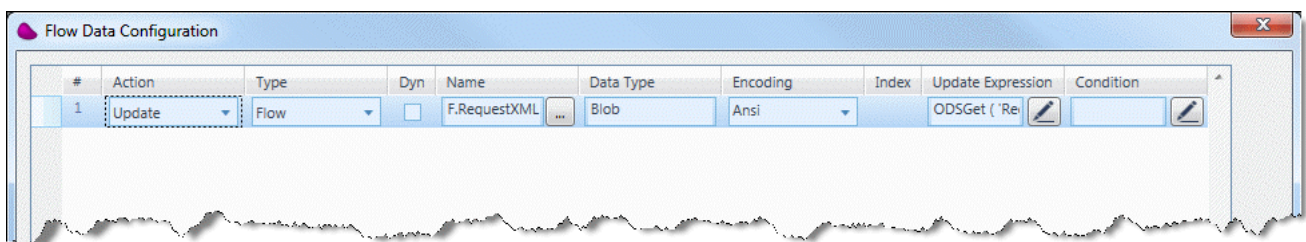
You have now finished creating a Web Service trigger.

## Retrieving Information from the ODS

In a previous lesson, Item Validity Check, you added the request to the ODS for future retrieval. When the current flow is invoked, you need to be able to fetch the request information so that you can extract the necessary information.

In the **Add Customer** flow:

1. Add the **Flow Data** utility to the **Flow** area as the first step in the flow.
2. In the **Name** field, type **Get request XML file from ODS**.
3. Double-click the **Flow Data** utility, or right-click on it and select **Configuration** from the context menu.
4. In the **Flow Data Configuration** dialog box, click **Add** and set the **Action** column to **Update**.
5. Set the **Type** column to **Flow**.
6. In the **Name** column, select the **F.RequestXML** flow variable.
7. In the **Encoding** column, select the required encoding type for the variable.
8. In the **Update Expression** column, type:  
`ODSGet ( 'Request_' & Trim ( Str ( F.RequestNum , '5' ) ) , 1 , 'B' , 1 )`
9. Click **OK**.



As of now, you received the request number as an argument from the Web service, and retrieved the request XML file from the dynamic ODS using the request number.

The next stage is to add the information to the **Customers** table. The source data will be the retrieved XML file currently stored in the **RequestXML** variable.

1. Drag a Data Mapper step as a child step of the **Get Request XML file from ODS** step. Name the step **Add Customer to DB**.
2. Double-click the Data Mapper step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

3. For the **Source**:
  - a. Drag an XML source into the **Source Tree** area.
  - b. Right-click on the XML source and select **Show Properties**.
  - c. Enter **Get\_Customer\_Detail** in the **Name** property.
  - d. In the **XSD File** field, type: `course_data\schemas\request.xsd`
  - e. Set the **Source Type** to **Variable** and select the **F.RequestXML** variable.
4. For the **Destination**:
  - a. Drag a Database destination into the **Destination Tree** area.
  - b. Right-click on the Database destination and select **Show Properties**.
  - c. Enter **Add\_To\_Customers** in the **Name** property.
  - d. Use the Database wizard and select the **Insert** operation. Then:
    - Select the **Customers** table.
    - Select all of the columns.
5. For the **Destination**:
  - a. Drag a Database destination into the **Destination Tree** area.
  - b. Right-click on the Database destination and select **Show Properties**.
  - c. Enter **Add\_To\_Contacts** in the **Name** property.
  - d. Use the Database wizard and select the **Insert** operation. Then:
    - Select the **Contacts** table.
    - Select all of the columns.
6. Expand the source and the destinations.

7. Connect the following nodes:

Source	Destination	Expression
Customer_ID	Customers.CustomerID	
Customer_Name	Customers.CustomerName	
country	Customers.CustomerCountry	
city	Customers.CustomerCity	
street	Customers.CustomerAddress	
zip	Customers.CustomerZipCode	
RequestTotal	Customers.CustomerCredit	
Customer_ID	Contacts.CustomerID	
ContactID	Contacts.ContactID	
ContactName	Contacts.ContactName	
Contact_eMail	Contacts.ContactEmail	

**Note:** You are using the same Data Mapper step to insert into two separate tables.

## Testing the Web Service

To invoke the flow externally as a Web service, you will use the **Systinet** console.

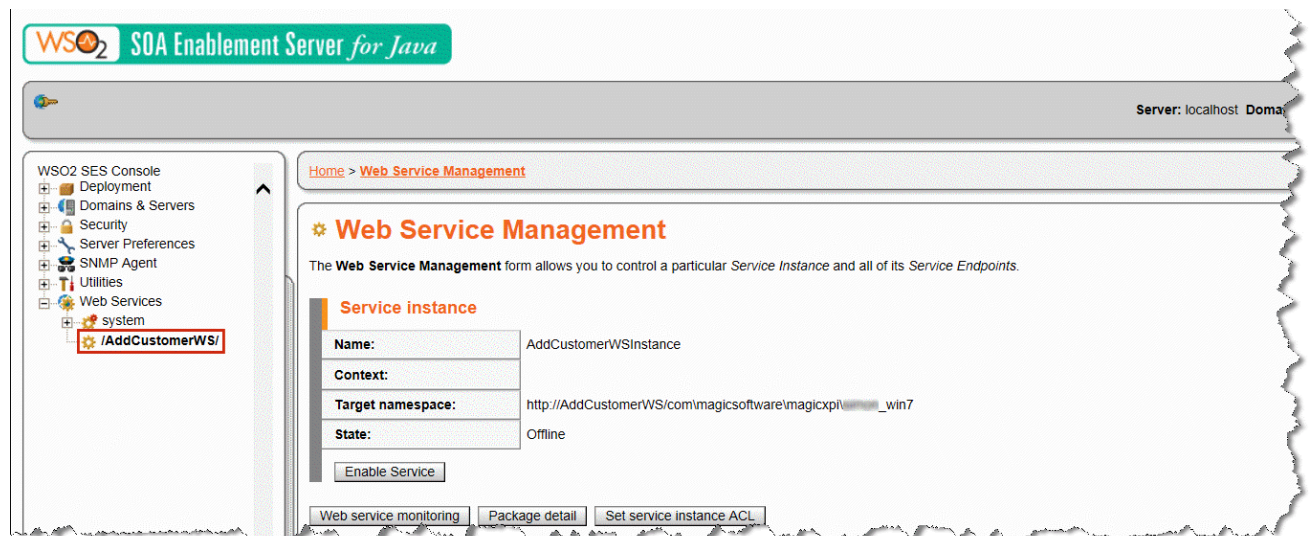
But first, you need to run the project. Remember to:

- Save the project.
- Create an execution file.
- Invoke the Magic xpi Server from the **Start** menu.

You can now invoke the flow from the Systinet console.

1. Start the console by entering the following URL in the Internet browser:  
**http://localhost:6060/admin/console**
2. Enter the user name (the default is **admin**).
3. Enter the password (the default is **changeit**).

Since you deployed the Web service automatically via Magic xpi, the Web service is listed as deployed in the Systinet server.



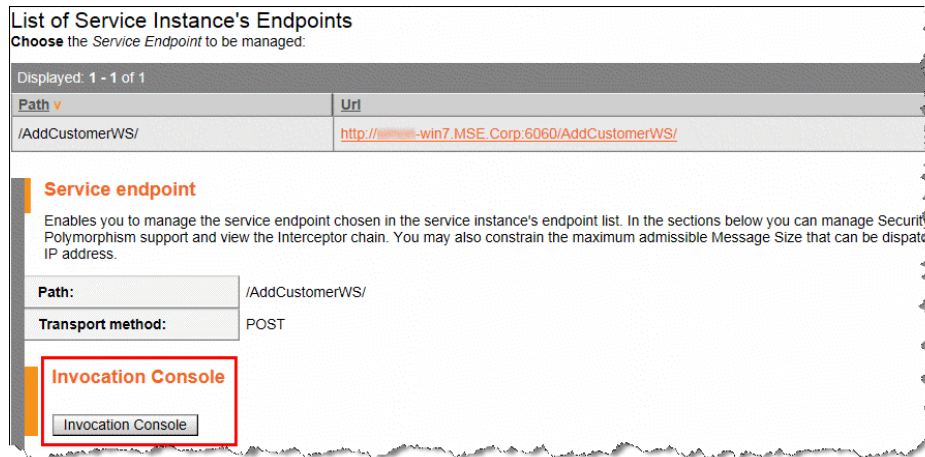
The screenshot shows the SOA Enablement Server for Java console. The main content area displays the 'Web Service Management' page for the 'AddCustomerWS' service instance. The page includes a navigation menu on the left, a breadcrumb trail 'Home > Web Service Management', and a form for managing the service instance. The form fields are:

- Name:** AddCustomerWSInstance
- Context:**
- Target namespace:** http://AddCustomerWS/com/magicsoftware/magicxpi/...\_win7
- State:** Offline

Below the form is an 'Enable Service' button. At the bottom of the page, there are three buttons: 'Web service monitoring', 'Package detail', and 'Set service instance ACL'.

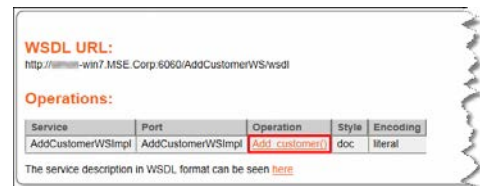


To test the Web service, you need to scroll down the list until you reach the **Invocation Console** section. Click the **Invocation Console** button.



You will now see the operation you defined, **Add\_Customer ()**. The operation has a hyperlink.

1. Click on the hyperlink to enter data.



In this case, you need to enter a value in the **RequestId** field.

### Operation:

Service	Port	Operation	Style	Encoding
AddCustomerWSImpl	AddCustomerWSImpl	Add_customer()	doc	literal

Add\_customer: Add\_customer  RequestID:

**Perform call**

2. Enter a request number and click **Perform call**. The Magic xpi **Add Customer** flow will be invoked.

## Exercise

In a previous lesson, you learned how to accept a request from the Web using the HTTP component. This process returned an HTML page that included a link for adding a new customer if the customer did not exist in the database.

1. Add an HTTP Trigger to the **Add Customer** flow. This will receive the request number and invoke the flow. Return a BLOB variable in the HTTP trigger. To do this, add a BLOB flow variable named **F.RequestHTML**.
2. Return an HTML file to the HTTP trigger, confirming that the customer was successfully added to the database. Use the **CustomerAdded.tpl** template.  
**Hint:** You do this by updating the **F.RequestHTML** BLOB that you defined in step 1.
3. The customer has now been successfully added to the **Customers** table and the **Contacts** table, but the indication of its existence in the **Requests** database is still set to false. Update the **Requests** table.
4. Execute the project again and use the HTML page you created in Lesson 12 to check the status of the request. If there is a hyperlink under the CustomerID instead of a name, click the hyperlink.

## Summary

In this lesson, you added the **Add Customer** flow, which was triggered by a Web service. Using the request number sent from the Web service, you fetched the request data from the ODS and inserted information into the **Customers** table and the **Contacts** table.

You learned how to:

- Expose Magic xpi as a Web service.
- Add an additional trigger to the flow.
- Retrieve information from the ODS.

# Lesson 15

## Handling Approved Requests

Publish and Subscribe (PSS) utilities are a way of distributing information to your network. When an event is published in your integration project, flows that are subscribed to the event are invoked.

The concept is the same as having a magazine subscription. When you subscribe to the magazine, you do not need to constantly check for the latest edition. The publisher sends you the latest edition when the magazine is published. If you are not subscribed to this magazine you will not be 'bothered' with information you don't want. The benefit of using the PSS method is that your systems do not need to constantly refresh and check for new information, and this saves system resources.

In previous lessons, you examined incoming requests and logged them in the database and ODS. Later, you added the option for the user to manually correct faulty requests where the customer did not exist in the database.

In this lesson you will add flows that will process valid requests. These are requests where the customer exists and the items are available in stock.

This lesson covers various topics, including:

- Publishing a topic
- Subscribing a flow to a topic

## Publish and Subscribe Utilities

To subscribe to or publish an event, you need to create a topic in the PSS Topics Repository. This topic represents the event that happened. In the magazine subscription analogy that was mentioned previously, the topic is the magazine name, but not the content itself.

The Magic xpi PSS system contains the subscription information. When a published event occurs, the PSS system sends an invocation request to all the subscribed flows, even on different Magic xpi Servers.



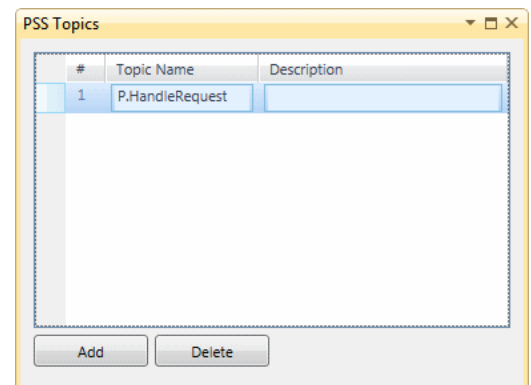
The PSS system is available to all Magic xpi Servers in the same project. This means that the other Magic xpi Servers can retrieve data from this system.

## 'Handle Request' Topic

You will define a new PSS topic, **Handle Request**. If the request is approved by the validation flows, the **Handle Request** topic will be published.

To define a new PSS topic, follow these steps:

1. In the Solution Explorer, under the **Repositories** folder, double-click **PSS Topics** to open the PSS Topics repository.
2. Click the **Add** button to add a PSS topic.
3. In the **Topic Name** column, type **HandleRequest**. The letter **P** is given as a prefix.  
**Note:** The Topic Name cannot contain any of the following characters: # - & \ / < > { } [ ] ( ) , ; " %, a space or three question marks (???).
4. Click **Save**.



## Subscribing a Flow

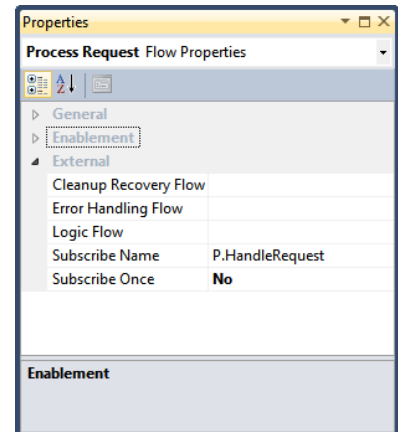
In this section, you will create a new flow: the **Process Request** flow.

When a request is approved, the following steps must be executed:

- Extract the request information.
- Add delivery information to the database.
- Clear the request from the database and delete the dynamic ODS.

To create the **Process Request** flow:

1. Create a flow named **Process Request**.
2. Right-click and select **Properties**.
3. In the **Properties** pane's **External** section, zoom from the **Subscribe Name** property and select the **P.HandleRequest** topic from the **PSS Topic List** dialog box.



A subscribed flow receives three parameters:

- String, passed through C.UserString
- Numeric, passed through C.UserCode
- Blob, passed through C.UserBlob



When the **Handle Request** topic is published, the request number will be passed to the **Process Request** flow in the **C.UserCode** variable.

You will create a flow variable that will hold the request number that was passed through a context variable. You do not need to define a new flow variable since you already have the value in the **C.UserCode** variable. However it is good practice to define your own variable with a meaningful name.

4. Add a flow variable named **F.RequestNum**, type: **Numeric** and size **5**.



During this course, every time you created a flow, you were asked to add a flow variable named **F.RequestNum**. How else could you have handled this?

5. Drag a **Flow Data** utility to the flow area as the first step in the flow.
6. In the **Step Name** field, type **Get request number**.
7. Double-click the **Get request number** step, or right-click on it and select **Configuration** from the context menu.
8. In the **Flow Data Configuration** dialog box, click **Add** and set the **Action** column to **Update**.
9. Set the **Type** column to **Flow**.
10. In the **Name** column, select the **F.RequestNum** flow variable.
11. In the **Update Expression** column, enter an expression for **C.UserCode**.

You will now take the request information and add it to the delivery database.

12. Drag a **Data Mapper** step as a child step of the **Get request number** step. Name the step **Add Delivery Header**.
13. Double-click the **Add Delivery Header** step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

14. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
15. Right-click on the Database source and select **Show Properties**.
16. Enter **Get\_request\_header** in the **Name** property.
17. Use the Database wizard:
  - a. Select the **Requests** table.
  - b. Select the **CustomerID** column.
  - c. Set the WHERE clause to: **[Requests].RequestId=<?F.RequestNum?>**
18. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
19. Right-click on the Database source and select **Show Properties**.
20. Enter **Add\_To\_Delivery** in the **Name** property.
21. Use the Database wizard:
  - a. Select the **Delivery** table.
  - b. Select all of the columns except for **DateSent**.
22. Expand the source and destination.
23. Connect **CustomerID** to **Delivery.CustomerID**.
24. Park on **Delivery.RequestID** and enter a **Calculated Value**: **F.RequestNum**.
25. Park on **Delivery.DateEntered** and enter a **Calculated Value** : **Date ()**.  
**Date ()** is an internal Magic xpi function that returns the current date.

You continue the process with the request items. You need to take the items that are authorized for delivery. In this case, the status of the request is TRUE.

26. Drag a Data Mapper step as a child step of the **Add Delivery Header** step. Name the step **Add Delivery Details**.
27. Double-click the **Add Delivery Details** step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

28. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
29. Right-click on the Database source and select **Show Properties**.
30. Enter **Get\_Request\_Details** in the **Name** property.
31. Use the Database wizard:
  - a. Select the **RequestItems** table.
  - b. Select the **ProductCode** and **Quantity** columns.
  - c. Set the WHERE clause to: **[RequestItems].RequestID=<?F.RequestNum?> and [RequestItems].Status=1**  
**Status** is a Logical field but is defined in the database as BIT. It accepts 0 or 1.
32. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
33. Right-click on the Database source and select **Show Properties**.
34. Enter **Add\_To\_Delivery** in the **Name** property.
35. Use the Database wizard:
  - a. Select the **DeliveryItems** table.
  - b. Select all of the columns.
36. Expand the source and destination.
37. Connect **ProductCode** to **DeliveryItems.ProductCode**.
38. Connect **Quantity** to **DeliveryItems.Quantity**.
39. Park on **DeliveryItems.RequestID** and enter a **Calculated Value** with the expression: **F.RequestNum**.

## Deleting the ODS from the System

Once the request has been processed, and in this case delivered, you no longer need to keep all the references to the request. You need to start a process of system cleanup. You can remove traces from the temporary databases so that they do not get cluttered with many records, most of which are no longer valid. One of these storage areas is the ODS. Once the request has been processed, there is no use for the request anymore in the ODS. In the course's example, it will no longer be accessed. Therefore, you can now remove the ODS from the system:

1. Drag a Flow Data utility as a child of **Add Delivery Details**. Name the step **Remove ODS entry**.
2. Double-click the **Remove ODS entry** step, or right-click on it and select **Configuration** from the context menu.
3. Click **Add**.
4. Set the **Action** to **Delete**.
5. Select the **Dynamic** property.
6. In the **Name** property, set the name to: `'Request_' & Trim ( Str ( F.RequestNum , '5' ) )`

You have finished defining the flow. The request has been added to the delivery system and the ODS has been cleared. Now you need to publish the topic for requests that have no problems.



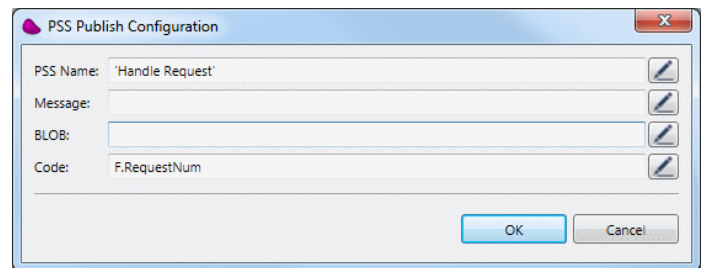
## Publish the 'Handle Request' Topic

The PSS Publish utility publishes events to the PSS system so that the system can invoke flows that subscribe to the published event.

In this section you will publish the **Handle Request** topic when the request is found to be valid.

To publish the topic, do the following:

1. Park on the **Scan for New Requests** flow.
2. Add the **PSS Publish** utility as a child step of the **Check Items** step.
3. Name this step: **Publish the Handle Request**.
4. Double-click the step, or right-click on it and select **Configuration** from the context menu.
5. Zoom from the **PSS Name** entry and enter **Handle Request** in the Expression Editor.
6. In the **Code** parameter, set the following expression:  
**F.RequestNum**.
7. Click **OK** to confirm.
8. Set a condition for the **Publish the Handle Request** step:  
**Trim ( F.CustomerName )<>" AND C.All\_Items\_Exist**



By setting the condition, the **Handle Request** topic will only be published if the customer exists and all items are available.

## Exercise

In the previous lesson, you added a new customer. Remember that the customer may have some outstanding requests. Now that the customer is already in the system, their requests may be attended to.

1. In the **Process Request** flow, remove the request from the local database tables, both the **Requests** and **RequestItems** tables. This is part of the system cleanup.
2. In the **Add Customer** flow, publish the **Handle Request** topic if the customer's request is valid. Remember that the request's items are saved in the **RequestItems** table and a valid item is one where the status is TRUE. To ship this request, all items in the request must be valid.

## Summary

This lesson introduced you to the **Publish and Subscribe** mechanism in Magic xpi. In this lesson, you used the **PSS Publish** utility to publish the **Handle Request** topic.

The **Handle Request** topic is only published if the request is valid.

The **Process Request** flow is subscribed to the **Handle Request** topic. The flow will be invoked when the **Handle Request** topic is published.

# Lesson 16

## Automatic Item Check

In cases where the incoming request was found to be valid, it is processed and removed from the database. Requests that were found to be invalid are kept in the database and ODS, until the fault is corrected manually (adding the customer to the database) or items that did not have sufficient quantity in stock are now fully stocked.

For these requests, you need to provide a follow-up mechanism that checks whether the requests that are logged in the database are still invalid.

The follow-up mechanism will re-examine all of the requests from the database and process those that are valid.

In this lesson, you will learn about various topic including:

- Scheduling a flow
- Flow Enablement

## Scheduler Utility

You have learned about many different ways to invoke a flow:

- The flow starts immediately by setting the **AutoStart** flow property to **Yes**.
- Calling the flow from a different flow using the **Call Flow** destination of the Data Mapper utility.
- An external event occurs, which triggers the flow.
- An internal event occurs using the Publish and Subscribe utility.

There are situations where a flow needs to be invoked at a specific time or time interval.

The **Scheduler** utility enables you to create schedules for flow invocation. During deployment, the Magic xpi Server uses the information in the Scheduler system to invoke flows at the required time periods.

The Scheduler handles timer events, created by the integration project developers, which determine the schedule for flow invocation.

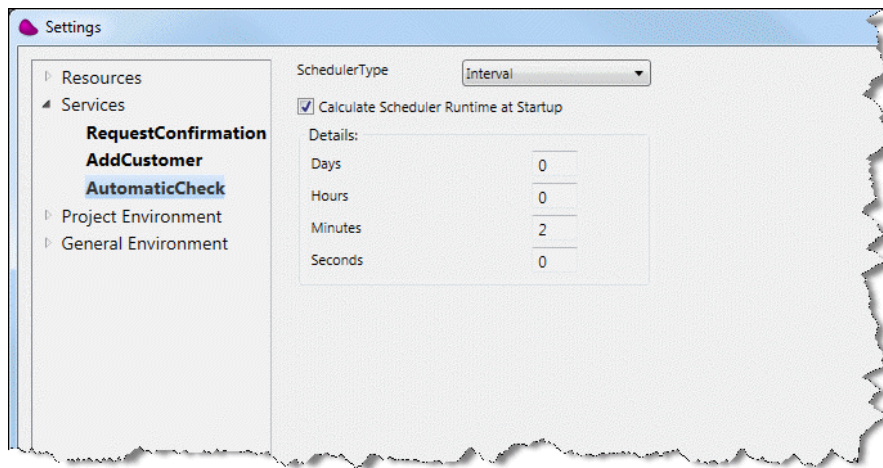
The Scheduler has an entry for every timer event for flows on the particular server. The entry contains the date and time for when the flow should be executed. The Magic xpi Server constantly checks the Scheduler, and, when a scheduler entry's time arrives, the Magic xpi Server invokes the appropriate flow.


When you make changes to your integration project, the Scheduler system is updated with any new timer events.

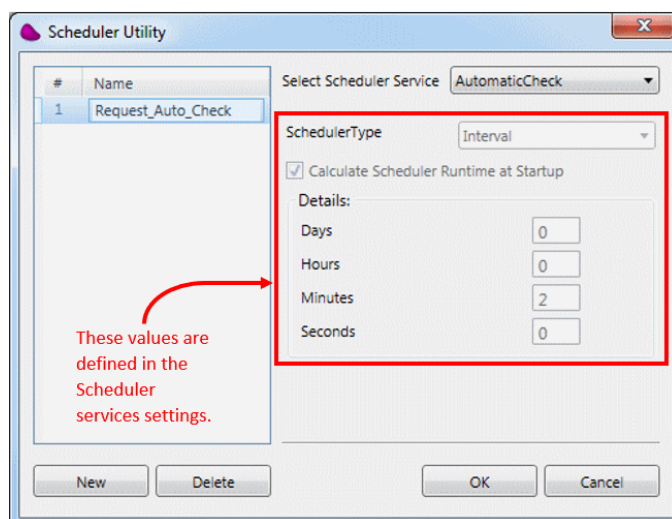
You define schedules in the Flow Editor. To define a scheduler for a flow, you need to first define a Scheduler service in the **Settings** dialog box's **Service** section. You then need to drag a Scheduler utility to the Triggers area, and select the Scheduler service that you just defined.

You will now create a flow that will scan all requests in a timely manner.

1. In the **Settings** dialog box's **Service** section, add a new Scheduler service. Name it **AutomaticCheck**.
2. Set the **Scheduler Type** field to **Interval**, and leave the **Calculate Scheduler Runtime at Startup** check box selected.
3. Set the **Minutes** field to **2**.



4. Click **OK** to close the **Settings** dialog box.
5. Insert a new flow named **Requests Auto Check**.
6. Right-click on the flow name and select **Move Flow Up**. Move it above the **Check Item** flow.
7. Drop a Scheduler utility into the flow's Triggers area. 
8. Double-click the Scheduler utility, or right-click on it and select **Configuration** from the context menu.
9. In the **Scheduler Utility** dialog box, type **Request\_Auto\_Check** in the **Name** field.
10. Set the **Select Scheduler Service** field to **AutomaticCheck**. You will then see the values that you defined in the Scheduler service (above).



In the course scenario, there are three possible reasons that the status of the request may be considered invalid:

- There is not enough stock to meet the required request.
- The price that the customer is prepared to pay is too low.
- The item does not exist in the database. You will not handle this scenario.

11. Drop a Data Mapper utility onto the flow. Name it **Check Request Items**.

12. Create a Database source named **RequestItems**.

13. Using the wizard:

- Select the **RequestItems** table.
- Select the **RequestId**, **ProductCode**, **Quantity** and **RequestPrice** columns.
- Set the WHERE clause to: **[RequestItems].Status=0**. Only lines that are considered invalid will be fetched.

You need to check the reason for each item; therefore, you can call a flow and pass parameters to the flow. Before continuing, you need to create the new flow.

14. Insert a new flow named **Check Reason**.

15. Right click on the flow name and select **Move Flow Up**. Move it below the **Check Item** flow.

16. Add the following flow variables:

- **F.RequestNum** – Numeric of size **5**.
- **F.ItemCode** – Alpha of size **30**.
- **F.RequestQuantity** – Numeric of size **9.2**.
- **F.RequestPrice** – Numeric of size **9.2**.

17. Add a Data Mapper step as the first step in the flow and name the step **Check Item Reason**.

18. Double-click the **Check Item Reason** step, or right-click on it and select **Configuration** from the context menu.

19. Create a Database source named **FetchItems**.

20. Using the wizard:

- Select the **Products** table.
- Select the **CatalogPrice** and **StockQuantity** columns.
- Set the WHERE clause to: **[Products].ProductCode='<?F.ItemCode?>'**

Now you are going to update the **RequestItems** table according to special conditions:

21. Create a Database destination named **UpdateItems**.
22. Using the wizard:
  - Set the **DB Operation** to **Update**.
  - Select the **RequestItems** table.
  - Select the **Quantity** and **Status** columns.
  - Set the WHERE clause to:  
**RequestID=<?F.RequestNum?> and ProductCode='<?F.ItemCode?>'**  
 You may need to manually add the apostrophes before and after **<?F.ItemCode?>**, because the product code is a string field and this is the requirement of the SQL syntax.
23. Expand the source and destination.
24. Connect **CatalogPrice** to **Quantity** and to **Status**.
25. Connect **StockQuantity** to **Quantity** and to **Status**.
26. Park on **Quantity** and enter a **Calculated Value** with the expression:  
**IF (Src.S1/Record/CatalogPrice < F.RequestPrice AND Src.S1/Record/StockQuantity < F.RequestQuantity, Src.S1/Record/StockQuantity, F.RequestQuantity)**  
 This expression means that if the price is more than the catalog price and the requested quantity is more than the stock, then the request will be updated with the stock amount. This means, of course, that all the stock will be used to deliver this request.
27. Park on **Status** and enter a **Calculated Value** with the expression:  
**IF (Src.S1/Record/CatalogPrice < F.RequestPrice AND Src.S1/Record/StockQuantity < F.RequestQuantity, 'TRUE'LOG, 'FALSE'LOG)**  
 In the previous expression, you set the value of the requested amount to a valid amount. Therefore, this entry can now be seen as valid.
28. Return to the **Requests Auto Check** flow.
29. Double-click the **Check Request Items** step, or right-click on it and select **Configuration** from the context menu.
30. Create a new **Call Flow** destination named **Call\_Request\_Check**.
31. In the **Flow Name** property, select the **Check Reason** flow.

In the **Data Mapper** window:

32. Connect the **RequestId** source node to **F.RequestNum**.
33. Connect the **ProductCode** source node to **F.ItemCode**.
34. Connect the **Quantity** source node to **F.RequestQuantity**.
35. Connect the **RequestPrice** source node to **F.RequestPrice**.

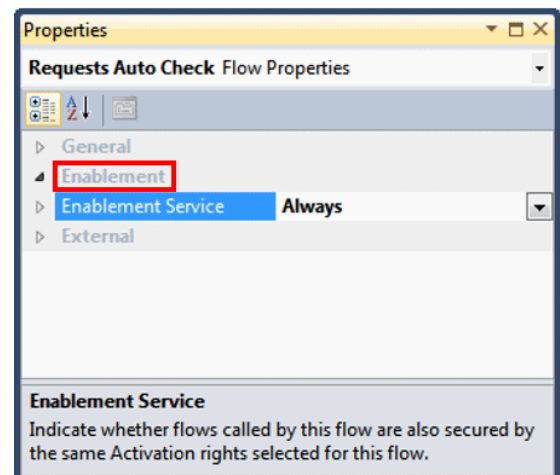
## Flow Enablement

Another flow property that you need to be aware of is the flow enablement. The **Enablement** section of a flow's **Properties** pane defines when the flow is available to be activated.

When a flow needs to be activated, the Magic xpi Server checks whether the flow is currently enabled.

You define when you want the flow to be activated. Select one of the following from option:

- **Always** – The flow is always enabled.
- **Weekly** – You can define on which days of the week and at which times this flow will be enabled.
- **Monthly** – You can define in which months and which days in the month this flow will be activated. For example, you can define that the flow will only run on the 7th of January, April, July and October.



In much the same way as you previously defined a Scheduler service, the **Weekly** and **Monthly** options (above) are configured in the **Settings** dialog box's **Service** section. You then 'import' these settings into the flow's properties.

First, you create a Flow Enablement service and select either **Weekly** or **Monthly** in the **Activation Type** field. You then select the relevant check boxes for the days or the months that you want to activate your flow. Then, in the **Enablement Service** field in the flow's **Properties** pane, you select the required Flow Enablement service's name. Your flow will then be enabled according to the settings that you defined in the Flow Enablement service.

If the flow is not enabled and a request to invoke it arrives via a trigger mechanism, such as an HTTP request or a Scheduler action, the flow will not be invoked and the activation request will be ignored.



## Exercise

In this lesson, you corrected invalid requests. However, they remain in the system and are not sent out. Remember that it is the **Handle Request** topic handles the valid requests.

- Define a scheduled flow that is activated at midnight and:
  - a. Scans the open requests.
  - b. Publishes the **Handle Request** topic if an open request has at least one valid line in the request. Remember that the status is saved in the **RequestItems** table.

Remember to only publish the request if the customer exists.

## Summary

This lesson provided you with additional ways to control when a flow is invoked.

You learned about:

- The Scheduler utility
- Flow enablement

During the lesson, you fixed invalid requests. If it was found that there was not enough in stock to supply the request, you modified the request. You then set a new scheduler that would run once a day and handle any requests that were still in the system.



Magic<sup>®</sup>  
University

# Lesson 17

## More About Magic xpi

In previous lessons, you learned how to transfer and manipulate data using the Data Mapper. However, there are some rules that govern what can be mapped and what cannot be mapped. This is very evident when trying to map XML data to other XML data. Knowledge of the rules that the Data Mapper uses will enable you to avoid future problems.

This lesson will introduce you to the rules used by the Data Mapper. You will also learn some additional mapping techniques.

This lesson covers various topics, including:

- Simple element mapping
- Complex element mapping
- Email XML configuration
- User Defined Storage (UDS)

## More about the Data Mapper

The Magic xpi Data Mapper deals with mapping issues on two levels:

- Single element – The following elements are considered to be single elements:
  - An element with the **Max Occurrences** node property set to 1. You can see the **Max Occurrences** value of a compound node when you select **Show Properties** and go to the **Properties** pane's **Additional XML Properties** section.
  - An element with a filter.
  - A duplicate node.
  - A complex element in the destination with no mapping.
- Complex element – The following elements are considered to be complex elements:
  - An element with child elements.
  - An element with the **Max Occurrences** node property greater than 1.

### Single Elements

You can map the following elements in the **Data Mapper** window:

- One source complex element to a destination complex element.  
Mapping a complex element tells the Data Mapper to execute the destination complex element the same number of times as the source complex element.
- Single source multiple simple elements to one destination simple element (if the destination type is XML or Template).
- Source complex elements can be mapped to a flow variable complex element.
- Most sibling elements can be mapped. You will learn more about this over the following pages.

Simple data mapping is done automatically by the Data Mapper. You do not need to create an expression.

For the examples on the following pages, you have a sample project.

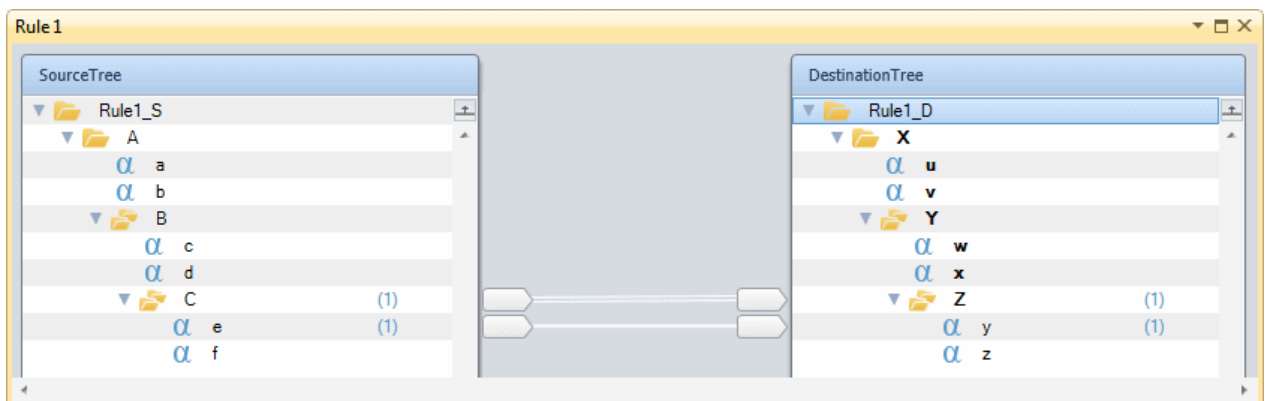
1. Copy the folder **Mapper\_Sample** from the **Course Files** folder to the **My Documents\Magic\projects** folder.
2. Open the **Mapper\_Sample** project.

## Automatic Mapping

Automatic complex element mapping ensures that no data is lost while mapping.

Whenever simple elements are mapped without a parent complex element mapping, Magic xpi automatically connects the first parent complex element of the source and destination elements where the **Max occurrences** node property is greater than **1**.

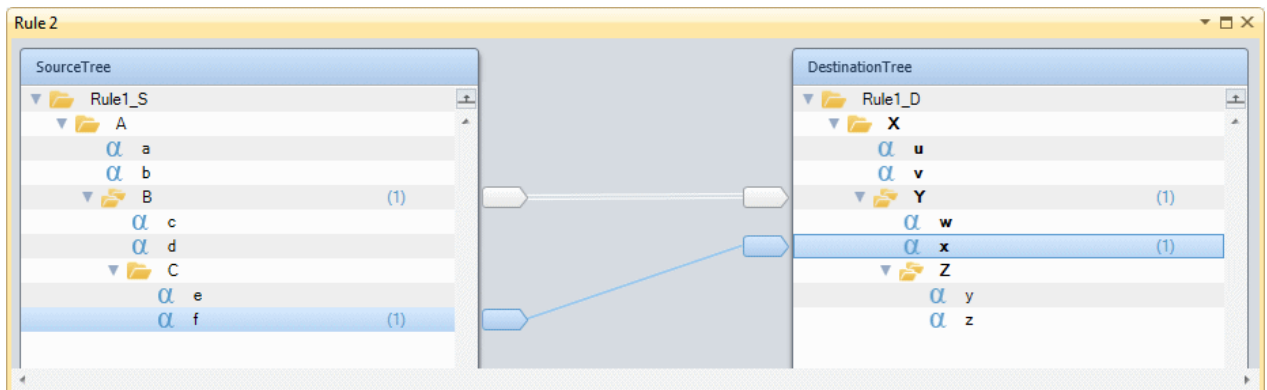
1. Park on the **Automatic mapping** flow.
2. Park on the first Data Mapper step, and double-click or right-click on it and select **Configuration**.
3. Connect **e** to **y**.



You see in the image above that a parent has been connected automatically. You used this throughout the course.

When the **Max Occurrences** node property equals 1, the Data Mapper will automatically map the parent of the step that has a **Max Occurrences** value greater than 1.

1. Park on the **Automatic mapping** flow.
2. Park on the second Data Mapper step, and double-click or right-click on it and select **Configuration**.
3. Connect **f** to **x**.



You will see in the image above that **B** was automatically connected to **Y**. This is because **C** has a Max occurrences value of 1.

## Complex Element

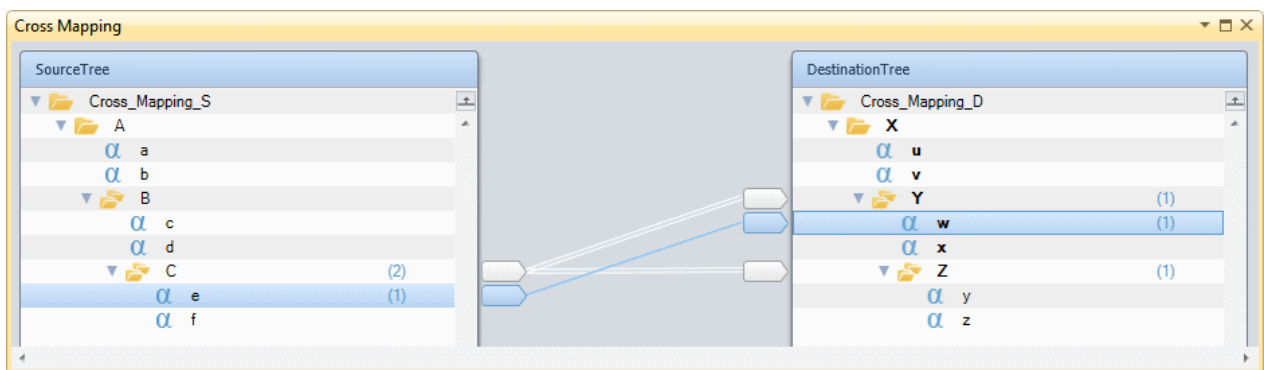
Complex data mapping is carried out by:

- Using the destination data expression.  
The expression can include flow variables and functions and simple elements from the source XML connected to the destination element.
- Using the **Expression Editor** to call external code, such as Java, uniPaaS, or DLL code.  
This option is used for complex data mapping that needs an external function call.

## Cross Mapping

Cross mapping occurs when you already have complex elements mapped and need to map source elements across the complex element mapping line to a parent destination element.

1. Park on the **Cross mapping** flow.
2. Park on the Data Mapper step, and double-click or right-click on it and select **Configuration**.
3. You can see that the complex element **C** is connected to **Z**.
4. Connect **e** to **w**.



You will see that **e** crosses the previous connection and that **C** was automatically mapped to **Y**.

## Sibling Mapping

Now you will try to map siblings:

1. Park on the **Source Sibling Compound** flow.
2. Park on the Data Mapper step, and double-click or right-click on it and select **Configuration**.
3. You can see that **c** is connected to **w** and therefore **B** was automatically mapped to **Y**.
4. If you try to connect **e** to **x**, you will get an error.  
The reason for this is that the Data Mapper has no way of determining the number of iterations to execute on the sibling.

To overcome this, you need to ensure that the **C** complex element has a **Max Occurrences** of 1. This can be implemented by changing the schema or adding a filter.

1. Park on **C** and access the properties. Enter a condition for the **Single Instance Filter**, such as **'TRUE'LOG**.
2. Connect **e** to **x**. This time, the action should be successful.  
The condition you entered is an example. The actual expression to use would depend on your own data.

## Data Management Best Practices

The Data Mapper is a very powerful tool, but has an overhead of time and memory. Consider using the Data Mapper only when you need mapping capabilities. For instance:

- Use the Flow Data utility instead of the Data Mapper to update variables.
- Use the Data Mapper's **Call Flow** option only when you need to process many records. Use the **Invoke Flow** utility if you need to process one record only.
- When you need to store data temporarily, it is more efficient to use a file saved in a variable, such as a BLOB, than to use the ODS. The ODS is a table stored in a database, and thus consumes more memory than a memory file. However, the ODS is persistent and can be recovered from failure. You will learn about the memory file in this lesson.



## Runtime Considerations

Now that you have a good understanding of how the Data Mapper works, there are some issues that are important during execution.

- When you have a condition and an update expression on a compound node, the update expression will be calculated first, regardless of the condition expression. This means that variables contained in the update expression will always be updated.
- The Data Mapper has a specific order in which it executes the requested operations. These should be taken into account when you are designing the step:

### Execution Order

The execution order is based on the order of the destination items, starting from the top.

If you reorder the destinations, the execution order may also change.

The following are executed at the same time:

- Database
- Flat File
- Variable
- ODS
- Call Flow
- UDS

The following are executed individually:

- XML
- JSON
- Template

## Email XML Configuration

You learned about the Direct Access Method of configuring a step, which is used in most cases. However, there are instances in which you need to configure the component using its XML configuration method. Some external components only have an XML interface.

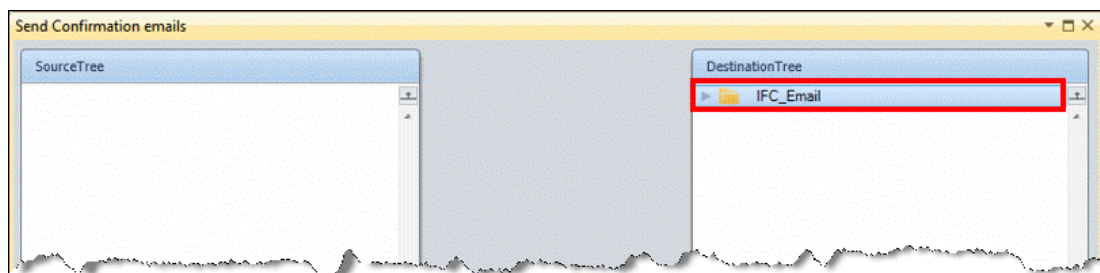
Assume that when a request is sent to the delivery system, you need to update a defined list of people that you have to update by email, such as the warehouse manager, the sales manager, and the accounts manager. You can use a single email step to send the same email to a list of names defined in a text file.

To map the information directly to the Email component, you will use the Email component's XML interface.

1. Open the **Magic\_xpi\_course** project.
2. Park on the **Process Request** flow.
3. Add an Email component as a child of the **Add Delivery Details** step. Name the step **Send Confirmation emails**.
4. Set the **Processing Mode** to **Parallel**.
5. Set the **Interface** to **XML**.
6. Double-click the Email component, or right-click on it and select **Configuration**.

The **Data Mapper** window opens.

In the **Destination Tree**, there is an entry called **IFC\_Email**. This is automatically added as a result of the XML interface.



## Using a Flat File

Before continuing with the Email XML configuration, you need to define the Data Mapper source.

You are going to map comma-delimited data from a flat file to an XML file. The flat file in question has a defined structure that you need to understand to extract information.


In the **Data Mapper** window:

7. From the Toolbox's **Mapper Schemas** section, drag a Flat File source into the **Source Tree** area. Name the source **FetchMailList**.

The comma-delimited file name is: **emails.txt**.

The file is located in **the %currentprojectdir%\course\_data\Files** directory.

You can take a look at the file in order to understand the file structure and how it should be set in the Flat File source's **Properties** pane. In this case, it is a very simple structure.

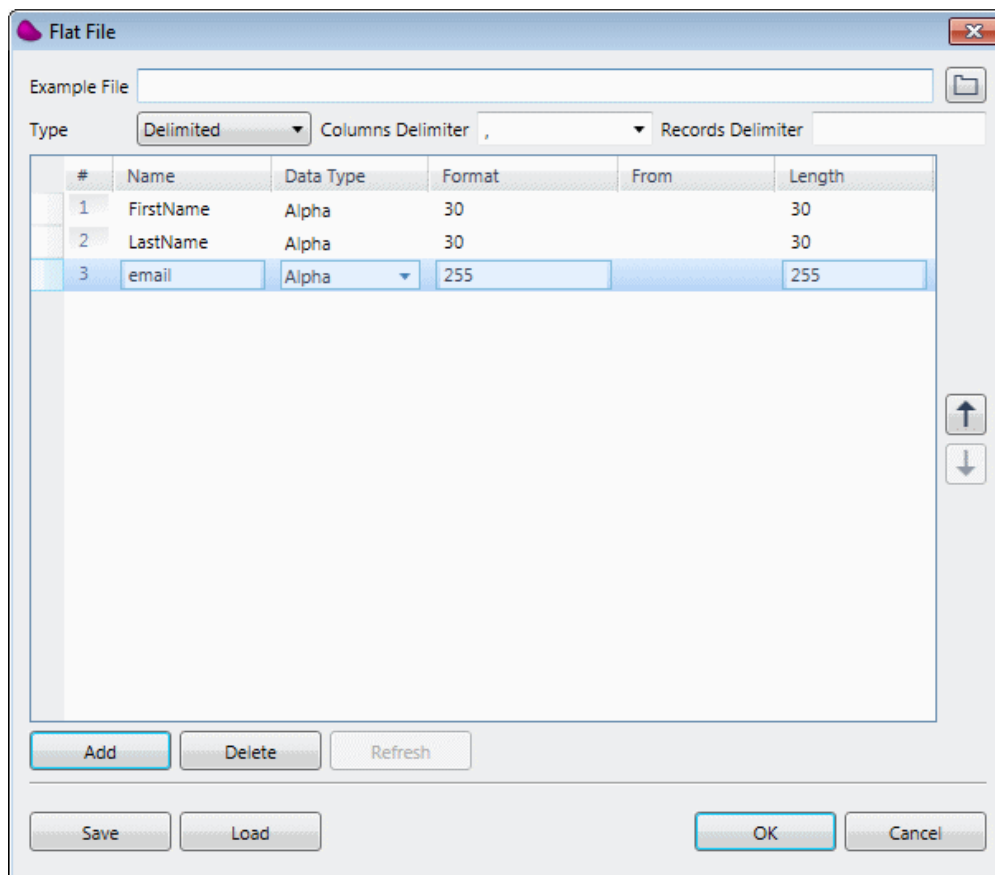
8. In the Flat File source's **Properties** pane:
  - Set the **Source Type** property to **File**.
  - In the **File Path** property, use the Expression Editor to select the **currentprojectdir** environment variable, and define the following expression:  
**EnvVal ('currentprojectdir')&'course\_data\Files\emails.txt'**
  - Set the **Include Delimiter** property to **Yes**.
  - In the **Lines** property, click .

The **Flat File** dialog box opens.

You can define whether each field in the text file is separated from the next by a delimiter or in a fixed position. The options will change according to the option you select in the dropdown list. In the **Flat File** dialog box:

9. Set the separator option to **Delimited**. This is the default.
10. Set the **Delimiter** to a comma. This is also the default.
11. Define the following:

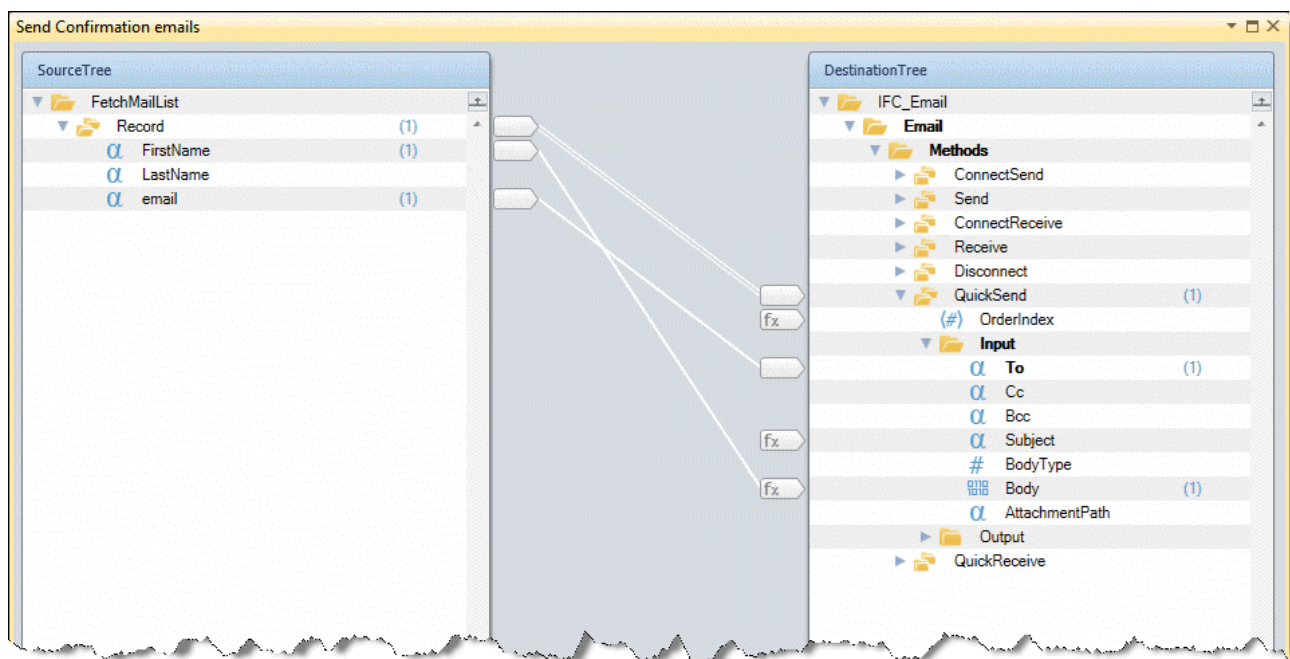
Name	Data Type	Format	From	Length
FirstName	Alpha	30		30
LastName	Alpha	30		30
email	Alpha	255		255



In the **Data Mapper** window:

12. In the **Destination Tree**, open the **QuickSend** node.
13. Connect the following nodes and set the following destination expressions:

Source	Destination	Expression
	OrderIndex	1
email	To	
	Subject	'Please follow up on the request'
FirstName	Body	'Good day '&Src.S1/Record/FirstName &','& ASCIIChr (10)&'Please follow up on Request '&Str ( F.RequestNum,'5')



## User Defined Storage (UDS)

While working with the ODS, you noticed some limitations:

- The structure is predefined and cannot be changed. For example, if you wanted to save two time fields in a record, you were unable to.
- ODS is kept in a table with all the overhead of a regular database table.

The User Defined Storage, or UDS, is very similar to ODS, but it addresses the above "limitations" by:

- Enabling you to define the structure.
- Creating the data as a user-defined table in memory in a BLOB.

### The UDS Repository

In the UDS Repository, you define the actual structure of the table. You define it according to your needs. This is very similar to defining a database table. You define the list of fields, and you define which of the fields will be the index.

Assume that when the **Scan DB for valid requests** flow runs, you need to create a simple audit trail of what was scanned and send the file to the system administrator.

The fields that need to be in the audit trail include:

- Time stamp
- Customer ID
- Request ID
- Request value

This structure is new and needs to be defined.

To access the UDS Repository:

1. In the **Repositories** section of the Solution Explorer, double-click on **UDS** to open the UDS Repository. You can also press **SHIFT+F10**.

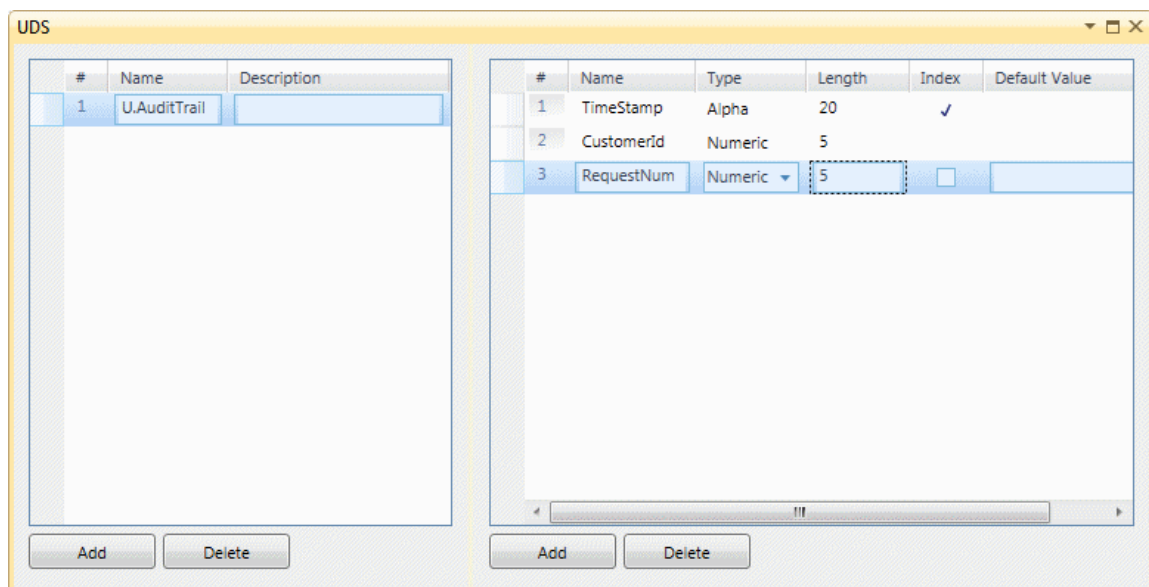
The UDS Repository opens. As with other repositories there are two panes, the left and the right. In the left pane you provide a name for the UDS, and in the right pane you define the variables.

2. Click **Add** on the left pane, and set the name to **AuditTrail**. You cannot use spaces here.
3. On the right pane, click **Add**.
4. Set the **Name** property to **TimeStamp** with a type of **Alpha** and a size of **20**.
5. Select the **Index** check box.

You have now defined the first column in the structure and have defined that this will be the index. You will now define two more columns:

6. On the right pane, click **Add**. Set the **Name** property to **CustomerId**, a **Numeric** field of size **5**.
7. Click **Add** again. Set the **Name** property to **RequestNum**, a **Numeric** field of size **5**.

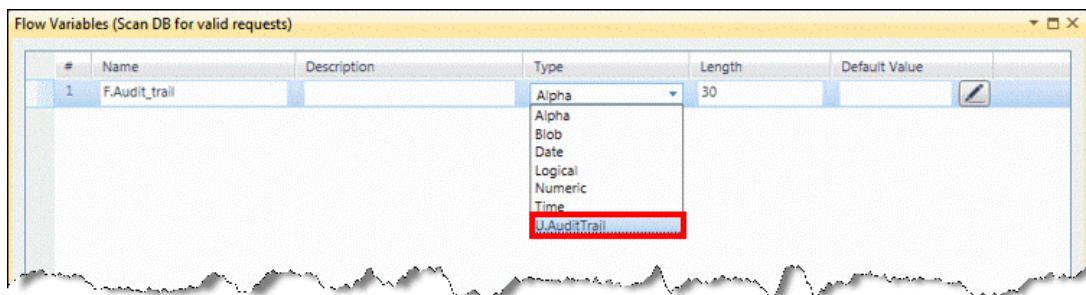
You have now defined a UDS structure. You have three columns, one of them being the structure's index.



The UDS structure forms a model for your use. It is simply the definition of a structure. You therefore need to define a variable that will be based on this model.

When you define a new flow variable, you can define that this variable is of the same type as the UDS that you previously created.

1. Under the **Scan DB for valid requests** flow, double-click **Flow Variables**.
2. Click **Add** to add a flow variable and name it **Audit\_trail**.
3. Park on the **Type** column and open the dropdown list. You will see that the UDS model that you defined is one of the structure's to be selected. Select **U.AuditTrail**.



4. Double-click the **Scan requests** step, or right-click on it and select **Configuration** from the context menu.
5. In the **Source Tree**, right-click on **ScanRequests** and select **Show Properties**.
6. In the Wizard process, add the **CustomerId** column.
7. From the Toolbox's **Mapper Schemas** section, drag a UDS destination into the **Destination Tree** area.
8. Right-click on the UDS destination and select **Show Properties**.
9. Enter **Create\_audit\_trail** in the **Name** property.
10. In the **Variable** property, select **F.Audit\_Trail**.
11. Expand the source and destinations.



You can now see the three columns of the UDS structure:

12. Connect **RequestId** to **RequestNum**.
13. Connect **CustomerId** to **CustomerId**.
14. Park on the **TimeStamp** node and enter a **Calculated Value**:  
`DStr (Date (),'DD/MM/YYYY')&' '&TStr (Time (),'HH:MM:SS')`

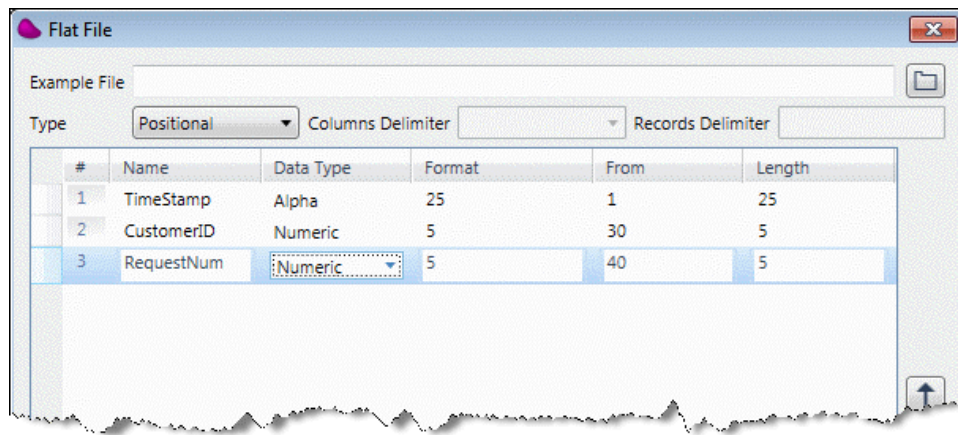
Now that you have written to the UDS, you can retrieve the information. You can do this with the Data Mapper.

15. Add a Data Mapper step as the child step of **Scan requests**. Name the step **Create Trail**.
16. Double-click the **Create Trail** step, or right-click on it and select **Configuration** from the context menu.
17. From the Toolbox's **Mapper Schemas** section, drag a UDS source into the **Source Tree** area.
18. Right-click on the UDS source and select **Show Properties**.
19. Enter **FetchUDS** in the **Name** property.
20. In the **Variable** property, select **F.Audit\_Trail**.

You can now map this to a text file.

21. From the Toolbox's **Mapper Schemas** section, drag a Flat File destination into the **Destination Tree** area.
22. Right-click on the Flat File destination and select **Show Properties**.
23. Enter **CreateFile** in the **Name** property.
24. Set the **Destination Type** to **File**, and set the destination file to:  
`EnvVal ('currentprojectdir')&'course_data\out\audit_trail.txt'`

25. Set the properties so that they look like the image below:



26. Expand the source and destination, and connect the nodes to one another.

## Exercise – Mapping a Flat File to an Order

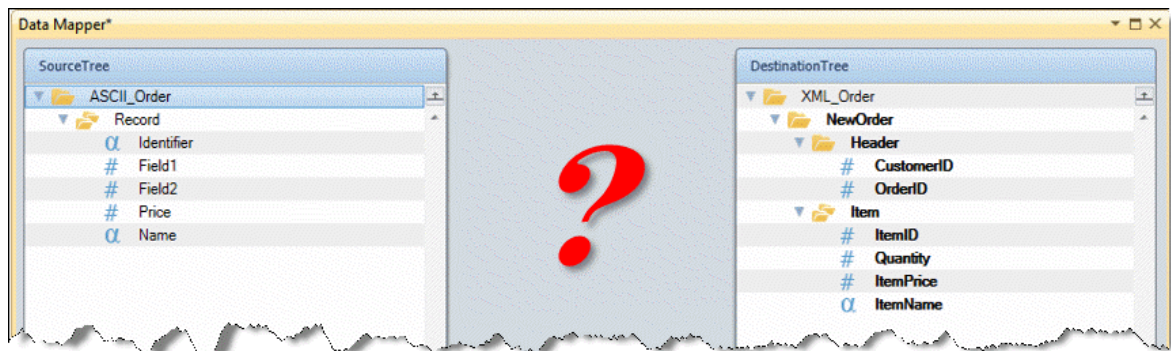
In this lesson, you discussed the XML interface, and you used an advanced Data Mapper technique in which you used a flat file.

In this exercise, you are going to map comma-delimited data from a flat file to an XML file.

The flat file contains comma-delimited data. The first column determines the line type (H = Header, L = Line).

You need to map all **header type** data to the **Header** compound in the XML, and all **line type** data to the **Item** compound in the XML.

The image below shows you the structure of the example files. It is presented here to give you a hint about how to map these files.



All files are located in the `%currentprojectdir%\course_data\Files` directory.

For the purpose of this example, add a new flow called **Create XML Order from ASCII**. Place the XML file in the **OUT** directory, and name it **Order.xml**.

## Summary

In this lesson, you learned about additional Magic xpi features.

- You learned more about the Magic xpi Data Mapper, including some of the issues to take into consideration on single and complex elements.
- You learned about flat file mapping.
- You also learned about UDS storage and how to use it.

When working with UDS, remember:

- UDS data is stored in the memory, and is therefore available to the current running flow only.
- You can copy the UDS data to a BLOB to enable you to use it at a later stage.
- The UDS Repository can be used only with the Flow Data utility and the Data Mapper utility.
- Once you define UDS models in the UDS Repository, they will be added to the list of the flow variable types (in the Flow Variable repository's **Type** column).

# Lesson 18

## From Development to Deployment

Magic xpi project development is carried out in the Magic xpi Studio. You build your project with the Magic xpi Studio, which runs on Windows. Magic xpi Integration Platform includes the Studio, the Magic Monitor, a deployment server known as the Magic xpi Server, and the In-Memory Data Grid (IMDG). This enables you to build your project and test it in a controlled environment.

When the development environment and the deployment environments both run on the Windows operating system, there should be very few problems with running the flows in a project. However, remember that the deployment environment may not be a Windows operating system.

As you are getting ready to move your project from the development environment to the test or production environment, there are several steps you need to perform.

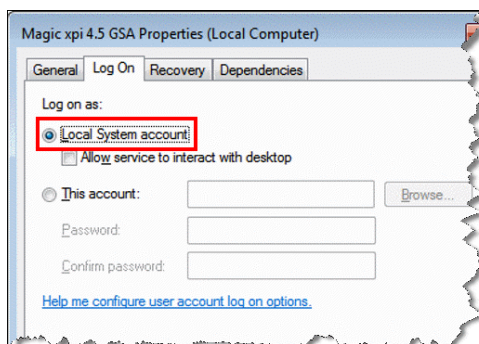
This lesson covers various topics, including:

- Deployment issues
- Database issues

## Recommendations

To be sure that your project will run when you deploy it on the server, you must consider the following issues:

- It is recommended to use the same database type for the Magic xpi internal and application databases that you are going to use in deployment environments. This means that if your development internal database was Microsoft SQL Server, then you should use the same server type in deployment.
- It is recommended that you disable or clear all debugging code from the project before deploying. For example, disable or clear the flows, messages, and programs used for debugging purposes.
- Use environment variables when you provide names or addresses to servers, messaging queues, and aliases.
- It is good practice to use environment variables in resources and services. Magic xpi achieves portability by translating the environment variables at runtime, according to the values entered in the **Settings** dialog box's **Environment** section.
- It is important to remember that the Magic xpi engines run under the user that is defined for the GSA service, and not by the logged in user. By default, the user that is defined for the service is the **Local System account** as shown in the image below. For running Magic xpi on a single machine, this is usually fine. However, on a clustered environment, the service should run as a user who has privileges to access network resources.



## Deployment Issues

When you are ready to deploy your projects, you need to prepare the server environment.

It is highly recommended that you rebuild your project whenever you make any changes to it, and also prior to deployment, by selecting the **Rebuild Solution** option from the **Build** menu.

If you saved any environment variables under the **General Environment** section of the **Settings** dialog box, make sure that you copy the relevant variables to the **Magic.ini** file of any new deployment environment.

If you deploy the project in a different deployment environment, you need to modify the **start.xml** file to match the new project location and host name. Alternatively, you can delete the file completely and then rebuild the project in the new environment.

It is good practice to test your project in the deployment environment to ensure that you have correctly configured and deployed your project.

For more information about deploying projects in a clustered environment, see the **Magic xpi 4.x - Advanced Deployment Guide.pdf** file included with the Magic xpi installation.

## Summary

In this lesson, you have learned about:

- Recommendations
- Deployment issues

Remember that when you deploy, the only component of Magic xpi Integration Platform that you will need is the Magic xpi Server.

If you plan to deploy your project on a non-Windows platform, you should consider the differences in the operating systems when you develop your project. Read the *Magic xpi Help* for assistance on these systems.



## Course Data

The sample data needed to do these exercises are in the **course\_data** directory, which contains the following folders:

- **schemas** – Holds the **request.xsd** schema.
- **out** – This contains three sample XML files that adhere to the **request.xsd** schema.
- **in** – This is empty. To start running the project you need to copy one of the request XML files into this folder.
- **templates** – This contains templates that you will need during the course.
- **Files** – This directory contains some files needed for the more advanced lessons.
- **DB** – This contains files needed to create the MSSQL database.

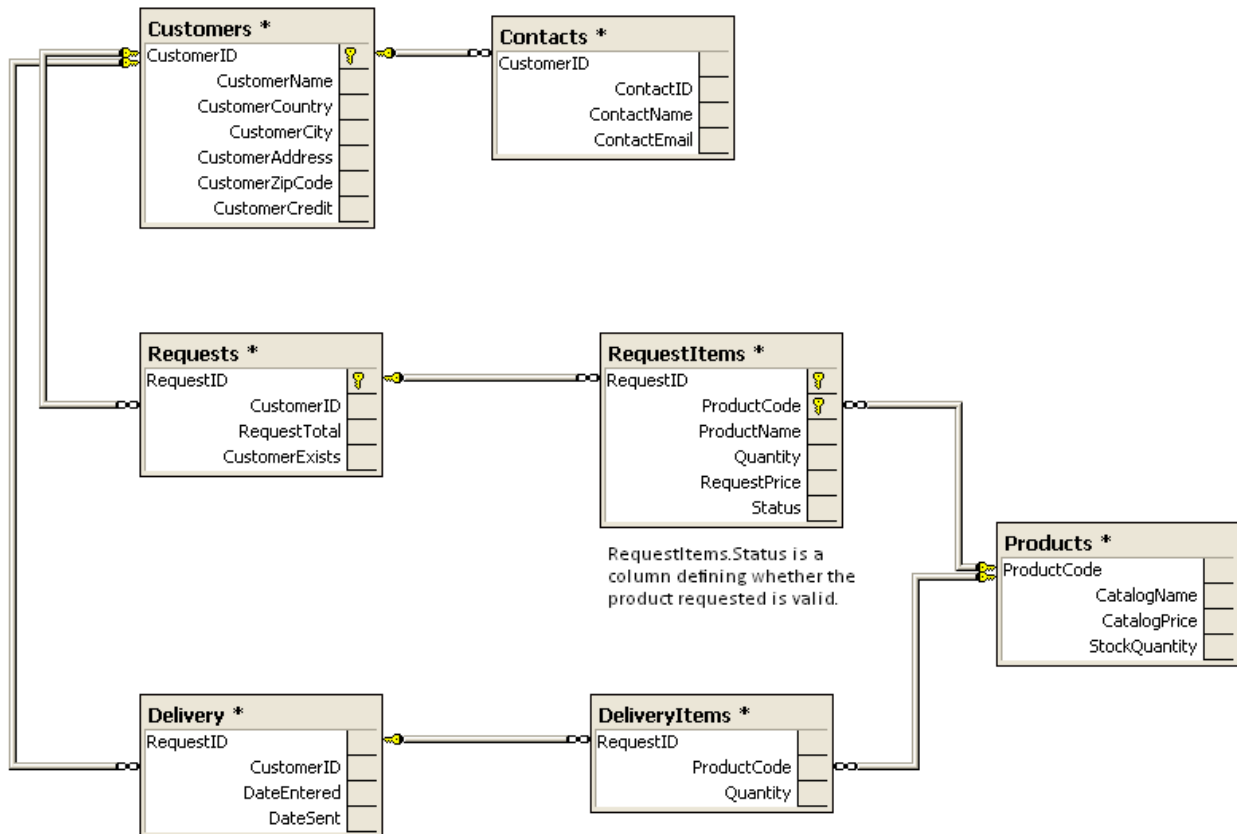
## Sample XML Requests

Three XML requests are provided as examples:

Request001.xml	The customer exists in the database. <ul style="list-style-type: none"><li>• Line 1 – The price is too low.</li><li>• Line 2 – The line is good.</li><li>• Line 3 – There is not enough in stock.</li></ul>
Request002.xml	The customer does not exist.
Request003.xml	The request is a valid request.

## Entity Relations Diagram (ERD)

The diagram below shows the relationship between the MSSQL tables that you will be using in this course:





# Solutions

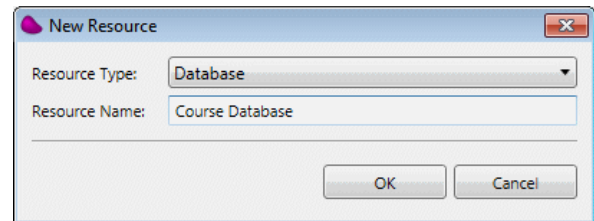


Magic®  
University

## Solution Lesson 4 – Resources

You need to add a resource that will connect to the course database. To do this:

1. From the **Project** menu, select **Settings**.
2. In the left hand pane of the **Settings** dialog box, park on **Resources**.
3. Click **Add**.
4. In the **Resource Type** field, using the dropdown list, select the **Database**.
5. In the **Resource Name** field, type **Course Database**.



In the right hand pane of the **Settings** dialog box, you will define the properties for MSSQL:

6. Set the **DBMS** property to **Microsoft SQL Server**.

The property list changes according to the MSSQL settings:

7. Set the **Database name** property to **Magic\_xpi\_course**. This is the course database.
8. Set the **Server**, **User**, and **Password** properties to the MSSQL login options of your system.
9. Click **Validate** to check the connection.

#	Name	Type	Format	Value
1	<b>DBMS</b>	Alpha	30	Microsoft SQL Server
2	<b>Database name</b>	Alpha	30	Magic_xpi_course
3	<b>Server</b>	Alpha	30	MUI-XP\SQLEXPRESS
4	<b>User</b>	Alpha	30	sa
5	<b>Password</b>	Alpha	30	**
6	<b>Starting Owner</b>	Alpha	30	

## Solution Lesson 5 – Scan for New Requests

You are required to add a step that sends an email to notify the administrator that a new request has arrived. In the course, you will be using a single valid email, **postmaster@magic.xpi.course.com**. You will use this for all the tests.

1. Park on the **Toolbox** pane.
2. Click and drag the Email component and drop it on the **Scan Directory** step.

Right-click on the Email component and select **Properties**.

3. In the **Properties** pane's **Name** field, enter **Send Email to Sales**.
4. In the **Properties** pane's **Settings** section, make sure that the **Course email** resource has been automatically selected. It is automatically selected because it is the only email resource that you have defined.
5. Double-click the **Send Email to Sales** step, or right-click on it and select **Configuration** from the context menu.

The **Direct Access Method** dialog box opens:

6. In the left pane, click **Add**.
7. Select **Quick Send** from the dropdown list.

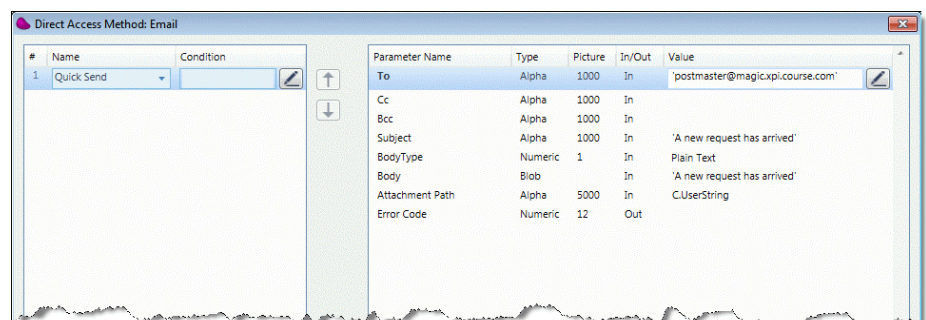
In the right pane, you define the **Method Details**.

8. In the **To** property, enter **postmaster@magic.xpi.course.com**.
9. In the **Subject** property, type **'A new request has arrived'**.

The apostrophes are important here as this is a string value.

10. In the **Body** property, type **'A new request has arrived'**.

The apostrophes are important here, since this is a string value.



In the **Scan Directory** step, you defined that the file name and path would be returned in a variable named **C.UserString**. Therefore:

In the **Attachment Path** property, type **C.UserString**. You will change this in a later lesson.

## Solution Lesson 6 – Flow Orchestration

### Adding Variables



You are required to add variables that are to be used in place of the Magic xpi predefined variables.

1. Under the **Scan for New Requests** entry in the Solution Explorer, double-click **Flow Variables**. The Flow Variables repository opens. Click **Add** to add a variable.
2. Set the **Name** property to **RequestXML** and tab to the **Type** property. The name will change to **F.RequestXML**.
3. Open the **Type** dropdown list and select **BLOB**.
4. Add the following flow variables:

Name	Type	Length
RequestFileName	Alpha	255
CustomerEmail	Alpha	100
CustomerName	Alpha	100
CustomerId	Numeric	9

5. Under the **Repositories** entry in the Solution Explorer, double-click **Global Variables**. The Global Variables repository opens.
6. Click **Add** to add a variable.
7. Set the **Name** property to **EmailTo** and tab to the **Type** property. The name will change to **G.EmailTo**.
8. Open the **Type** dropdown list and select **Alpha**.
9. In the **Length** field, enter **100**.
10. In the **Default Value** property, enter **postmaster@magic.xpi.course.com**.

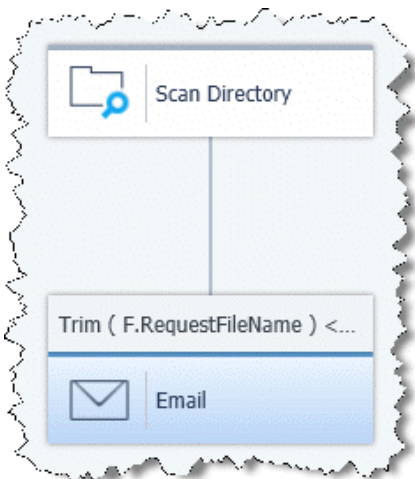
Now you have to use these variables:

11. Double-click the **Scan Directory** step, or right-click on it and select **Configuration** from the context menu.
12. Make sure that **LAN to LAN** is selected in the left pane.
13. On the right pane, park on the **Return File To** property. Click the Browse button . Select the **F.RequestXML** variable that you defined.
14. Park on the **Return Destination Filename To** property. Click the Browse button . Select the **F.RequestFileName** variable that you defined.
15. Double-click the **Send Email to Sales** step, or right-click on it and select **Configuration** from the context menu.
16. Make sure that **Quick Send** is selected in the left pane.
17. On the right pane, park on the **To** property and enter **G.EmailTo** as the value.

The last stage is to define a condition:

18. Park on the **Send Email to Sales** step.
19. Open the context menu and select **Condition**.
20. Enter the following expression: **Trim ( F.RequestFileName ) <>"**.
21. Click **Verify** to make sure that the expression you defined is correct.
22. Click **OK**.

Your flow will look something like this:





## Solution Lesson 7 – Checking Customer Existence

### Checking Customer Existence

You are required to check the existence of the contact.

The first stage is to define the contact ID variable.

1. In the Solution Explorer, under the **Scan for New Requests** entry, double-click on **Flow Variables**.
2. In the Flow Variables repository, click **Add** to add a variable.
3. Set the **Name** property to **ContactId** and tab to the **Type** property. The name will change to **F.ContactId**.
4. Open the **Type** dropdown list and select **Numeric**.
5. Set the Length to **9**.


Now you need to extract the information from the XML:

6. Double-click the **Extract details from request** step, or right-click on it and select **Configuration**.
7. In the **Destination Tree** area, right-click on the **Variables** entry and select **Show Properties**.
8. In the **Properties** pane, click the **Variable** field's  button and select the **F.ContactId** variable that you defined. Note that you need to close the **Extract details from request** step's window and reopen it again for this variable to be added successfully.
9. In the **Source Tree**, open the **Request** node, then the **CustomerDetail** node, and then the **ContactDetail** node.
10. Connect **ContactID** to **F.ContactId**.

To check the existence of the contact:


11. Drag a Data Mapper utility as a child step of the **Check if the customer exists** step. Name this step **Check the Contact**.
12. Double-click the Data Mapper utility, or right-click on it and select **Configuration** from the context menu to open the **Data Mapper** window.
13. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Data Mapper** window's **Source Tree** area.
14. Right-click on the XML source and select **Show Properties**.
15. Enter **CheckContact** in the **Name** property.

The **Database Definition** property points to the **Course Database** that you defined.

16. In the **Properties** pane's **Wizard** field, click the  button. The **Select Tables** dialog box opens.
17. Park on the **Contacts** entry in the **Available Tables** pane.
18. Click **Add**. The **Contacts** entry is removed from the **Available Tables** and added to the **Selected Tables** pane.
19. Click **Next**. The **Select Columns** dialog box opens.
20. Park on the **ContactName** entry in the **Available Columns** pane.
21. Click **Add**. The **ContactName** entry is removed from the **Available Columns** and added to the **Selected columns** pane.
22. Click **Next**. The **Where Clause** dialog box opens.
23. Park on **[Contacts]CustomerID** in the **Available Columns** pane and double-click. **[Contacts]CustomerID** is added to the **Where Clause Text** pane.
24. Park on the **Where Clause Text** pane and type **=** after **[Contacts]CustomerID**. The text should now read: **[Contacts]CustomerID =**
25. Park on **F.CustomerId** in the **Variables** pane and double-click. The text should now read: **[Contacts]CustomerID = <?F.CustomerId?>**
26. You now need to fetch the contact part. Park on the **Where Clause Text** pane and type **AND** after **[Contacts]CustomerID = <?F.CustomerId?>**.
27. Complete the WHERE clause so that the WHERE clause will be:  
**[Contacts]CustomerID = <?F.CustomerId?> AND [Contacts].ContactID=<?F.ContactId?>**

You have finished fetching the contact from the database.

The next stage is to map the record retrieved by the Data Mapper to a variable. You already learned how to do this earlier in this lesson.

28. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Data Mapper** window's **Destination Tree** area.
29. Right-click on the Variable destination and select **Show Properties**.
30. Enter **ContactName** in the **Name** property.
31. In the **Properties** pane, click the **Variable** field's  button and select the **F.ContactName** variable.
32. Connect the **ContactName** node on the source to **F.ContactName** in the destination.



Remember that if no record is found, **F.ContactName** will be blank.

Now you need to condition this step so that if the previous step did not succeed there is no need to do this step.

33. Select **Condition** from the context menu of the step.
34. Set the following condition: **Trim ( F.CustomerName )<>"**.

If the contact does not exist, then you need to send them a reply email. This is something you have already done in a previous exercise.

35. Park on the **Toolbox** pane.
36. Click and drag the Email component and drop it on the **Check the contact** step.
37. In the Email component's **Properties** pane, enter **Send Rejection Email** in the **Step Name** field.
38. In the **Properties** pane's **Settings** section, make sure that the **Course email** resource has been automatically selected. It is automatically selected because it is the only email resource that you have defined.
39. Double-click the Email component, or right-click on it and select **Configuration** from the context menu.

The **Direct Access Method** dialog box opens:

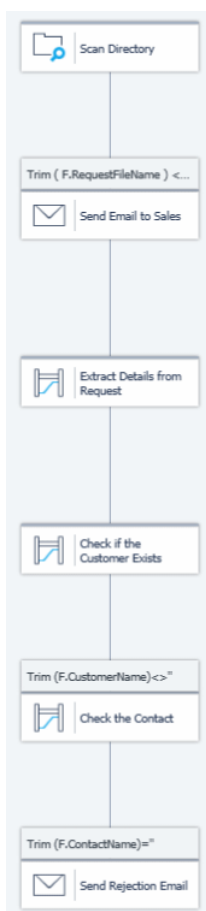
40. In the left pane, click **Add**.
41. Select **Quick Send** from the dropdown list.

In the right pane, you define the **Method Details**.

42. In the **To** property enter **F.CustomerEmail**.
43. In the **Subject** property, type **'Request status'**. The apostrophes are important here as this is a string value.
44. In the **Body** property, type **'You are not registered as an official contact for your company. Please approach your company representative.'**. The apostrophes are important here as this is a string value.
45. Click **OK**.

Now you simply need to condition the step:

46. Select **Condition** from the context menu of the **Send Rejection Email** step.
47. Set the following condition: **Trim ( F.ContactName )=''**.



## Solution Lesson 10 – Item Validity Check

### Requested price is too low

The client wants to purchase the item at a price less than the list price. The solution is very similar to the process where you checked whether there is enough stock. To check whether the price is less than the list price, do the following:

1. Under the **Check Item** flow, double-click on **Flow Variables**.
2. In the Flow Variables repository, add a flow variable named **F.RequestPrice** which is **Numeric** with a size of **9.2**.
3. Double-click the **Report Items Existence** Data Mapper step, or right-click on it and select **Configuration** from the context menu.
4. Right-click on the **CheckItem** source node and select **Show Properties**.
5. In the **Properties** pane, use the Database Wizard to update the SQL statement:
  - Click **Next** until you reach the columns selection dialog, and select the **CatalogPrice** column of the **Products** table. This will add the **CatalogPrice** as another selected column.
  - Click **Next** until you have finished, as you will not be making any other selections. You will receive a message that some of the connections may be lost, but you can ignore the messages by clicking **OK**.
6. In the Data Mapper window, connect **CatalogPrice** to the **C.All\_Items\_Exist** node.
7. Right-click on **C.All\_Items\_Exists** and select **Show Properties**. You may find that your previous **Calculated Value** expression has been removed. This happened as a result of modifying the SQL statement. In any case, you will need to enter a new expression:  
**Src.S1/Record/CatalogPrice < F.RequestPrice AND Src.S1/Record/StockQuantity >=F.Quantity AND C.All\_Items\_Exist**
8. Park on the **Scan for new requests** flow.
9. Double-click the **Check Items** Data Mapper step, or right-click on it and select **Configuration** from the context menu.
10. Map from **USPrice** to **F.RequestPrice**.

## The item does not exist

The client wants to purchase an item that does not exist in the catalog. There are a number of ways to solve this. The solution that will be explained here involves:

- Defining a flow variable.
- Updating the flow variable with FALSE.
- Updating the flow variable with TRUE in the Data Mapper.
- Updating **C.All\_Items\_Exist** with FALSE if the flow variable is FALSE.

How does it work? When the Data Mapper finds a record, it updates the flow variable with TRUE. If the flow variable is FALSE after exiting the Data Mapper, it indicates that the Data Mapper did not find a matching record and therefore there was no mapping connection.

1. Under the **Check Item** flow, double-click on **Flow Variables**.
2. In the Flow Variables repository, add a flow variable named **F.ItemExists** and set it as a **Logical** type.

You need to add a parent step to the **Report Items Existence** step:

3. Click and hold the keyboard **Ctrl** button.
4. Select the Flow Data utility from the **Toolbox** pane, and drag and drop it on the **Report Items Existence** step.
5. A menu will pop up. Select **Make As Parent**.
6. In the Flow Data utility, set the **Step Name** property to: **Update flow variable**.
7. Double-click the **Update flow variable** step, or right-click on it and select **Configuration** from the context menu.
8. In the **Flow Data Configuration** dialog box, add an entry for:
  - Action: **Update**
  - Type: **Flow**
  - Name: **F.ItemExists**
  - Update Expression: **'FALSE'LOG**

You can now modify the Data Mapper step:

9. Park on the **Report Item Existence** step. Select **Configuration** from the context menu.
10. In the **Destination Tree** area, right-click the **Update\_General\_Check\_Var** node and select **Show Properties**.
11. Select the **F.ItemExists** variable. Close the dialog box.
12. Park on the **F.ItemExists** node and select **Show Properties**.
13. Enter a **Calculated Value** for **'TRUE'LOG**.
14. Add a Flow Data utility as a child step of the **Report Item Existence** step.
15. In the Flow Data utility, set the **Step Name** property to: **Update context variable**

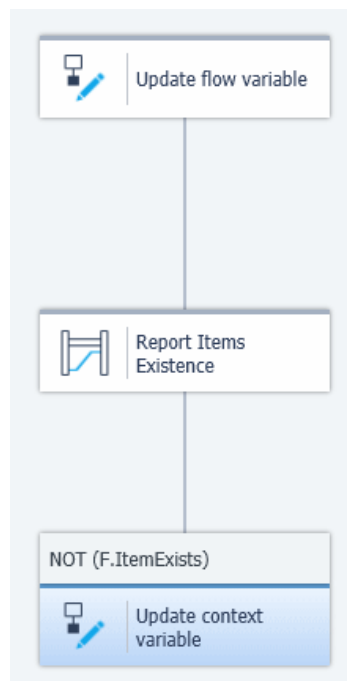
16. Add an entry for:

- ▣ Action: **Update**
- ▣ Type: **Context**
- ▣ Name: **C.All\_Items\_Exist**
- ▣ Update Expression: **'FALSE'LOG**

17. Set a condition for this step: **Not (F.ItemExists)**.

**F.ItemExists** is a logical variable that contains either TRUE or FALSE. There is no need to implicitly use a condition such as **F.ItemExists = 'FALSE'LOG**.

Your **Check Item** flow will now look similar to the image below:



## Adding the Request Information to the Database

To do this you need to have a Database type as the destination in a Data Mapper step. You can do this in the Data Mapper step that you have already defined. In this step you currently read from the Request XML and then call the **Check Item** flow to check each item. You will modify this step in the following way:

- ▣ In addition to the Call Flow destination, you will be adding a Database destination. You will use the Database destination to insert records to the **Requests** table.
- ▣ The items will be inserted in the called flow. You are already handling each item in the called flow so as part of the process you can add the current record to the database.

The default transaction level of the Data Mapper is for the entire step. Therefore, if there is a problem writing to one of the tables and there is a rollback, no data will be written to the database tables.

1. Park on the **Scan for New Requests** flow.
2. Double-click the **Check Items** step, or right-click on it and select **Configuration** from the context menu.

Bear in mind that because the data is being added within a transaction, you will not see them in the database until the **Check Items** step is completed.

3. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
4. Right-click on the Database destination and select **Show Properties**.
5. Enter **AddRequest** in the **Name** property.
6. Use the Wizard and:
  - ▣ Select **Delete** as the **DB Operation**.
  - ▣ Select the **Requests** table.
  - ▣ Select all of the columns from the **Requests** table.
7. In the **Data Mapper** window:
  - ▣ Park on the **Requests.RequestId** node of the **Request table** destination and select **Show Properties** from the context menu. Enter a **Calculated Value** for **F.RequestNum**.
  - ▣ Connect **Customer\_ID** from the source to **Requests.CustomerId** on the destination.
  - ▣ Connect **Request\_Total** from the source to **Requests.RequestTotal** on the destination.
  - ▣ Park on **Requests.CustomerExists** and select **Show Properties** from the context menu. Enter a **Calculated Value** for **Trim (F.CustomerName) <>"**. Remember that if the **Check Customer Exists** step failed to fetch a customer name, the **F.CustomerName** variable will be blank.

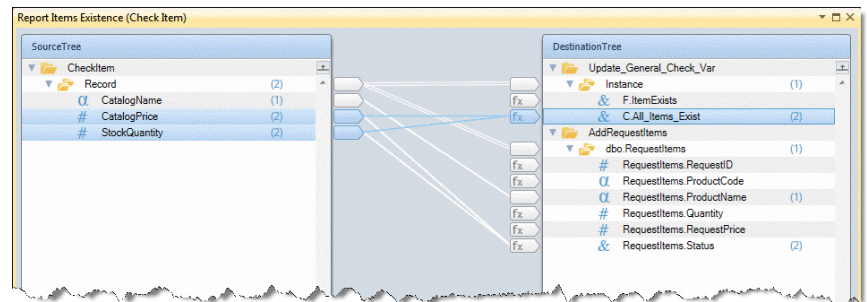


You have finished adding the request. Now you need to add the lines. You will do this in the called flow.

8. Park on the **Check Item** flow.
9. Double-click the **Report Items Existence** step, or right-click on it and select **Configuration** from the context menu.
10. Park on the Database source and select **Show Properties**. Use the Wizard and add the **CatalogName** column from the **Products** table.
11. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
12. Right-click on the Database destination and select **Show Properties**.
13. Enter **AddRequestItems** in the **Name** property.
14. Use the Wizard and:
  - ▣ Select **Insert** as the **DB Operation**.
  - ▣ Select the **RequestItems** table.
  - ▣ Select all of the columns from the **RequestItems** table.

15. In the **Data Mapper** window:

- ▣ Park on the **RequestItems.RequestId** node of the **RequestItems table** destination and select **Show Properties** from the context menu. Enter a **Calculated Value** for **F.RequestNum**.



- ▣ Park on the **RequestItems.ProductCode** node of the **RequestItems table** destination and select **Show Properties** from the context menu. Enter a **Calculated Value** for **F.ItemCode**.
- ▣ Park on the **RequestItems.Quantity** node of the **RequestItems table** destination and select **Properties** from the context menu. Enter a **Calculated Value** for **F.Quantity**.
- ▣ Park on the **RequestItems.RequestPrice** node of the **RequestItems table** destination and select **Show Properties** from the context menu. Enter a **Calculated Value** for **F.RequestPrice**.
- ▣ Connect **CatalogName** to **RequestItems.ProductName**.
- ▣ Connect **CatalogPrice** to **RequestItems.Status**.
- ▣ Connect **StockQuantity** to **RequestItems.Status**.
- ▣ Park on the **RequestItems.Status** node and select **Show Properties** from the context menu. Enter a **Calculated Value** with the following expression:  
**Src.S1/Record/CatalogPrice < F.RequestPrice AND Src.S1/Record/StockQuantity >= F.Quantity**

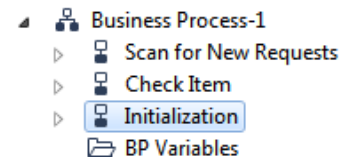
This is a similar expression to the expression you entered to update **C.All\_Items\_Exist**.

You can now run the flow and test your work.

## Initialization Flow

You have used the Data Mapper to select information from a table and you have now used the Data Mapper to insert records into a table. You will now use the Data Mapper to delete records from a table.

1. In the Solution Explorer, right-click on **Business Process-1** and select **Add Flow**.
2. Change the flow name to **Initialization**.
3. Select **Properties** from the context menu of the flow. Set the **AutoStart** property to **Yes**.
4. Add a Data Mapper step to the **Initialization** flow and name it **Delete Items DB**.
5. Double-click the **Delete Items DB** step, or right-click on it and select **Configuration** from the context menu.



In this step, you will not be using a source in the Data Mapper.

6. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
7. Right-click on the Database destination and select **Show Properties**.
8. Enter **DeleteRequests** in the **Name** property.
9. Use the Wizard and:
  - a. Select **Delete** as the **DB Operation**.
  - b. Select the **Requests** table. Note that you are unable to select other tables once this has been selected.
  - c. As you want to remove all the records, there is no need for a WHERE clause.
10. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
11. Right-click on the Database destination and select **Show Properties**.
12. Enter **DeleteRequestItems** in the **Name** property.
13. Use the Wizard and:
  - a. Select **Delete** as the **DB Operation**.
  - b. Select the **RequestItems** table. Note that you are unable to select other tables once this has been selected.
  - c. As you want to remove all the records, there is no need for a WHERE clause.

There is no need to map in this case.

You can now run this flow to test.

## Solution Lesson 11 – Services

You are going to add a service that you will be using in the next lesson. You are adding an HTTP service that you will need in the next lesson.

1. Open the **Projects** menu and select **Settings**. The **Settings** dialog box opens.

In this lesson, you added the **RequestConfirmation** service. If you did not add it, then select **Add** and, in the **New Service** dialog box, select **HTTP** and then type **RequestConfirmation**.

2. Park on the **RequestConfirmation** node and click **Endpoints**.

The **EndPoint** dialog box opens.

3. Click **Add** and type **check\_request\_status**.
4. Park on the **Argument Details** pane and click **Add**.
5. Enter **RequestNum** as the **Name** parameter. This must have a data type of **Numeric** and a size of **5**.
6. Select the **Generate Sample HTML** check box.  **Generate Sample HTML:**

When you click **OK**, Magic xpi automatically creates a sample HTML page under:

**My Documents\Magic\projects\<> project name>\<current project>\Service\<>Service name>**

As an example, the path to the HTML will be:

**C:\Users\<>your user name>\My Documents\Magic\projects\Magic\_xpi\_course\  
Magic\_xpi\_course\Service\RequestConfirmation\check\_request\_status.html**

If you click on the HTML in the path above, you will get an HTML similar to the image below. You can modify the look and feel of this HTML page.

### HTTP Trigger

**HTTP Post For Service : RequestConfirmation**  
**HTTP Post For Trigger : RequestConfirmation#check\_request\_status**

Parameter	Type	Picture	Mandatory	Value
RequestNum	Numeric	5	No	<input style="width: 100%;" type="text"/>

## Solution Lesson 12 – Checking Request Status

You need to return the **F.Result\_HTML** file to the Internet browser. You have only updated the header section in which you have either displayed the name of the customer or a hyperlink.

To update the **F.Result\_HTML** file you will be using the Data Mapper with a Template destination.

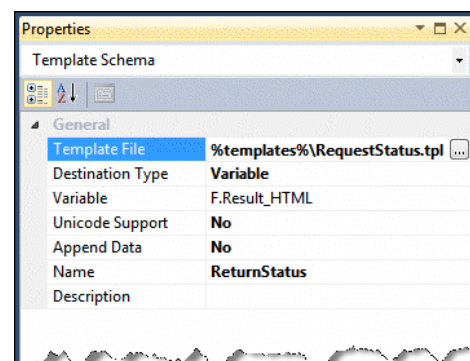
1. Park on the **Check Request Status** flow.
2. Add a Data Mapper step as the child step of **Get Customer Information**. Name this step **Send Response Page**.
3. Double-click the step, or right-click on it and select **Configuration** from the context menu.

As you have already fetched the customer details, you only need to fetch the request's lines:

4. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
5. Right-click on the Database source and select **Show Properties**.
6. Enter **Request\_items** in the **Name** property.
7. Use the Wizard and:

- Select the **RequestItems** table.
- Select **ProductCode**, **ProductName** and **Status** from the **RequestItems** table.
- Enter the following WHERE clause:  
**[RequestItems].RequestID=<?F.HTTP\_Request\_Num?>**

8. From the Toolbox's **Mapper Schemas** section, drag a Template destination into the **Destination Tree** area.
9. Right-click on the **Template** destination and select **Show Properties**.
10. Enter **ReturnStatus** in the **Name** property.
11. Set the **Template File** property to: **%templates%\RequestStatus.tpl** (if you remember, you already created the **templates** environment variable). If you do not have the **templates** environment variable set up, you can manually select the template file. The **tpl** extension is not a required extension. You can use any file as a template as long as it has the required Merge tags.
12. Set the **Destination Type** to **Variable** and select the **F.Result\_HTML** variable.



Now you have to map to update the variable.

In the **Data Mapper** window, you will see that the **Destination Tree** contains the Merge tags that were fetched from the template. The tags that were defined within an **MGREPEAT** section are listed as a compound node in the destination.

13. Park on the **Header** node and select **Show Properties** from the context menu. Enter a **Calculated Value** of **F.RequestHeader**. This is the header that you updated during the lesson.
14. Connect **ProductCode** to **ItemCode**.
15. Connect **ProductName** to **ItemName**.
16. Connect **Status** to **LineStatus**.
17. Park on the **LineStatus** node and select **Show Properties** from the context menu. Enter a **Calculated Value** with the following expression:  
**IF (Src.S1/Record/Status , 'All OK', 'There is a problem!')**


## File Management Component

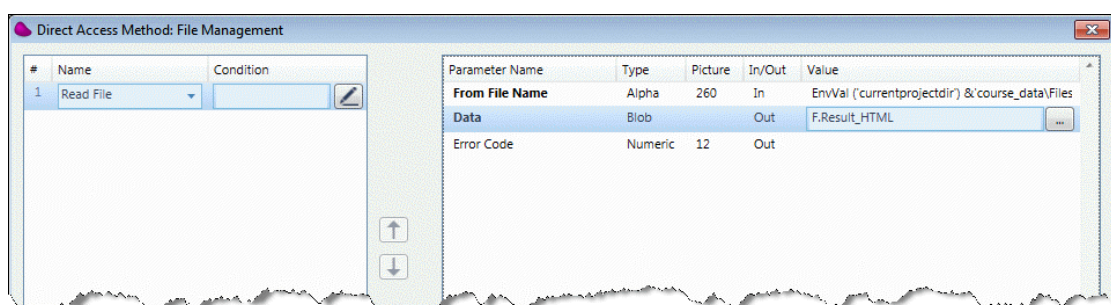
When the HTTP trigger invokes a flow, it is recommended to send a response page for every scenario of the flow logic. As you may have realized, the user may request information for a request that has not yet been logged in the system. To answer that need, you will return a predefined HTML file. You will now add a File Management component to the flow that loads a predefined HTML page when a request is not found.

The predefined file is currently located under the **Files** directory. The first step will be to create an environment variable that points to this location. It is important to do this as you will be referring to this in the next lesson.

1. From the **Project** menu, select **Settings**.
2. Under **General Environment**, click **User Environment Variables**.
3. Add an entry named **Files**, with a value of `%currentprojectdir%course_data\Files%sl%`.

You can now continue with the File Management step:

4. Add a **File Management** component as a child step of the **Get Information from DB** step.
5. Name this step: **Request Not Found**.
6. Double-click the step, or right-click on it and select **Configuration** from the context menu.
7. In the **Configuration** dialog box, add a method for **Read File**.
8. In the **From File Name** parameter, enter:  
**EnvVal ('currentprojectdir') &'course\_data\Files\RequestNotFound.htm'**  
You can do this with the Expression Assistor. You can select an environment variable by clicking .
9. In the **Data** parameter, select the **F.Result\_HTML**.



10. Condition the **Request Not Found** step to **NOT (F.RequestExists)**.
11. Park on the **Get Customer Information** step and condition the step to **F.RequestExists**.

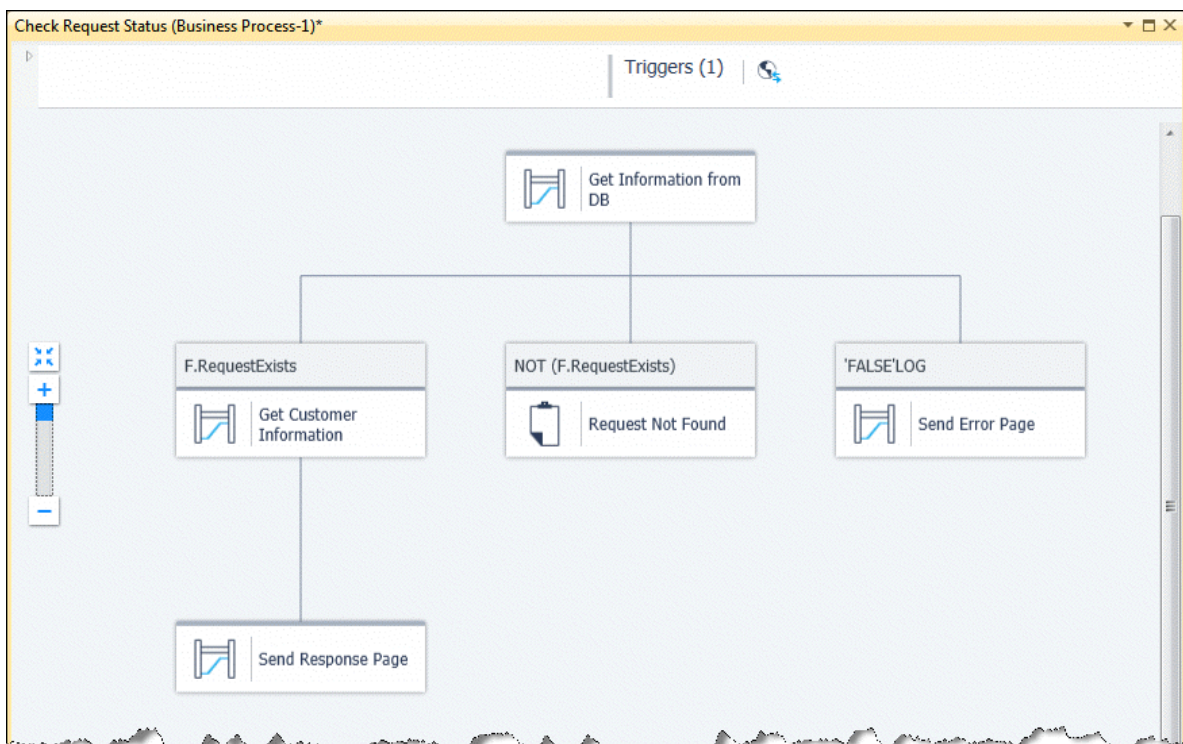


## Solution Lesson 13 – Error Handling

You are asked to handle a situation where the File Management component returns an error. In the specific example, the Internet browser must also be updated. As this is a specific issue, a generic flow will not be useful. Remember, of course, that you need to update the **F.Returned\_HTML** variable so that the requester can return it to the Internet browser.

To solve this issue, you can create an error policy in the same flow where the error occurs.

1. Park on the **Check Request Status** flow.
2. Add a Data Mapper step as the child step of **Get Information from DB**. Name this step **Send Error Page**.
3. Set a condition of **FALSE** for this step.



4. Double-click the **Send Error Page** step, or right-click on it and select **Configuration** from the context menu.

5. From the Toolbox's **Mapper Schemas** section, drag a Template destination into the **Destination Tree** area.
6. Right-click on the Template destination and select **Show Properties**.
7. Enter **ReturnErrorPage** in the **Name** property.
8. Set the **Template File** property to: **%templates%\Error.tpl**. Remember that the **tpl** extension is not a required extension. However **error.tpl** is the name of the file.
9. Set the **Destination Type** to **Variable** and select **F.Result\_HTML**.

You do not need to define anything in the **Source Tree**.

10. Expand the destination.

The template file has only two Merge tags.

11. Park on **ErrorCode** and enter a **Calculated Value** with the expression:  
**Str ( C.sys.LastErrorCode , '5')**
12. Park on **Errordescription** and enter a **Calculated Value** with the expression:  
**C.sys.LastErrorDescription**

Although you are adding a single step here you can add as many steps as needed. For example, you can add an email step that sends an email to the administrator. It is important that the administrator is also updated.

13. Add an **Email** component as a child step of the **Send Error Page** step. Name the step **Update Admin**.
14. Double-click the **Update Admin** step, or right-click on it and select **Configuration** from the context menu.
15. Select the **Method** interface. Select the **Quick Send** method and set the following:
  - a. To: **postmaster@magic.xpi.course.com**
  - b. Subject: **'Urgent: A necessary file was not found'**



**Remember:** The email step is provided purely as an example.

16. Under the **Check Request Status** flow, double-click **Error Policies**.
17. Click **Add**.
18. Click the zoom button in the **From** column. This loads the **Errors Repository** dialog box, where you can select an error for a specific component.
19. From the dropdown list, select **File Management**. Only the error types for the File Management component will now be displayed.
20. Select **200**, which is Folder does not exist.
21. In the **To** property, select **216**, which is **Operation failed**. This is the last component error defined by the File Management component.
22. In the **Error Policy** column, select **Jump**.
23. Click the zoom button in the **Step** column. You see only the steps in the current flow. Select the **Send Error Page** step.

You are done. You can test this by sending a request that has not been logged. Before doing that, rename the **Files** folder to something else.

The image below is similar to what will appear in your Internet browser.



## Solution Lesson 14 – Adding a Customer

In a previous lesson, you learned how to accept a request from the Web using the HTTP component. This process returned an HTML page that included a link for adding a new customer if the customer did not exist in the database.

In the current lesson, you added the **Add Customer** flow which was triggered by a Web service. Using the request number sent from the Web Service, you fetched the request data from the ODS and inserted information into the **Customers** table and the **Contacts** table.

Therefore, the necessary steps for adding a customer exist. You will add another trigger to this flow that completes the **Add Customer** scenario. This is triggered from the Internet browser.


You will first add a new endpoint to the HTTP Service.

1. From the **Project** menu, select **Settings**. Open the **Services** node and select the **RequestConfirmation** service.
2. Click **Endpoints** to open the **Endpoint** dialog box. Click **Add** to add a new endpoint.
3. Set the **Endpoint Name** to **add\_customer**.
4. Park on the **Arguments** pane and click **Add**.
5. Enter **RequestNum** as the **Name** parameter. This must be a **Numeric** data type and have a size of **5**.

Now you can use the **Add Customer** flow:

6. Park on the **Add Customer** flow.
7. Add another BLOB type flow variable, **F.ResultHTML**. This will contain the returned HTML information.
8. Drop an HTTP component into the Triggers area and set the name to **Add Customer**.
9. Double-click the HTTP trigger, or right-click on it and select **Configuration** from the context menu.
10. Open the **Endpoint Name** dropdown list. You will now see two entries. Select the **add\_customer** entry.

In the **Argument Details** section:

11. Park on **Mapping Variable** and click the  button. Select **F. RequestNum** from the selection list.
12. Park on the **Return Value** variable and select the **F.ResultHTML** variable.
13. Click **OK** to close the dialog box.

A new trigger has been added to the flow.

The next stage is to return the information to the Internet browser that the customer was added.

1. Add a **Data Mapper** utility as a child step of the **Add Customer to DB** step. Name the step **Update Browser**.
2. Double-click the Data Mapper, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

3. From the Toolbox's **Mapper Schemas** section, drag an XML source into the **Source Tree** area.
4. Right-click on the XML source and select **Show Properties**.
5. Enter **RequestXML** in the **Name** property.
6. In the **XSD File** field, type: `course_data\schemas\request.xsd`
7. Set the **Data Source** to **Variable** and select the **F.RequestXML** variable.
8. From the Toolbox's **Mapper Schemas** section, drag a Template destination into the **Destination Tree** area.
9. Right-click on the Template destination and select **Show Properties**.
10. Enter **Customer\_was\_added** in the **Name** property.
11. In the **Template File** field, type: `%Templates%CustomerAdded.tpl`
12. In the **Destination Type** field, select **Variable** and select the **F.ResultHTML** flow variable.
13. Expand the source and destination.
14. Connect **Customer\_Name** to **CustomerName**.

You could, of course, have added this operation to the previous step. It was separated in this example for clarity.

## Update Customer Status

The last stage is to update the Requests table that the customer now exists. You will now configure the Data Mapper utility with an update statement.

1. Add a **Data Mapper** utility as a child step of the **Update Browser** step. Name the step **Change Customer Status in DB**.



You can add all the steps in this solution as a single Data Mapper step.

2. In the **Description** field, type: **This step updates the DB using a dynamic WHERE clause.**
3. Double-click the Data Mapper, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

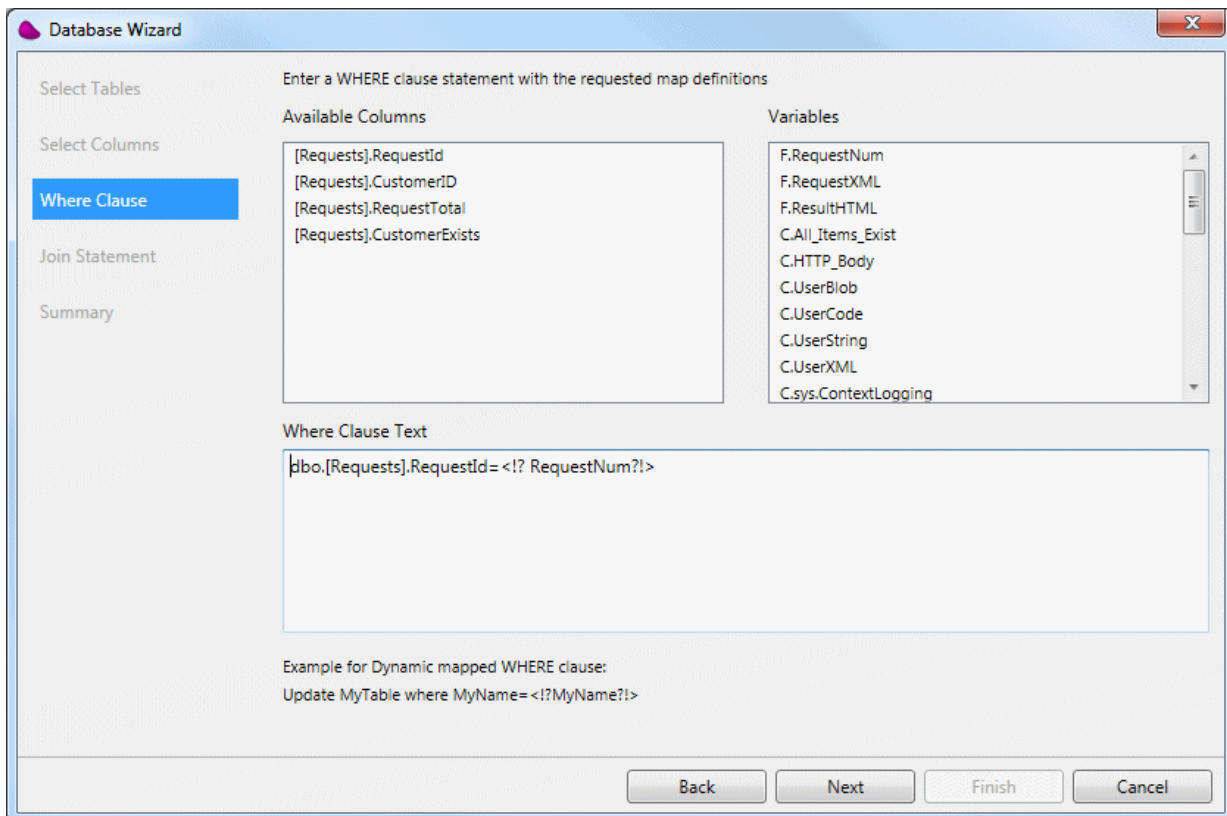
4. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
5. Right-click on the Database destination and select **Show Properties**.
6. Enter **Customer\_Details** in the **Name** property.
7. Open the **Wizard**.
8. Set the **DB Operation** field to **Update**. This is the first time you are using this operation.
9. Select the **Requests** table and then select the **CustomerExists** column.
10. Click **Next**.
11. In the **Where Clause** screen, type:  
**RequestId = <!?RequestNum?!>** (See note below.)



When specifying a dynamic name in the **Where Clause Text** section, the name must be surrounded with **<!?** as the prefix and **?!>** as the suffix

In the Data Mapper, the Where compound will contain a new node and the name will be the one that you specified in the **<!?Name?!>** tag.

You can then connect values to the node; thereby building a Dynamic Where Clause.



You have completed the Database Wizard for an **Update** statement.

The final result will be:

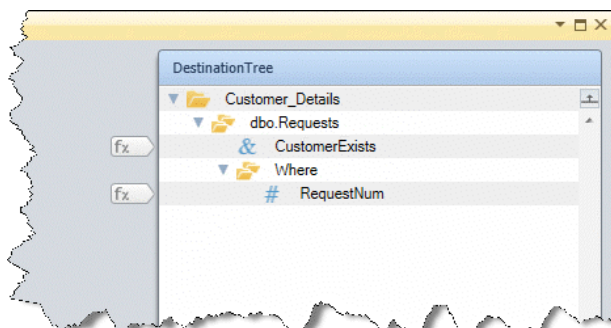
**UPDATE [Requests] SET CustomerExists=<!?CustomerExists?!> WHERE RequestId=<!?RequestNum?!>**

12. In the **Data Mapper** window, expand the destination.

13. Park on the **CustomerExists** node and enter a **Calculated Value** of 'TRUE'LOG.

14. Park on the dynamic variable, **RequestNum**, and enter the following **Calculated Value**:

**F.RequestNum**



You are now ready to test.

## Solution Lesson 15 – Handling Approved Requests

During the lesson, you were asked if using the flow variable, **F.RequestNum**, every time you defined a flow was the ideal solution. The answer is that you could define a context variable, **C.RequestNum**. The value in this variable is valid as long as the context is alive, but the variable definition exists throughout the project.

The request is still in the local databases, you need to remove it from the database so that the request will not be handed a second time.

1. Park on the **Process Request** flow.
2. Add a Data Mapper step as a child step of **Add Delivery Details** and name it **Remove from DB**.
3. Set the processing mode to **Parallel**.
4. Double-click the Data Mapper, or right-click on it and select **Configuration** from the context menu.

In this step, you will not be using a source in the Data Mapper.

5. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
6. Right-click on the Database destination and select **Show Properties**.
7. Enter **DeleteRequests** in the **Name** property.
8. Use the Wizard and:
  - ▣ Select **Delete** as the **DB Operation**.
  - ▣ Select the **Requests** table. Note that you are unable to select other tables once this has been selected.
  - ▣ As you want to remove only those records matching the current request, set the WHERE clause to: **RequestId = <?F.RequestNum?>**.
9. From the Toolbox's **Mapper Schemas** section, drag a Database destination into the **Destination Tree** area.
10. Right-click on the Database destination and select **Show Properties**.
11. Enter **DeleteRequestItems** in the **Name** property.
12. Use the Wizard and:
  - ▣ Select **Delete** as the **DB Operation**.
  - ▣ Select the **RequestItems** table. Note that you are unable to select other tables once this has been selected.
  - ▣ As you want to remove only those records matching the current request, set the WHERE clause to: **RequestId = <?F.RequestNum?>**.

There is no need to make any mapping connections in this case.

In the **Add Customer** flow, you added a customer if it did not appear in the database. The customer may have shippable requests. If there are shippable items, you will publish the topic so that they may be added to the delivery database. Remember that a shippable item is one where the status is 1. There are many ways to do this. You can use the solution from Lesson 11:

- Defining a flow variable.
- Updating the flow variable with FALSE.
- Updating the flow variable with TRUE in the Data Mapper.
- If the flow variable is TRUE, then you can publish.

How does it work? When the Data Mapper finds a record, it updates the flow variable with TRUE. If the flow variable is FALSE after exiting the Data Mapper, it indicates that the Data Mapper did not find a matching record and therefore there was no mapping connection.

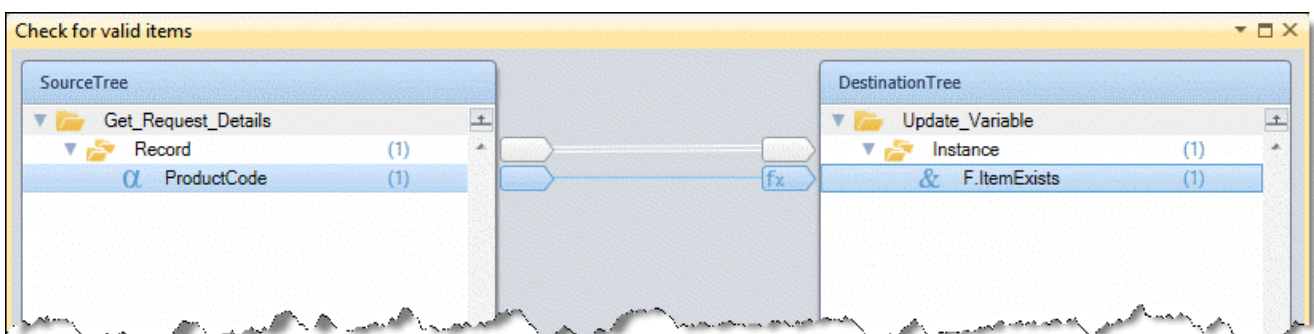
1. Park on the **Add Customer** flow.
2. Add a flow variable named **F.ItemExists** which has a **Logical** type.
3. Select the Flow Data utility from the Toolbox, and drag and drop it on the **Add Customer to DB** step.
4. In the Flow Data utility, set the name property to: **Update flow variable**.
5. Set the processing mode to **Parallel**.
6. Double-click the Flow Data utility, or right-click on it and select **Configuration** from the context menu.
7. Add an entry for:
  - Action: **Update**
  - Type: **Flow**
  - Name: **F.ItemExists**
  - Update Expression: **'FALSE'LOG**

Defining the Data Mapper step:

8. Add a Data Mapper step as the child step of **Update flow variable**. Name the step **Check for valid items**.
9. Double-click the Data Mapper, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

10. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
11. Right-click on the Database source and select **Show Properties**.
12. Enter **Get\_Request\_Details** in the **Name** property.
13. Use the Database wizard:
  - ▣ Select the **RequestItems** table.
  - ▣ Select the **ProductCode** column.
  - ▣ Set the WHERE clause to: **[RequestItems].RequestID=<?F.RequestNum?> and [RequestItems].Status=1**  
**Status** is a logical field, but is defined in the database as **BIT**. It accepts **0** or **1**.
14. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Destination Tree** area.
15. Right-click on the Variable destination and select **Show Properties**.
16. Enter **Update\_Variable** in the **Name** property.
17. Expand the source and destination.
18. Connect the **ProductCode** source node to **F.ItemExists**. The Data Mapper needs at least one connection.
19. Right-click on the **F.ItemExists** node and select **Show Properties** from the context menu.
20. Enter the following **Calculated Value**: **'TRUE'LOG**.



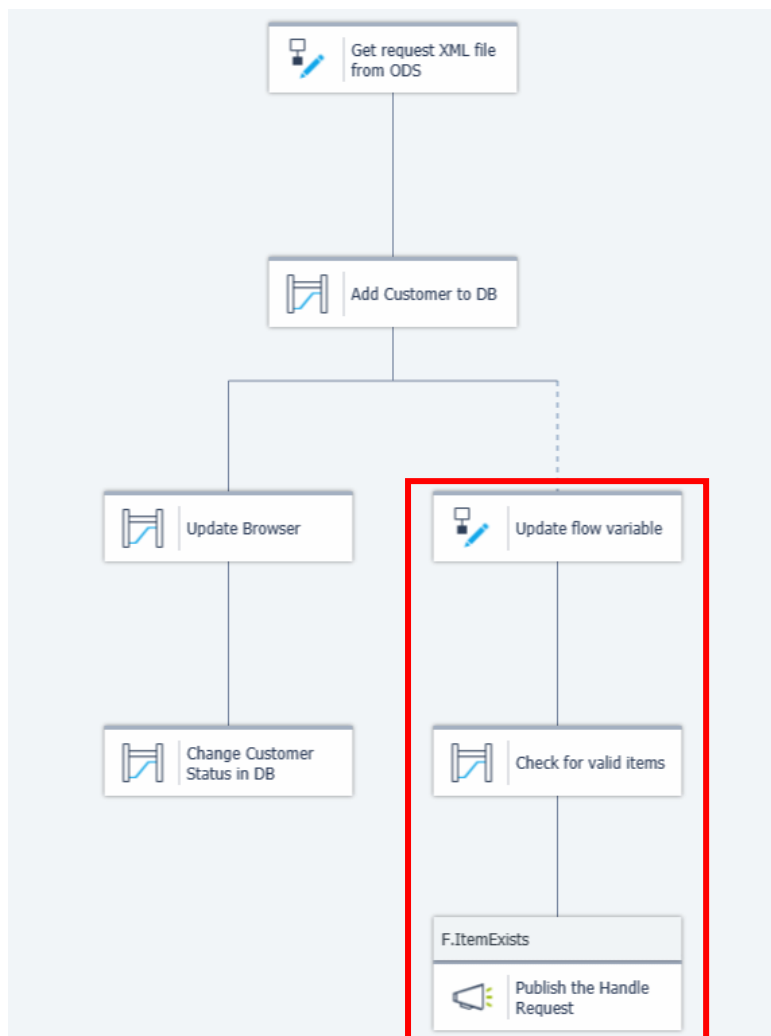


The last stage is to publish the topic:

21. Add the PSS Publish utility as a child step of the **Check for valid items** step.
22. Name the step **Publish the Handle Request**.
23. Double-click the PSS Publish step, or right-click on it and select **Configuration** from the context menu.
24. Zoom from the **PSS Name** entry and enter **Handle Request** in the Expression Editor.
25. In the **Code** parameter, set the following expression: **F.RequestNum**.
26. Click **OK** to confirm.

Now you need to set a condition for this step.

27. Set a condition for the **Publish the Handle Request** step: **F.ItemExists**.




You have now published to the same flow from two different flows.

## Solution Lesson 16 – Automatic Item Check

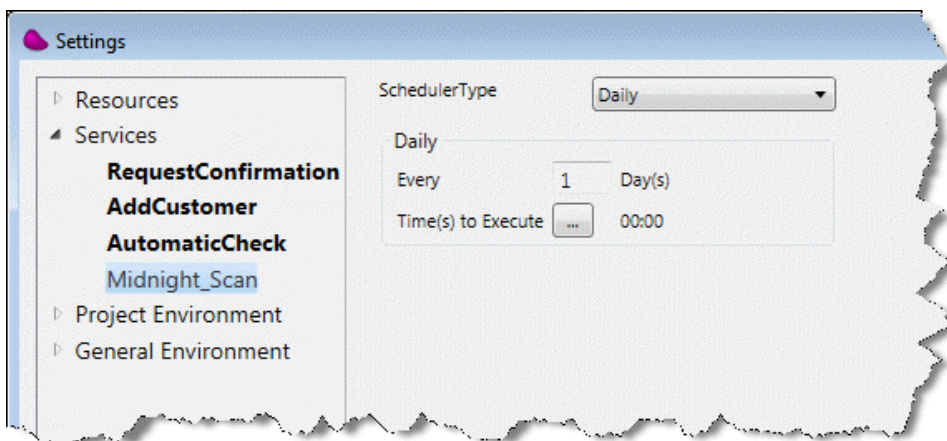
In the lesson, you fixed problematic requests. However, these requests are not passed to the delivery system.

You will now create a flow that will scan all requests in a timely manner.

1. Insert a new flow named **Scan DB for valid requests**.
2. Right-click on the flow name and select **Move Flow Up**. Move it above the **Check Reason** flow.
3. In the **Settings** dialog box's **Service** section, add a new Scheduler service. Call it **Midnight\_Scan**.
4. Select **Daily** from the **Scheduler Type** pulldown menu.
5. Park on the **Time(s) to Execute** property and click the  button.

The **Enablement Time** dialog box opens.

6. Click **New**.
7. Set the time to **00:00**.



8. Drop a Scheduler utility into the flow's Triggers area.
9. In the **Name** field, type **Midnight Scan**.
10. Double-click the Scheduler utility step, or right-click on it and select **Configuration** from the context menu.
11. In the **Scheduler Utility** dialog box, click **New** and type **MidnightScan** in the **Name** column.
12. In the **Select Scheduler Service** field, select **Midnight\_Scan** to use the Scheduler settings that you defined above.
13. Click **OK**.

The Scheduler is now set to midnight.

You now need to scan the system for open requests where the customer also exists. To do this you need to:

- Scan the **Requests** table for requests and fetch the Request number.
- For each request, scan the request items table and check whether there is a valid request. If there is a valid request, you need to publish to the handle request topic. You will perform this part of the scenario in a separate flow.
  1. Insert a new flow named **Scan requests items**.
  2. Right-click on the flow name and select **Move Flow Up**. Move it to just below the **Check Reason** flow.
  3. Add the following flow variables:
    - **F.RequestNum** – a Numeric variable of size **5**.
    - **F.ItemExists** – a Logical variable.
  4. Add a Flow Data utility from the Toolbox. Set the **Name** property to: **Update flow variable**.
  5. Add an entry for:
    - Action: **Update**
    - Type: **Flow**
    - Name: **F.ItemExists**
    - Update Expression: **'FALSE'LOG**

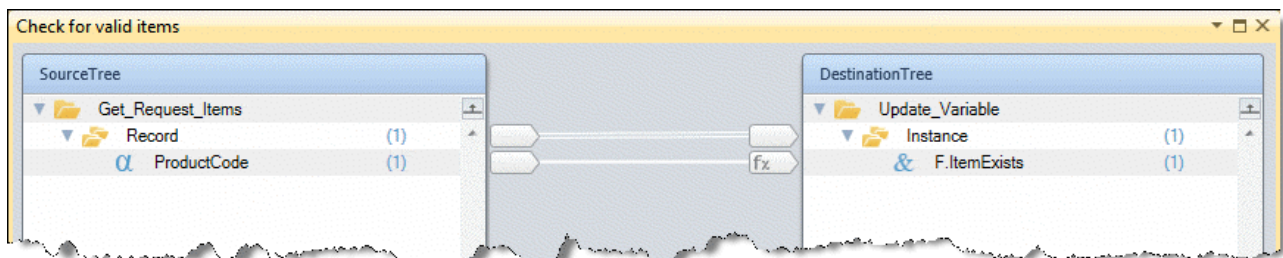
Defining the Data Mapper step:

6. Add a Data Mapper step as the child step of **Update flow variable**. Name the step **Check for valid items**.
7. Double-click the Data Mapper step, or right-click on it and select **Configuration** from the context menu.

The **Data Mapper** window opens.

8. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
9. Right-click on the Database source and select **Show Properties**.
10. Enter **Get\_Request\_Items** in the **Name** property.
11. Use the Wizard and:
  - Select the **RequestItems** table.
  - Select the **ProductCode** column.
  - Set the WHERE clause to: **[RequestItems].RequestID=<?F.RequestNum?>** and **[RequestItems].Status=1**  
**Status** is a logical field but is defined in the database as BIT. It accepts **0** or **1**.

12. From the Toolbox's **Mapper Schemas** section, drag a Variable destination into the **Destination Tree** area.
13. Right-click on the Variable destination and select **Show Properties**.
14. Enter **Update\_Variable** in the **Name** property.
15. Select the **F.ItemExists** variable.
16. Expand the source and destination.
17. Connect the **ProductCode** source node to **F.ItemExists**. The Data Mapper needs at least one connection.
18. Right-click on the **F.ItemExists** node and select **Show Properties**.
19. Enter a **Calculated Value** for 'TRUE'LOG.



The next step is to publish the request if an item was found:

20. Add the PSS Publish utility as a child step of the **Check for valid items** step.
21. In the **Name** field, type **Publish the Handle Request**.
22. Double-click the PSS Publish step, or right-click on it and select **Configuration** from the context menu.


The **PSS Publish Configuration** dialog box opens.

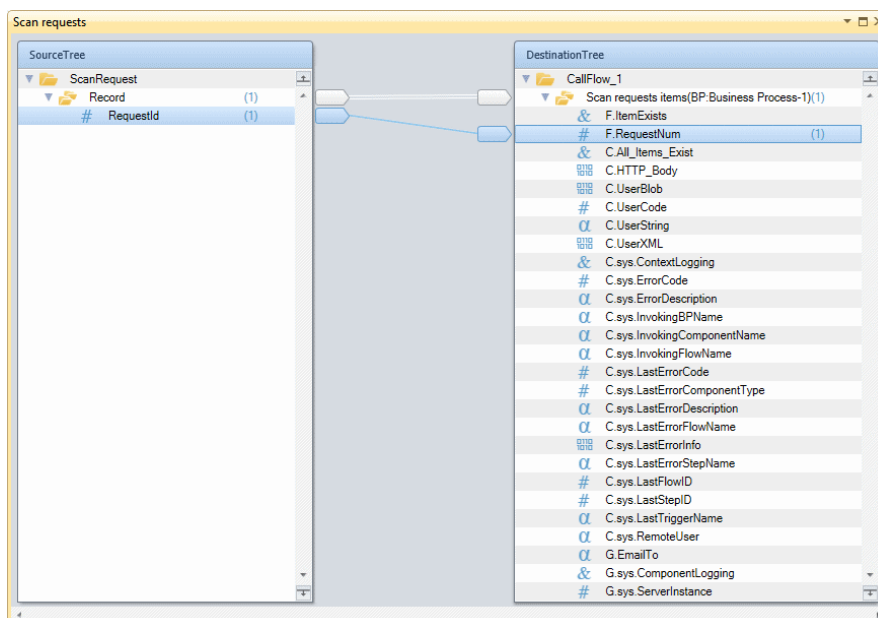
23. Zoom from the **PSS Name** entry and enter **Handle Request** in the Expression Editor.
24. In the **Code** parameter, set the following expression: **F.RequestNum**.
25. Click **OK** to confirm.

Now you need to set a condition for this step.

26. Set a condition for the **Publish the Handle Request** step: **F.ItemExists**.

All that is left to do is to call this flow:

1. Park on the **Scan DB for valid request** flow.
2. Add a Data Mapper step as the first step and name it **Scan requests**.
3. Double-click the Data Mapper step, or right-click on it and select **Configuration** from the context menu.
4. From the Toolbox's **Mapper Schemas** section, drag a Database source into the **Source Tree** area.
5. Right-click on the Database source and select **Show Properties**.
6. Enter **ScanRequest** in the **Name** property.
7. Use the Wizard and:
  - Select the **Requests** table.
  - Select the **RequestId** column.
  - Enter the following WHERE clause: **[Requests].CustomerExists=1**
8. From the Toolbox's **Mapper Schemas** section, drag a Call Flow destination into the **Destination Tree** area.
9. Right-click on the Database source and select **Show Properties**.
10. Enter **ScanItems** in the **Name** property.
11. From the **Flow Name** property, click the  button and select the **Scan Requests Items** flow.
12. Expand the source and destination.
13. Connect **RequestId** to **F.RequestNum**.



This flow is the last part of the cleanup of the invalid requests. Every night at midnight, the Scheduler will run and clean the system.

## Solution Lesson 17 – More About Magic xpi

### Mapping Flat File to Order


This is an exercise which has nothing to do with the course scenario. It is provided as an extra scenario so that you may understand other techniques.

1. Insert a new flow named **Create XML Order from ASCII**.
2. Add a Data Mapper to the flow.
3. Name it **ASCII Order to XML**.
4. Double-click the Data Mapper step, or right-click on it and select **Configuration** from the context menu.
5. From the Toolbox's **Mapper Schemas** section, drag a Flat File source into the **Source Tree** area.
6. Right-click on the Flat File source and select **Show Properties**.
7. Enter **ASCII\_Order** in the **Name** property.

The comma-delimited file name is: **order.txt**.

The file is located in the **%currentprojectdir%course\_data\Files** folder.

You can take a look at the file to better understand the file structure and to see how it should be set in the **Flat File** properties dialog box.

8. In the **Flat File** source's **Properties** pane, set the following properties:
  - Set the **Source Type** property to **File**.
  - Set the **File Path** to: **EnvVal('currentprojectdir') &'course\_data\Files\Order.txt'**.
9. In the **Lines** property, click .
10. In the **Flat File** dialog box, define the following:

Name	Data Type	Format	From	Length
Identifier	Alpha	30		30
Field1	Numeric	9		9
Field2	Numeric	9		9
Price	Numeric	5.2		9
Name	Alpha	30		30

11. From the Toolbox's **Mapper Schemas** section, drag an XML destination into the **Destination Tree** area.
12. Right-click on the XML destination and select **Show Properties**.
13. Enter **XML\_Order** in the **Name** property.
14. Set the **XSD File** to: `course_data\Files\Order.xsd`
15. Set the **Destination Type** to **File**. Create the file in:  
`EnvVal ('currentprojectdir')&'course_data\out\order.xml'`

## Data Mapping

The idea behind this example is that the same field in the source file should be mapped to both **Header** columns and **Item** columns.

In addition, the first column should be processed to separate the line type indicator and the actual data.

By using an expression in the **CustomerID** and **ItemID** elements, you can filter the first column's source data. Using an expression in the destination compounds helps you filter the mapped data.

In the **Data Mapper** window, map the following sources to destinations:

1. Connect **Record** to **Header** and **Item**.
2. Connect **Field1** to **CustomerID** and **ItemID**.
3. Connect **Field2** to **OrderID** and **Quantity**.
4. Connect **Price** to **ItemPrice**.
5. Connect **Name** to **ItemName**.

Setting an **Identifier = Header** expression in the **Header** compound ensures that only **Header** type rows will be written to this compound. The line identifier is the first character in each line.

Note that when you map compounds, all of their columns/elements are available in the Expression Editor.

In the **Destination Tree**:

6. Park on the **Header** node and select **Show Properties**.
7. Park on the **Condition** property and select a condition for:  
`Src.S1/Record/Identifier ='H'`

You will repeat this for the **Item** node.

8. Park on the **Item** node and select **Show Properties**.
9. Park on the **Condition** property and select a condition for:  
`Src.S1/Record/Identifier ='L'`

