Getting Started with Magic xpa 3.x and Mobile

Self-Paced Tutorial

Book ID: UTGSMAM32

Edition: 1.00, September 2016

Course ID: UCGMXAM3x

Magic University Official Courseware

Getting Started with Magic xpa 3.x and Mobile

September 2016

Contents

Introduction

Welcome to Magic Software University's **Getting Started with Magic xpa 3.x and Mobile** 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.

About Magic xpa

Magic xpa provides all aspects of the application development and deployment process within a single end-to-end platform. It features a ready-made business application engine that simplifies the code-writing process and enables you to deploy to market faster, using fewer resources.

Applications developed using Magic xpa typically have fewer coding mistakes, undergo more thorough prototyping, benefit from greater business side input and optimization, and can be more easily adapted to changing business needs.

Magic xpa enables you to focus more on the business logic of the application and less on what is happening behind the scenes.

Install the Magic xpa Studio and ensure that your computer meets the hardware requirements explained on the next page.

About the Course

Magic University's **Getting Started with Magic xpa 3.x and Mobile** course is designed to provide the student with a fundamental basic understanding of the concept behind Magic xpa as well as tools for Magic xpa programming. The course will teach you all of the basic skills that are needed to start programming in Magic xpa.

In this course you will:

1. Learn the fundamentals of Magic xpa and how to get the best out of Magic xpa.
2. Become familiar with the Magic xpa Studio interface.
3. Get to know the Magic xpa wizards and utilities.
4. Understand the Magic xpa concepts and standards.
5. Create a basic Magic xpa business application that:
    - Has a full Mobile interface.
    - Works with an SQL database, SQLite
    - Exhibits one-to-one and one-to-many data source relationships.
    - Produces reports.

Course Prerequisites

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

Development knowledge	Familiar with databases – tables, rows, fields, indexes and segments

Your computer must also meet some basic requirements:

Hardware	• Windows XP Pro and later. The course was tested on Windows 7. • Pentium processor 1.8GHz and upwards (recommended: Dual-core, 2.66-GHz or faster) • 1GB of RAM or greater • Screen resolution of at least 1024x768 pixels
Software	• IIS 7 or later • .NET Framework 4 or later • For the Studio, you also need to have .NET Framework V2.0 SP2 or V3 installed on your machine.

Mobile	Android SDK 1 installed on your desktop or laptop. This is needed for the lesson on customization. Customization for iOS requires using a MacBook as well as a valid *iOS Developer Enterprise* Program account. Customizaton for windows 10 Mobile requires Visual Studio 2015 with the Universal Windows App Development Tools and SQLite for Universal Windows Platform. These will not be discussed in this course. It is strongly advised to perform the exercises using a tablet.
License	The course uses the MGDEMO license. This is a single user license.

How to Use this Guide

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

This guide also contains boxes with more information and tips. Here is what you may find:

This is a hint.

This is an important note.

? This is a question for you. Something to think about.

Exercises

At the end of most lessons there is an exercise. The solutions for the exercises are provided at the end of this book. 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.

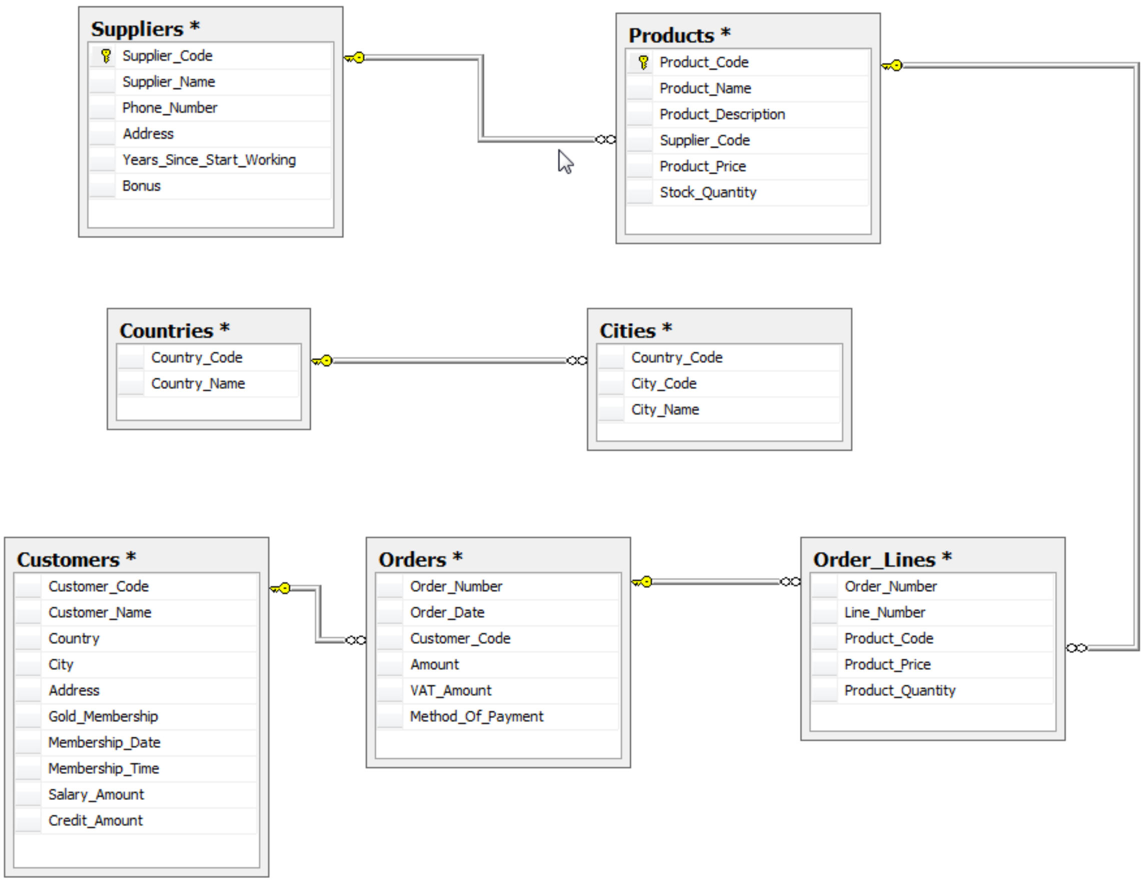
Course Materials

The course materials contain the following:

- Course data – This is data that you will need during the course exercises. This includes the course database, images and other data files.
- Classroom solutions

Entity Relations Diagram (ERD)

The diagram below shows the relationship between the SQLite tables that you will be using in this course.



Lesson 1

Magic xpa Studio Interface

Magic xpa has a graphical user interface that is fully compatible with current Microsoft Windows operating systems. The interface is composed of three main parts: the Navigator pane, the work area and the status line. In this lesson you will learn about the Studio interface and how to dynamically control the displayed items.

The Magic xpa Studio interface is composed of three main parts:

- Navigator pane
- Work area
- Status line

The Navigator pane enables you to select one of the Magic xpa repositories to be displayed, such as the Data or Program repository.

An important part of the Studio interface is the property sheet. Most of Magic xpa's objects have dynamic properties, which are organized in a property sheet. The property sheet can be displayed as a floating window, docking window, or as a tab in the Navigator pane.

The interface enables you to view the comments of the project's objects using the Comments pane and the checker results using the Checker pane.

The line at the bottom of the Studio interface is the **status line**, which displays different information such as editing status and user name.

By the end of this lesson you will:

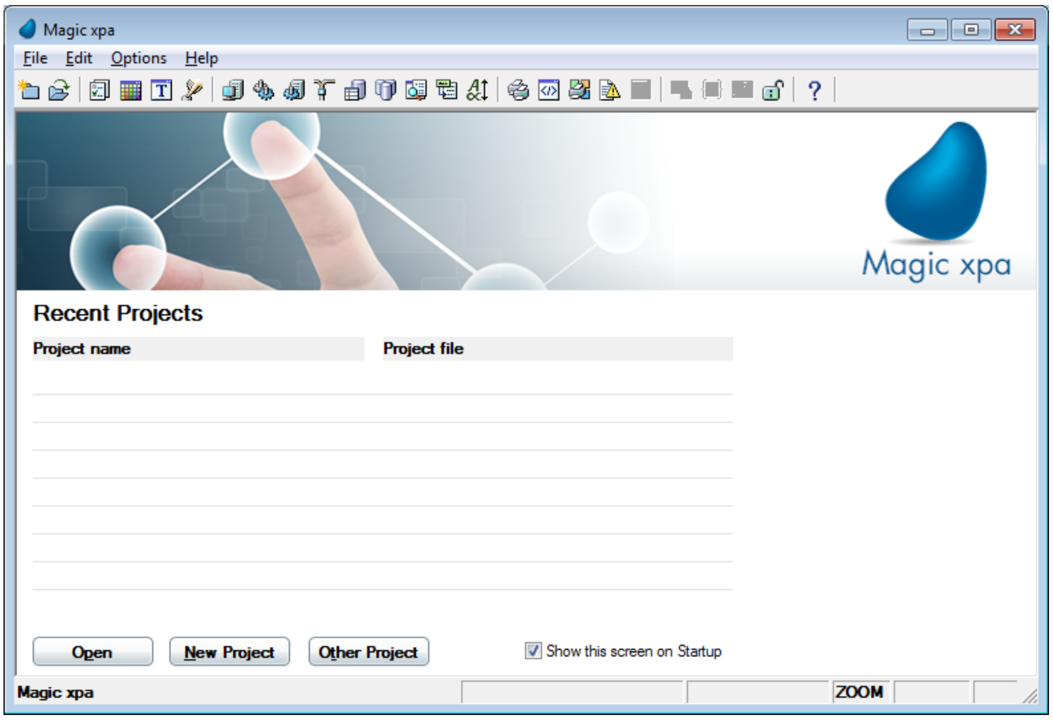
- Know how to create a new project.
- Be familiar with the Studio interface.
- Know how to dynamically navigate between the repositories.
- Know how to view an object's comments.
- Know how to view the latest checker results.

Creating a New Project

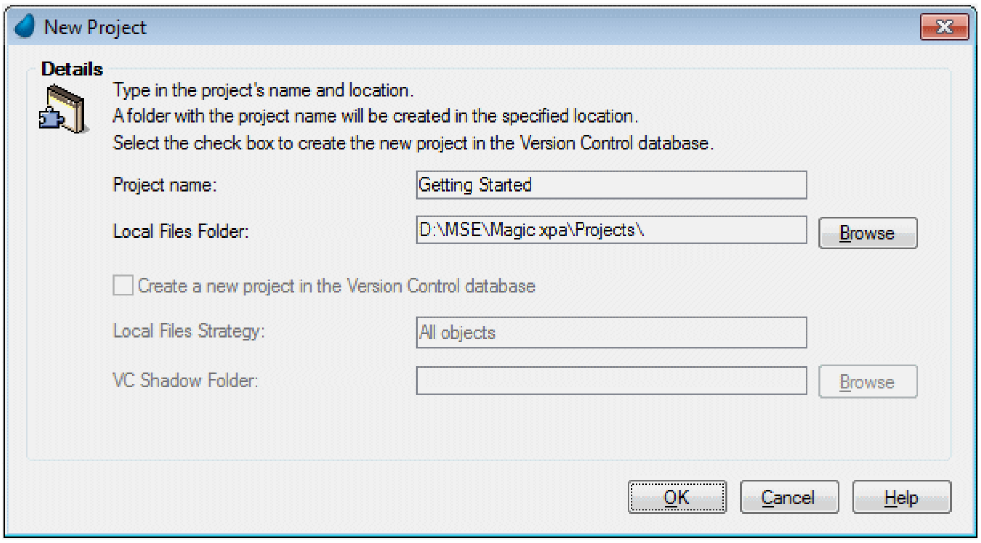
The first step in programming in Magic xpa is creating a project. In Magic xpa, a project is a collection of units that includes all the information related to a specific implementation. This includes the main code, any related components, and references to other projects. The main code is basically composed of models, data sources, programs, and menus.

Now, you will create the **Getting Started** project. The models, data sources, programs, and menus will be discussed later in this course.

1. Open the Magic xpa Studio.
2. In the Startup screen, click the **New Project** button.



The **New Project** dialog box opens.



Type in the project's name and location.
A folder with the project name will be created in the specified location.
Select the check box to create the new project in the Version Control database.

Project name:	Getting Started
Local Files Folder:	D:\MSE\Magic xpa\Projects\

☐ Create a new project in the Version Control database

Local Files Strategy:	All objects
VC Shadow Folder:	

3. In the **Project name** entry, type **Getting Started**.
4. The **Local Files Folder** entry has a default value of:
   [Magic xpa path]\Projects
   Use the Browse button if you want to specify a different location.
5. Click **OK**.

As a result of the project creation process, the project files are created and the new project is opened.

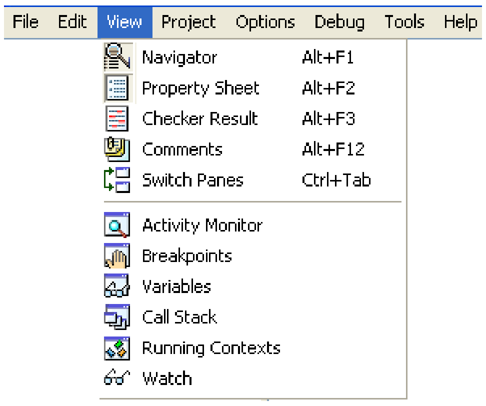
The Navigator Pane

The Navigator pane lets you navigate between the project's objects in the Studio environment.

Displaying the Navigator pane as part of the Studio interface is optional.

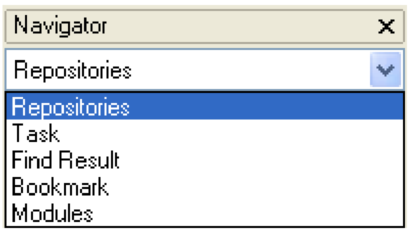
The Navigator pane is usually located on the left side of the Studio.

1. From the **View** menu, select **Navigator** (Alt+F1) to open or close the Navigator pane.



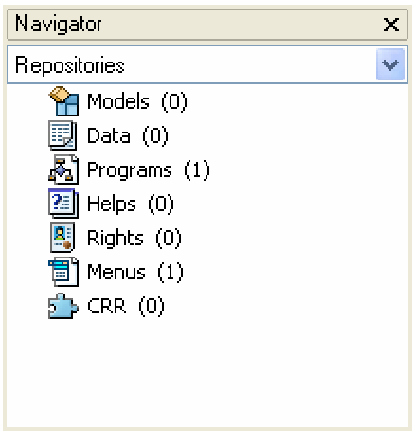
The first entry in the Navigator pane is a selection box, which enables you to display one of five categories.

2. Open the drop-down list to view the categories.
3. Select the **Repositories** option.



The Repositories View

The Repositories view displays a list of Magic xpa's main repositories and a number that indicates the number of items in each repository.


Navigator

Repositories

Models (0)
Data (0)
Programs (1)
Helps (0)
Rights (0)
Menus (1)
CRR (0)

Selecting a repository (by clicking the option in the Repositories view), displays it in the work area. For example, by selecting the **Programs** option, the project's Program repository is displayed.

The Property Sheet

The property sheet enables you to display the specific properties for a selected object.

To display the property sheet, you select the **Property Sheet** option from the **View** menu.

The image displayed below is an example from an existing program. Later in this course, you will be able to access similar property sheets in your own program.

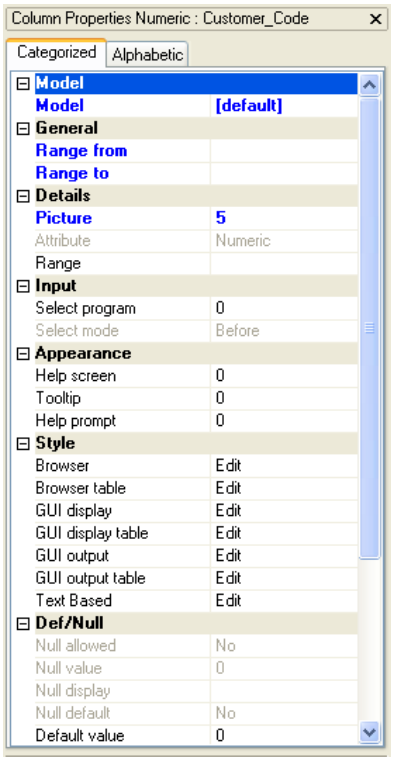
Property sheets are available for the following objects:

- Models
- Variables and Columns
- Forms
- Controls
- Help Screens

The displayed property sheet dynamically changes according to the selected object.

You can sort the displayed properties alphabetically or by category by clicking the **Alphabetic** or **Categorized** tab.

The modified properties are displayed in a different color (blue or bold).


Column Properties Numeric : Customer_Code

Categorized | Alphabetic

⊟ **Model**	
Model	**[default]**
⊟ **General**	
Range from	
Range to	
⊟ **Details**	
Picture	**5**
Attribute	Numeric
Range	
⊟ **Input**	
Select program	0
Select mode	Before
⊟ **Appearance**	
Help screen	0
Tooltip	0
Help prompt	0
⊟ **Style**	
Browser	Edit
Browser table	Edit
GUI display	Edit
GUI display table	Edit
GUI output	Edit
GUI output table	Edit
Text Based	Edit
⊟ **Def/Null**	
Null allowed	No
Null value	0
Null display	
Null default	No
Default value	0

The Checker Result Pane

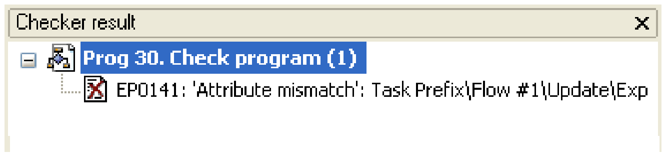
Magic xpa enables you to check an object. The latest check results are displayed in the Checker Result pane.

To display the latest checker result you select the **Checker Result** option from the **View** menu.

The image displayed below is an example from an existing program. Later in this course, you will be able to access similar Checker Result panes in your own program.

The Checker Result pane is displayed (by default) under the work area.

The results are displayed in a data tree format. The levels of the tree can be sorted by the object, message type, or by both the object and the message type.



Prog 30. Check program (1)

EP0141: 'Attribute mismatch': Task Prefix\Flow #1\Update\Exp

The Comment Pane

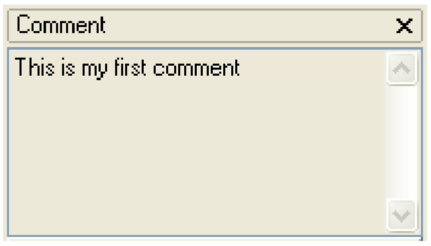
Magic xpa enables you to attach a comment to an object in your project.

You can attach a comment to each of the following object types: models, columns, variables, data sources, indexes, foreign keys, tasks, components, Direct SQL, events, handlers, helps, ranges, rights, and I/O devices.

Attaching a Comment to an Object

To attach a comment to an object, such as a program:

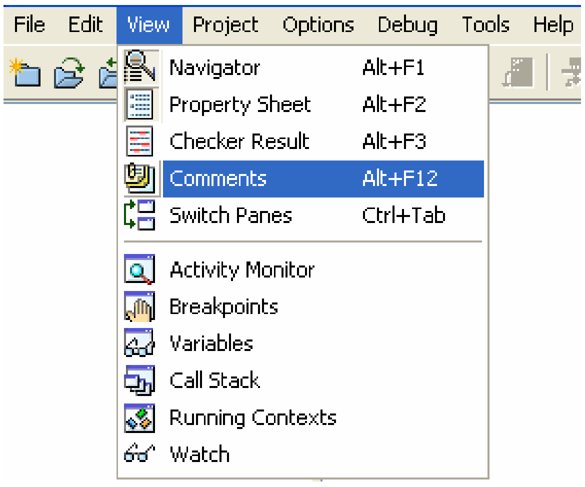
1. Open the Program repository.
2. Park on the Main Program.
3. From the **Options** menu, select **Comment** (F12).
4. Type a comment in the displayed box.
5. Click **OK** to close the **Comment** box.



This is my first comment

Viewing Comments in the Comment Pane

The Comment pane displays the comments for a selected object.



To display an object's comments:

1. From the **View** menu, select **Comments** (Alt+F12).
2. Park on the Main Program (from the example above).

Summary

In this lesson you were introduced to the Magic xpa Studio interface.

You learned about the Navigator pane options and how to navigate between the main repositories.

You learned how to open the property sheet for a selected object.

You also learned how to display an object's comments using the Comment pane.

In addition, you learned how to view the latest checker results.

Lesson 2

Creating Your First Program

Magic xpa supports project-based development. In this lesson you will learn more about the project concept and you will create your first program.

By the end of this lesson you will be:

- Familiar with the project concept.
- Able to create simple programs.
- Able to define Virtual variables.
- Able to add variables to a form.
- Able to execute a program.

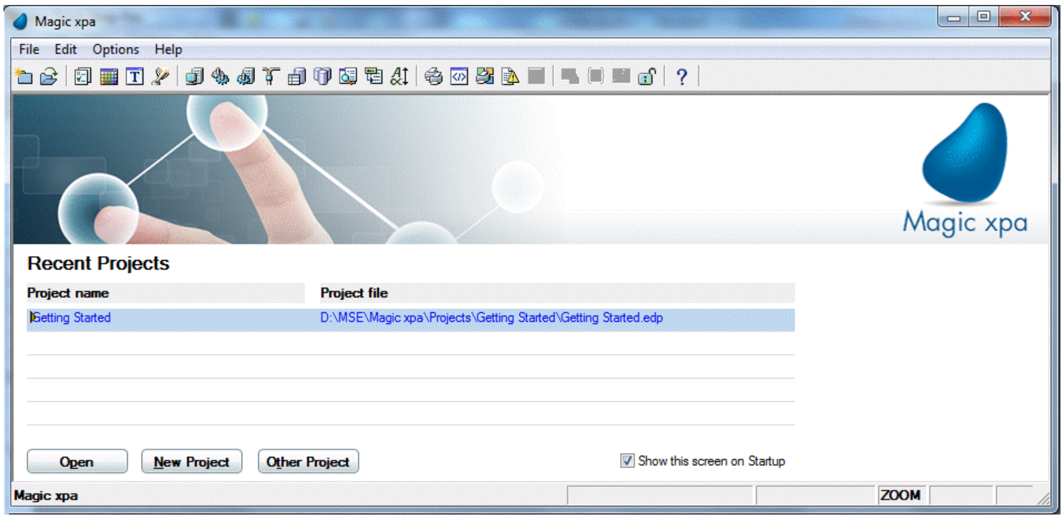
A program is the fundamental part of creating a project. Writing a program is the basis for any development tool or language.

In this lesson you will create your first program. Then, you will execute the program to view the results on your device.

Opening the Getting Started Project

In the previous lesson you created the **Getting Started** project. Now you will open the project and create a program in it.

- In the Startup screen, click the **Open** button.



Recent Projects

Project name	Project file
Getting Started	D:\MSE\Magic xpa\Projects\Getting Started\Getting Started.edp

The Project Source Files

When you create a new project, the Magic xpa Studio engine creates a new directory with the name of the project, in the location that you specify in the **New Project** dialog box. All of the project files are saved in the new directory.

The project files include:

- A main project file, which has the same name as the project name and an **.edp** suffix.
- A **Source** subdirectory, which contains all of the project's source files.
- An **Exports** subdirectory for future export files.

Viewing the Project Source Files

Go to the **[Magic xpa path]\Projects\Getting Started** directory and see that the new project file, **Getting Started.edp**, and the **Source** and **Exports** subdirectories were created.

The Application Properties

Each project has its own properties. You can set the properties for the currently open project. The set properties are saved as part of the project source files.

1. From the **File** menu, select **Application Properties** (Ctrl+Shift+P).

The properties are organized according to three main subjects:

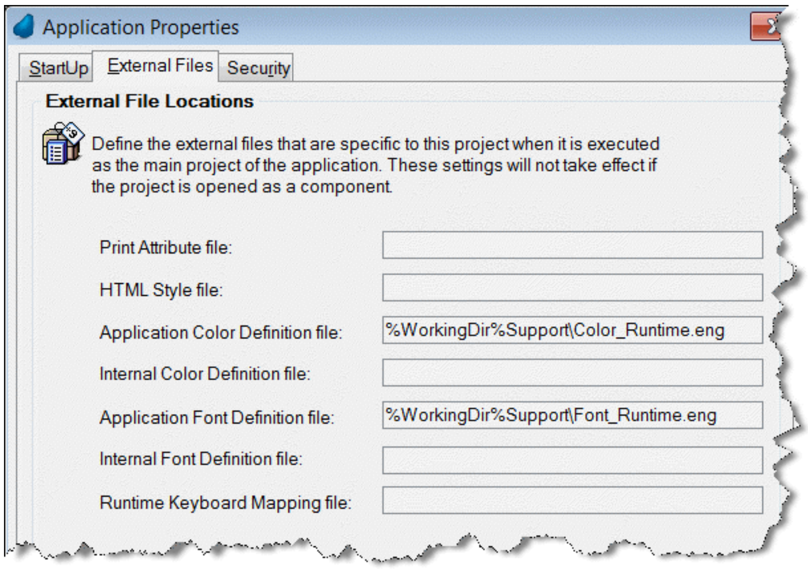
- StartUp
- External Files
- Security

The **StartUp** and **Security** properties will not be discussed in this course. For more information, see the *Magic xpa Help*.

External Files

In Magic xpa, you can define general settings for all of the projects and you can define specific settings for each project. You can do this from the **Options** menu.

2. From the **Options** menu, select **Settings** to open its submenu.



External File Locations

Define the external files that are specific to this project when it is executed as the main project of the application. These settings will not take effect if the project is opened as a component.

Print Attribute file:	
HTML Style file:	
Application Color Definition file:	%WorkingDir%Support\Color_Runtime.eng
Internal Color Definition file:	
Application Font Definition file:	%WorkingDir%Support\Font_Runtime.eng
Internal Font Definition file:	
Runtime Keyboard Mapping file:	

Each of the options in the **Settings** submenu enables you to change the default setting definitions for all of the projects.

For example, you can select the **Colors** option and add new entries to the Color repository. You will learn about this in a later lesson.

However, some options can be set specifically for each project:

- Colors
- Fonts
- Keyboard Mapping
- HTML Styles
- Print Attributes

The options above are external files and you can override the general files by clicking the **External Files** tab in the **Application Properties** dialog box.

The idea is to enable you to provide unique settings for each project.

Creating a Program

In this section you will learn how to create a program in Magic xpa.

The program creation process consists of the following steps:

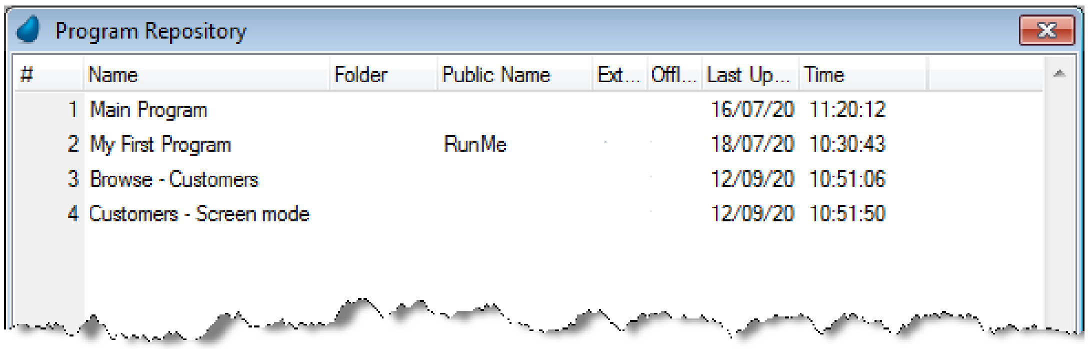
- Creating an entry in the Program repository
- Defining the task properties
- Defining the task data view
- Defining the task logic
- Defining the task form

After the program creation is completed, the next step will be to execute the program and view the results.

The Program Repository

The Program repository contains an entry for each program within the project, starting with the Main Program.

The image below is an example of a Program repository that contains four programs.



Program Repository

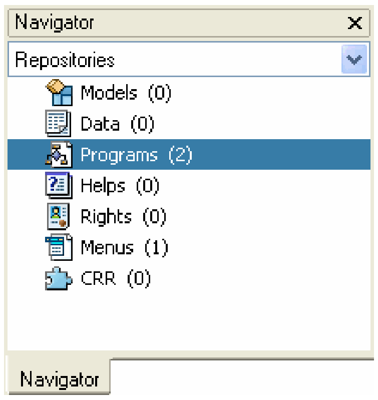
#	Name	Folder	Public Name	Ext...	Offl...	Last Up...	Time
1	Main Program					16/07/20	11:20:12
2	My First Program		RunMe			18/07/20	10:30:43
3	Browse - Customers					12/09/20	10:51:06
4	Customers - Screen mode					12/09/20	10:51:50

The following table describes each column in the Program repository.

Column	Description
#	A program identifier number that is generated automatically by Magic xpa. When you edit the Program repository by adding, deleting rows or moving a program from one place to another, Magic xpa automatically recalculates the number of the programs and updates their identifier numbers. You cannot directly change the program number.
Name	In this column, type in a descriptive name for the program. The program name does not have to be unique and can consist of Alpha and Numeric characters.
Folder	You can group some programs under the same folder. From this entry you can select a folder where you want to place the program entry. **Note:** This column is only accessible when you have created a folder.
Public Name	This column defines the public name of the program. This name must be unique in the project file, and it can be used for calling remote programs or using the program in components.
External	This column allows the program to be exposed for external calls. **Note:** This column is only available when a public name has been defined. This is important for your program to be exposed to calls from mobile devices.
Offline	This property indicates whether the program can run on the server (not connected to the server).
Last Update Date and Time	These columns are automatically updated by Magic xpa when any change to the program is saved. You cannot change the date and time.

Adding a Program in the Program Repository

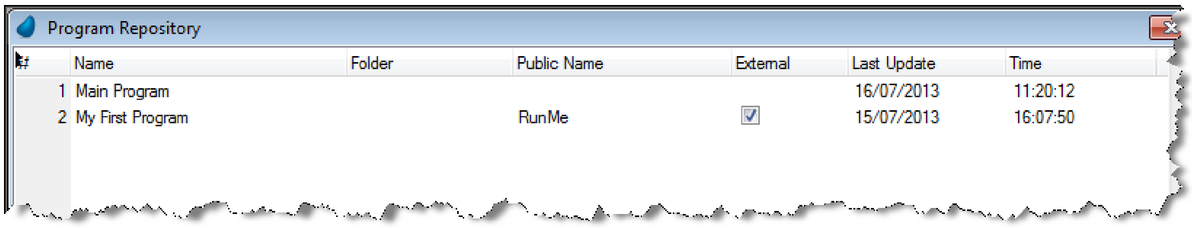
1. From the **Project** menu, select **Programs** (Shift+F3).



2. Park on the first row (**Main Program**).
3. From the **Edit** menu, select **Create Line** (F4). In Magic xpa, **F4** is a shortcut for creating a new line.
4. In the **Name** column of the new row, type: **My First Program**.

In order for this program to run on a mobile device, you need to add an external name. You will learn more about this at a later stage.

5. In the **Public Name** column, type: **RunMe**. It is important to type this exactly as it appears here.
6. Check the **External** check box.


Program Repository

#	Name	Folder	Public Name	External	Last Update	Time
1	Main Program				16/07/2013	11:20:12
2	My First Program		RunMe	✓	15/07/2013	16:07:50

What Is a Task?

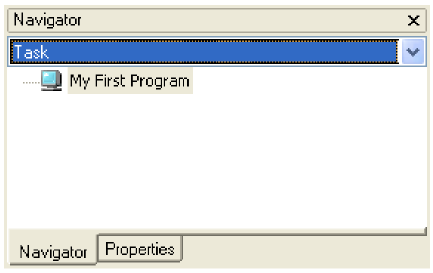
Both of the terms **program** and **task** are used to describe an application task in Magic xpa.

There is a difference between a **program** and **task** in Magic xpa. A task is the basic unit for constructing a program. A program can be constructed of a main task and subtasks.

Actually, if a program consists of one main task only, like in the program you just created, it is equivalent to a task.

1. Park on My First Program.
2. Zoom to the program by pressing **F5**.

The Navigator pane displays the task tree.
In this case, the new program is constructed of only one main task.

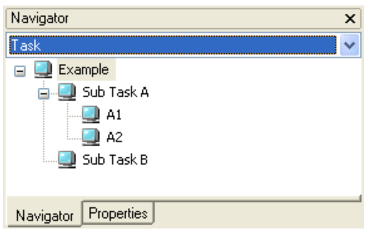


In the image to the right, you can see an example of another task tree:

The **Example** task is the main task.

Sub Task A and **Sub Task B** are subtasks of the **Example** task.

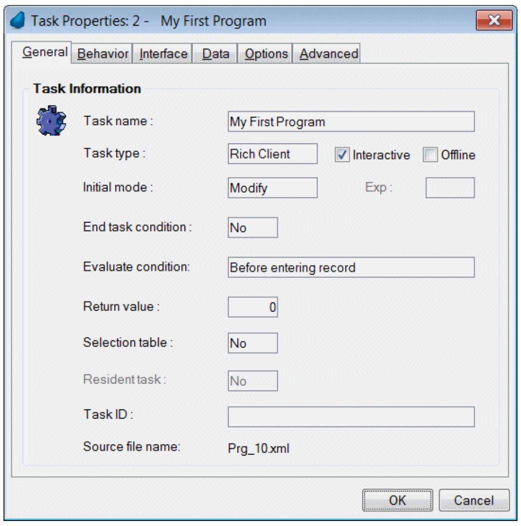
The **A1** task and **A2** task are subtasks of the **Sub Task A** task.



The Task Properties

Each task has its own property definitions.
The task properties include a collection of properties that let you define the task's type, behavior, data, interface, and more advanced features.

3. When you zoom into a task, the **Task Properties** dialog box opens. If it is not open, go to the **Task** menu and select **Task Properties** (Ctrl+P).



Task name :	My First Program
Task type :	Rich Client ✓ Interactive ☐ Offline
Initial mode :	Modify Exp :
End task condition :	No
Evaluate condition:	Before entering record
Return value :	0
Selection table :	No
Resident task :	No
Task ID :	
Source file name:	Prg_10.xml

The **Task name** property is a descriptive name of the task.

The **Task type** property defines the task type. The task type can be:

- **Online (default)**, which means that the task requires user interaction, such as a data entry task.
- **Batch**, which means that the task is an automatic task, not requiring any user interaction.
- **Browser**, which means that the task runs on a Web browser, using the Browser Client methodology.
- **Rich Client**, which means that the task runs as a Rich Client task and will run on mobile devices.

4. Change the **Task Type** to **Rich Client**. This is what makes the program run as a Rich Client program.

The **Initial mode** property defines the mode of operation in which execution of the task starts.

The **Source file name** property is the name of the program's source file as it exists on the disk. Each program is saved in a different file.

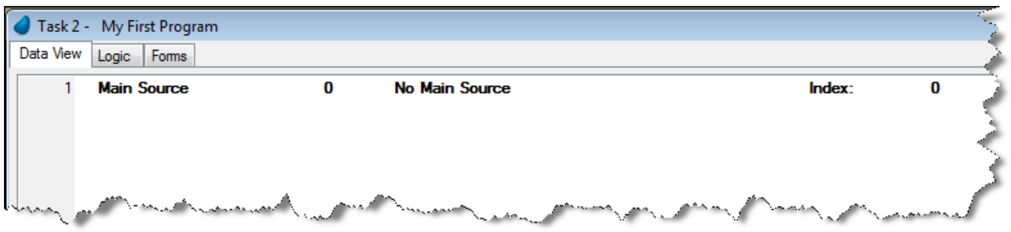
5. Click **OK** to close the **Task Properties** dialog box.

The Task Interface

When you zoom to a task, the task interface is displayed.

The Task interface is composed of three editors:

- Data View
- Logic
- Forms



1 Main Source 0 No Main Source Index: 0

- Press the Ctrl+Tab keys to switch between the editors.
- Press the Ctrl+1 keys or click the **Data View** tab to switch to the Data View Editor.
- Press the Ctrl+2 keys or click the **Logic** tab to switch to the Logic Editor.
- Press the Ctrl+3 keys or click the **Forms** tab to switch to the Form Editor.

The Data View Editor

The Data View Editor enables you to:

- Define the task's Main data source or Direct SQL statement.
- Link to other data sources.
- Define data source columns, Virtual variables, and parameters.
- Declare additional data sources.

The Data View Lines

In the Data View Editor, you can create two types of lines:

- **Header line** – The header line defines the task's data source types and properties. The first header line defines whether the task has a Main data source or a Direct SQL statement.
- **Detail line** – The detail line defines the task's variables. The variables can be data source columns, parameters, or Virtual variables.

In this lesson you will learn how to define Virtual variables only. Later on in this course, you will learn how to define other sources in the data view.

Defining the Task Data View

Now you will define the new program data view, which is composed of Virtual variables.

A **Virtual variable** is a local variable that is used for computation and temporary storage. The Virtual variable exists only for the program duration, so its value is not saved after the program execution is terminated. This means that in each execution, the values of the variables return to their default values.

There are two other kinds of variables in Magic xpa:

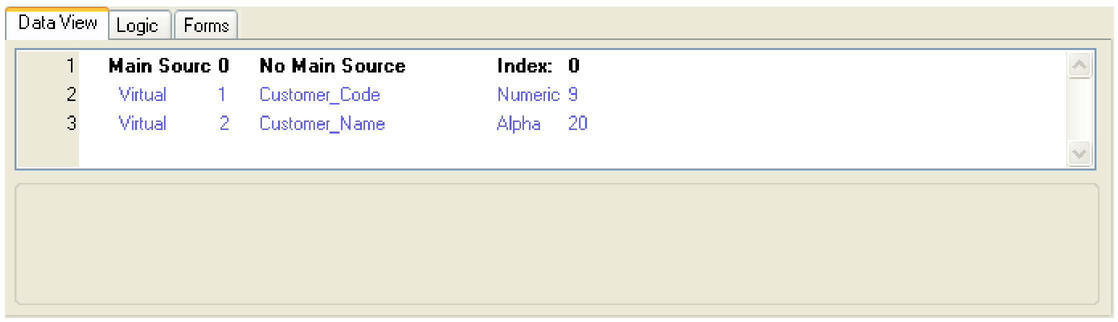
- **Parameter** – Variables which receive values from a different program
- **Column** – Data object columns defined within a database table

You will learn more about these other variables later in this course.

Creating Virtual Variables

6. Verify that you are parked on the Data View Editor.
7. Create a line (either by selecting from the **Edit** menu or pressing the **F4** shortcut).
8. Set the first line as shown in the table below.
9. Create another line and set the information as per the second line of the table below.

Variable Type	Variable Name	Attribute	Picture
Virtual	Customer_Code	Numeric	9
Virtual	Customer_Name	Alpha	20



```
1   Main Sourc 0   No Main Source      Index:  0
2   Virtual    1   Customer_Code       Numeric 9
3   Virtual    2   Customer_Name       Alpha   20
```

Although it might seem like it, the **Virtual variables** are **not** children of the **Main Source** header line. Later in this course, you will learn how to select a **Main Source** and Main Source columns.

The Task Forms

Most projects allow for user interaction. A screen is one way to enable interaction. A screen is a form that is displayed during the task's execution. Other types of forms are used to output reports, files, and so on.

A form displays information to the end user. Magic xpa supports various types of forms. This lesson will focus on the main form, the **Rich Client Display** screen.

Each task has a main form as the default and you can add additional forms.

For interactive tasks such as Online and Rich Client tasks, you use the form to display the task's data view and to allow end-user interaction.

For non-interactive tasks such as Batch and Browser tasks, you can use the form to display data regarding the task progress while it is executed.

The Form Editor contains form definitions for a task. Each entry represents a form.

When you define a new task, Magic xpa automatically creates an entry in the Form Editor, which is the task's main window. The name of the form is taken from the name of the task. You can edit its name.

The task's main form cannot be deleted or moved to another position in the repository.

Mobile Design Mode

When the Mobile Design mode is enabled and controls are added to the form, some of the properties will be set with different default values, which better support the mobile platforms.

10. Enable this feature by clicking the **Mobile Design Mode** button from the toolbar or by opening the **Options** menu and selecting the **Mobile Design Mode** option.

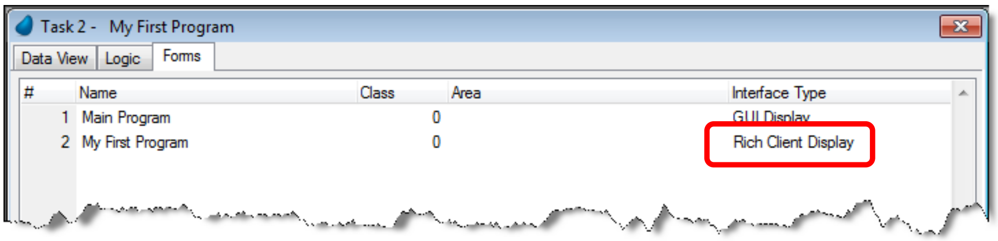
Main Form

Now you will define the task's main form.

11. Park on the Form Editor. Because this is a Rich Client task, the main form will default to **Rich Client Display**. This is the form type used for Rich Client programs.

In the Form Editor you can see all of the forms that are hierarchically above the task that you are in. This is why the **Main Program** form is displayed.

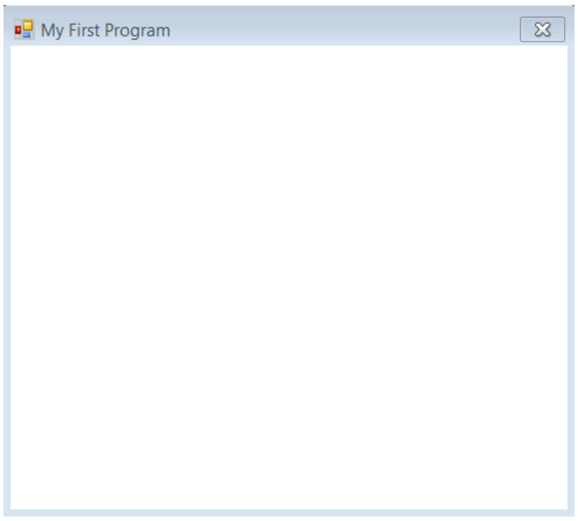
12. Park on the **My First Program** form entry and **zoom** (**F5**) or double click.



#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	My First Program	0		Rich Client Display

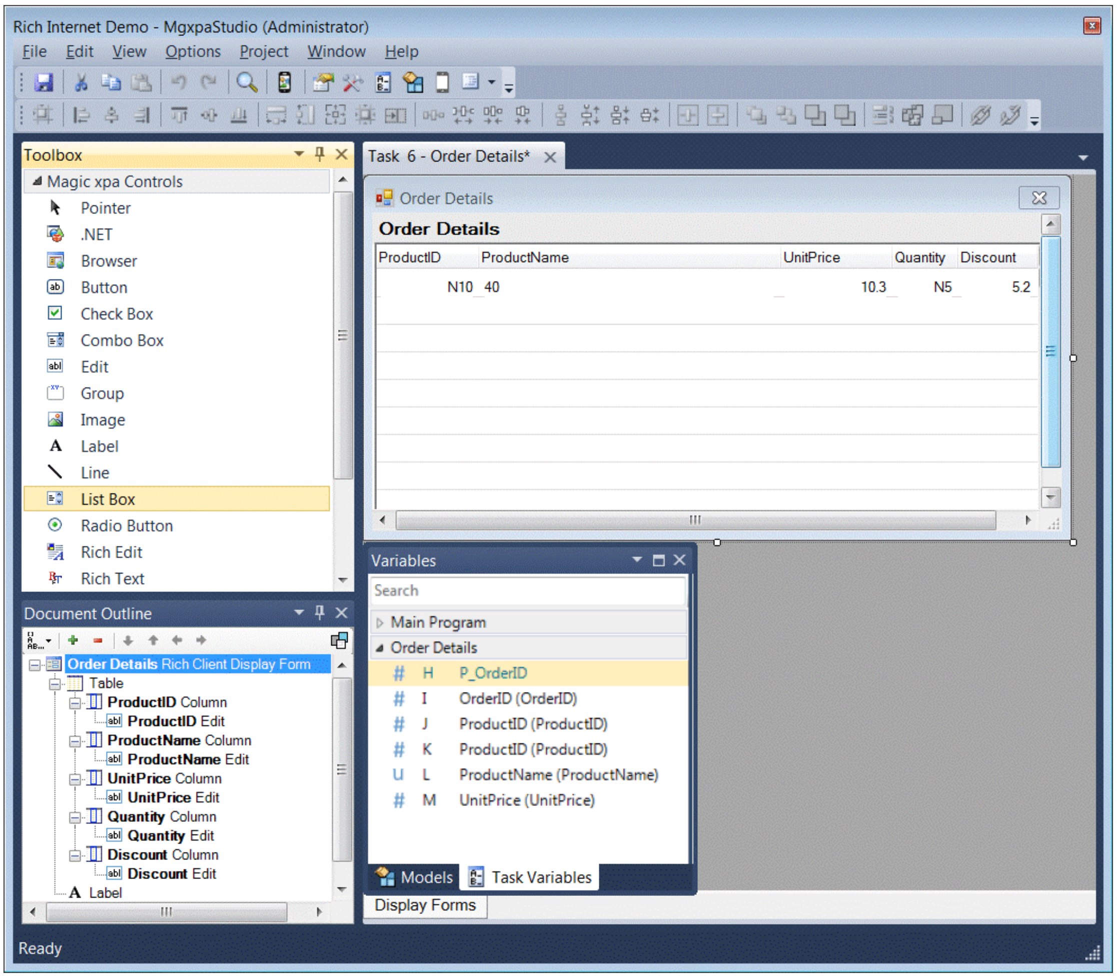
By default, the Mobile Design Mode prepares a small screen to fit the dimensions of a mobile phone screen. During this course, you will be developing for a tablet.

13. Therefore, increase the size of the screen by dragging it, so that it looks similar to the one below.



14. Now, switch off the Mobile Design Mode.

The Form Designer



The Form Designer includes:

- A toolbar with the control commands – You can use the commands to align the controls' location, size, and so on.
- The controls' Toolbox pane – This shows you the controls that you can add to the form.
- The Document Outline pane – This shows you the controls that exist on the form. You can use it to easily move controls from one container to the other.
- The Variables pane and Models pane. This shows you all of the variables and models in your data view.

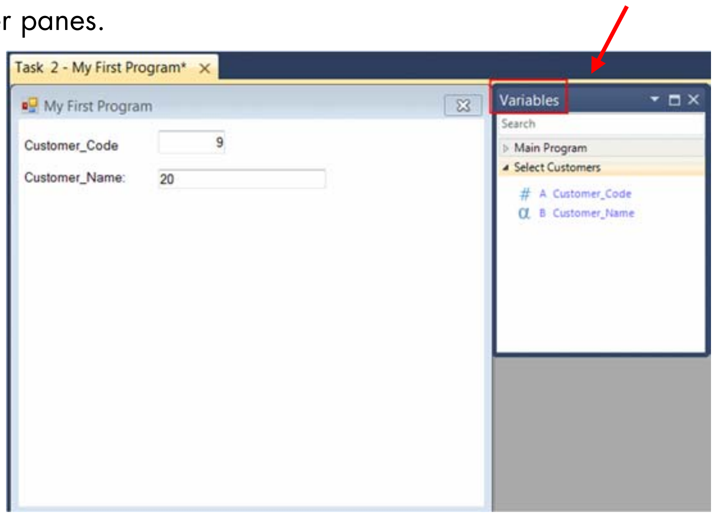
You will use these panes while editing forms in Magic xpa. The same options are also available on the overhead menu and via shortcut keys. It is good to get familiar with the options and the shortcut keys.

If for some reason the panes are not showing in the workspace, select **View** menu (also available as an overhead icon).

Adding Variables to the Form

Now you will add the **Customer_Code** and **Customer_Name** variables to the task's main form using the Form Designer panes.

15. From the **Variables** pane, drag the **Customer_Code** variable, and drop it onto the form), as shown in the image on the right.
16. Repeat the last step (2) for the **Customer_Name** variable.

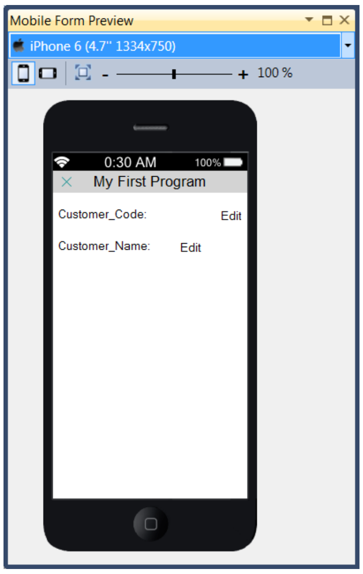


When you drag and drop a variable from the Variables pane, Magic xpa automatically adds a Label control, which displays the variable name next to the variable's Edit control.

Mobile Form Preview

When developing for mobile platforms, you can preview the Display forms that you are developing by using the **Mobile Form Preview** pane.

17. Clicking the mobile button from the toolbar to open this pane. You can also access it from the **Form** menu.



This lets you play around with the placement and size of the controls and see how the controls will appear on various mobile devices.

From the drop-down list at the top of the pane you can select which mobile device you want to preview, such as iPhone 6. Next to the name of the device, you will see the dimensions of the device. You can also change the orientation of the preview and the zooming.

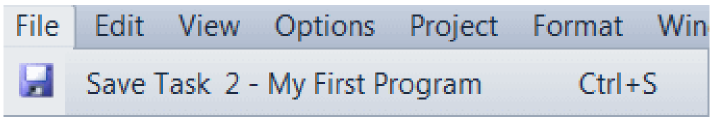
Note that the **Mobile Form Preview** pane is not currently supported for Windows 10 Mobile devices.

Saving the Program

You have created a new program and defined its data view and form.

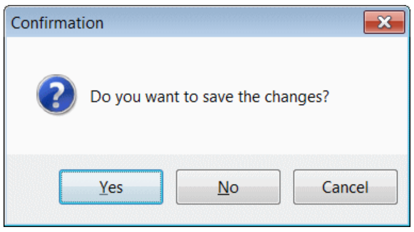
Now you will save and exit your first program:

18. From the **File** menu, select the **Save** option or click the **Save** button to save the changes.



19. Close the form by clicking the ☒ icon at the top right of the screen or by pressing the **Esc** button.

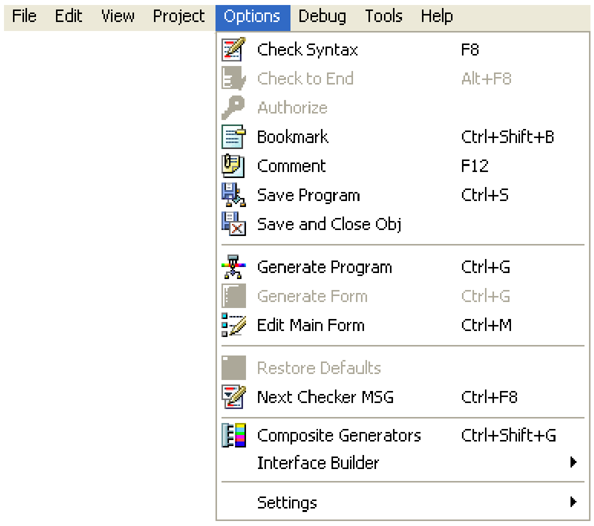
If you try to exit the program (by pressing the **ESC** key), a confirmation box is displayed to let you confirm or cancel the changes.



- Click the **Yes** button to confirm the changes and exit the program.
- Click the **No** button to reject the changes and exit the program.
- Click the **Cancel** button to stay in the program.

Alternative Ways to Save

When the Form Designer is closed, from the **Options** menu, you can select the **Save Program** option (Ctrl+S) to save the changes up to that point ***without*** leaving the program.



Changes to the Data View, Logic, or Form editors are regarded as changes to the program itself. When you save or cancel modifications from within one of the editors, changes in the other editors will also be saved or cancelled.

Checking the Program

1. Park on the **My First Program** program in the Program repository.
2. From the **Options** menu, select the **Check Syntax** option or press **F8.**

If there are no syntax mistakes in your program, the message **Program is OK** will appear in the status bar.

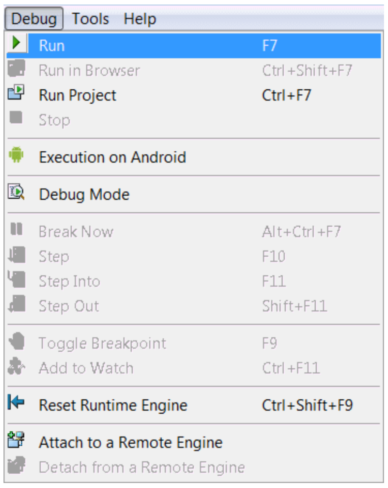
If there are mistakes, the errors are listed in the Checker Result pane.

After completing any program it is advisable to press **F8** to check for errors.

Running the Program

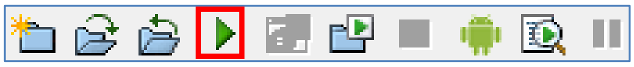
Now you will execute the program from the Studio environment.

1. Park on the **My First Program** program in the Program repository.
2. From the **Debug** menu, select **Run** (F7).



The Magic xpa Runtime window is opened and the executed program's main form is displayed.

To execute a program, you can press **F7** or select the Run icon from the toolbar:



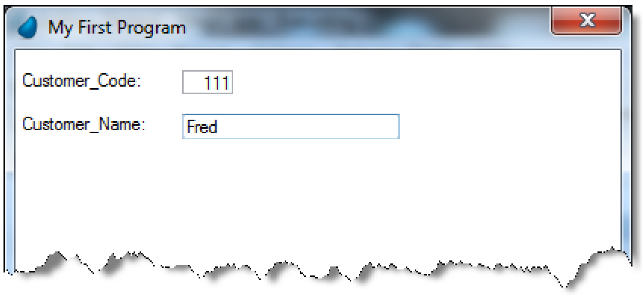
The Studio engine automatically loads the Runtime engine when it initially loads. However, this process is not visible to the end user until a request is made from the Studio to run something, such as pressing **F7** to execute a program. Then, the Runtime engine is visible and active.

The Virtual variables that you created will be displayed and the task is now available for end-user input.

1. In the **Customer_Code** variable, type the number **111**.
2. Press the Tab key to move to the **Customer_Name** variable.
3. In the **Customer_Name** variable, type **Fred**.

You have just typed in values in the **Customer_Code** and **Customer_Name** Virtual variables. Since Virtual variables exist only for the duration of the program, when you exit the program the Virtual variable values are cleared.

4. Close the program and execute it again to see that the Virtual variable values have been cleared.



Customer_Code: 111
Customer_Name: Fred

Built-in Task Behavior

This section provides you with two examples that highlight Magic xpa's built-in task behavior. Later on in this course, you will learn more about this internal behavior.

Internal Data Validation – Numeric Variable Example

Magic xpa handles data entry validation internally, according to the variable attribute.

In this example, the **Customer_Code** is a Numeric variable. Magic xpa verifies that only numbers (and the negative/positive sign) are typed in this variable.

Picture Limitation Example

Magic xpa provides internal handling for Picture limitation validation, which is done according to the variable picture.

In this example, the **Customer_Code** variable picture is limited to 9 characters. Magic xpa will verify that only 9 numeric characters (and the negative/positive sign) are accepted in this variable.
The **Customer_Name** variable picture is limited to 20 characters. Magic xpa will verify that only 20 characters are accepted in this variable.

For example:

1. Type a 10-digit number in the **Customer_Code** variable.
   Magic xpa will accept the first 9 digits and ignore the rest.
2. Type more than 20 characters in the **Customer_Name** variable.
   Magic xpa will accept the first 20 characters and ignore the rest.

Executing on Your Mobile Device

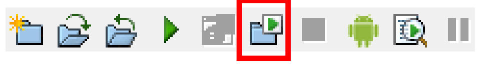
To execute the application on your mobile device, you need to first install the Magic client app on your device. We will use a generic client. This is for development purposes only. In a later lesson, you will learn how to customize your own client.

For Android devices, you can install the **MagicDev.APK** application, which is available in the **RIAModules\Android** folder.

For iOS devices, download the Magic xpa client from the Apple App Store.

Before you run the Magic xpa client on your device, your Magic xpa application needs to be running:

1. Open the Magic xpa Studio.
2. From the **Options** menu, select **Settings** and then **Environment**.
3. In the **System** tab, set the **Deployment Mode** environment setting to **Background**.
4. Open the application.
5. Execute the Magic xpa application by pressing **CTRL+F7** or by selecting the **Executes the project** icon on the toolbar.



You also need to define the connection details for this server:

6. Copy the **DevProps.txt** file from the **RIAModules\Android**, **RIAModules\iOS** or **RIAModules\Windows 10 Mobile** folder to the **scripts** folder.
7. Open the file in a text editor and set all of the details, such as the server IP, program name and the application name.

When you run the Magic xpa client on your device, a dialog box will open asking you to provide the path to the execution properties file. You will learn more about the execution properties file in a later lesson.

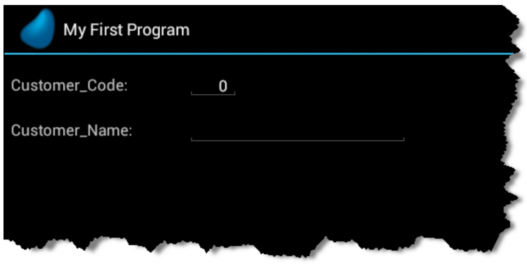
8. Run the Magic xpa client on your device.
9. Type **http://<your ip address>/MagicScripts/DevProps.txt**.
   MagicScripts is the default alias created during the installation. If you changed the alias in the installation, then you should also change it here.

You will then see your program running on your mobile device.

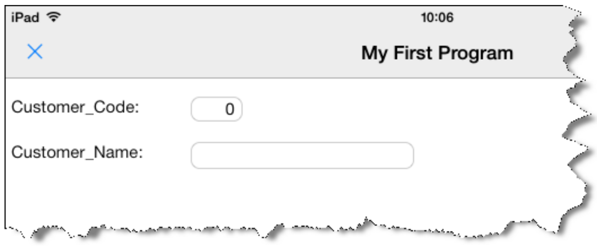
Throughout this course you will see a number of side-by-side comparison images for Android and iOS devices. Windows 10 Mobile is also available as of Magic xpa 3.2.

The program will look similar to the images below:

Android:



iOS:



The same program was executed on your desktop computer as well as on your mobile device with no changes to the program. As you can see, the same program has a different look and feel on different operating systems. It also has the native look and feel of each specific device.

When you park on the **Customer_Code** control, the device will automatically display a numeric keyboard. You will not be able to type any text even if you access the ABC keyboard.

When developing for Android devices, you can connect your Android device to your computer and then click the Android toggle button on the toolbar 🤖. You can then run the project or the program and it will be displayed automatically on your device. For information on how to setup this up, please look at the appendix.

About Magic xpa Attributes

In the last example, you created two variables. You specified various properties that help Magic xpa define the variable. One of these properties is the variable attribute (in some databases it is referred to as **type**). The attribute defines the characteristics of the variable. The following table describes some of the attributes that Magic xpa supports:

Attribute Type	Description
Alpha	Alpha is an attribute that allows the storing of alphanumeric characters. In the Alpha attribute, Numeric characters are stored as a string. The Alpha attribute is the default attribute.
Unicode	Unicode is an attribute that allows the storing of alphanumeric characters in Unicode format.
Numeric	Numeric is an attribute that allows the storing of an integer or a decimal number. You can store numeric values with up to 18 digits. When you define a Numeric type, your device will initially open a numeric keyboard for your user to enter digits.
Logical	Logical is an attribute that Magic xpa stores internally as a single byte with the 0 value (False) or the 1 value (True). Use the Logical attribute when you are storing Boolean values, such as: True or False and Yes or No.
Date	When you park on a Date control, your device will automatically display a Date Picker for you to select the date from a list. Each device displays a different date picker control. Date is an attribute that is stored by Magic xpa internally as Numeric or as a String. The numeric Date attribute is a counter of days since 01/01/01 or since 01/01/1901, depending on its storage field model. The fact that a Date attribute is stored as a Numeric value lets Magic xpa perform calculations to create new date values. A Date attribute is translated to its visible value only when it is displayed in Magic xpa.
Time	When you park on a Time control, your device will automatically display a Time Picker for you to select the time from a list. Each device displays a different Time picker control. Time is an attribute that Magic xpa stores internally as a counter of seconds, starting from midnight. You can use a Time attribute to represent either a duration of time or an absolute time value. Just as with Date attributes, Time attributes can be subtracted from one another, and values can be added to them or subtracted from them. A Time attribute is translated to its visible value only when it is displayed in Magic xpa.

About the Magic xpa Picture

The picture properties can be set using a number that specifies how many characters the specific variable will accept.
Magic xpa provides even more functionality for the picture properties, enabling you to determine more than just the number of characters.

In Magic xpa, the **Picture** property is a string that defines the variable length format.

The variable length is set using a number or mask (specific characters that function as directives).

For example:

> To configure a variable that will hold only 6 digits, you can specify its **Picture** in one of the following methods:

- The number six (6)
- The number sign (#) repeated six times: **######**
- The number sign only once and the number six (the 6 means that the number will have six digits): **#6**

To set a picture with a mask, zoom from the **Picture** property to a Picture dialog box, which is different for each attribute.

Magic xpa will translate the settings in the **Picture** dialog box to directives in the variable picture.

Each Magic xpa attribute has its own set of picture directives.

Exercise

Now that you are familiar with additional Magic xpa attributes, you will add more variables to the program.

1. Add the following variables in **My First Program** and display them on the form:

Variable Type	Variable Description	Attribute	Picture
Virtual	Country	Alpha	20
Virtual	City	Alpha	20
Virtual	Address	Alpha	20
Virtual	Gold Membership	Logical	5
Virtual	Membership_Date	Date	##/##/####
Virtual	Membership_Time	Time	HH:MM:SS
Virtual	Salary_Amount	Numeric	12.2C
Virtual	Credit_Amount	Numeric	12.2C

2. Execute the program.
3. Set the **Gold Membership** to **True**.
4. Park on the **Membership_Date** control and select the date **15th March, 1997**.
5. Park on the **Membership_Time** control and select the time **3:15PM**.

Note: You will learn more about the picture in the next lesson.

Summary

In this lesson you created your first program in Magic xpa.

You learned about the project concept.

You also learned about the Program repository and how to create a program in the repository.

You learned how to define a program and some of its properties.

In addition, you learned how to define a task data view that includes local variables.

You were introduced to Magic xpa attributes and pictures.

Now, you are familiar with the program concept and you are ready to move on to the next lesson.

Lesson 3

Architecture Overview

During the previous lesson you created a simple Rich Client program in Magic xpa and executed it and you saw it running both on your desktop machine and your mobile device. Developing applications in this manner is known as developing Rich Internet Applications, RIA.

Working with Rich Client programs means that you are working with distributed application architecture. It can be helpful to understand what is happening behind the scenes, so that you can learn how to get the best performance.

This lesson covers:

- The RIA architecture
- Automatic logic partitioning
- Transparent context management
- Execution details

Mobile Deployment

Magic xpa Application Platform provides the tools to build core business applications that are suited to the mobile environment, providing device-independent enterprise mobility in a single unified interface.

Magic xpa applications can run natively on ssmartphones by using a native client module to implement a single unified business logic repository that supports both Apple iOS and Google Android.

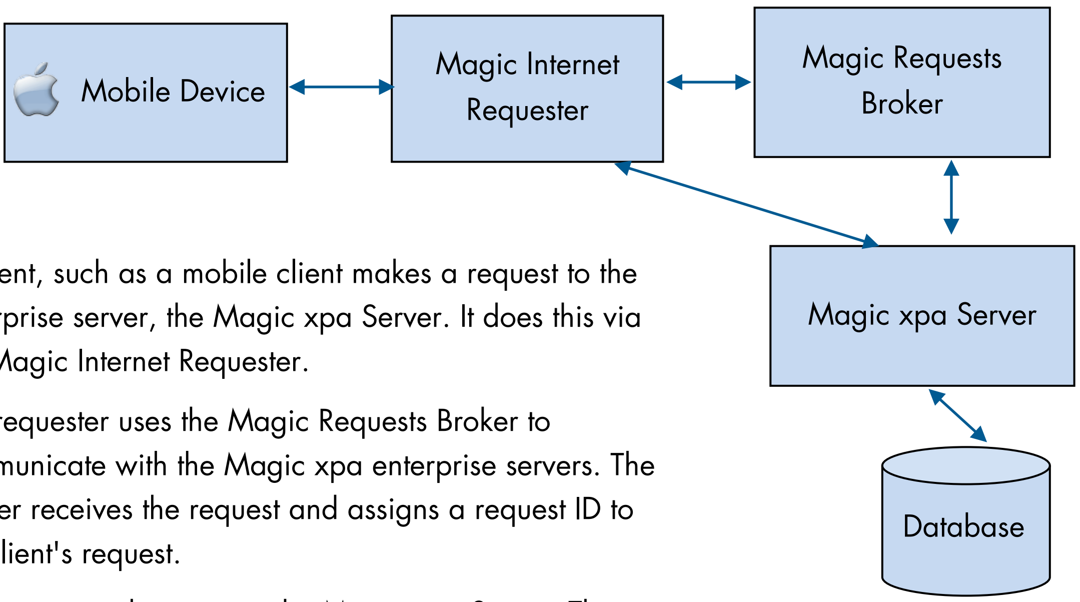
The RIA Architecture

In the previous lesson you executed the application on your desktop machine and then ran the client on your smartphone. The program was then displayed on your smartphone. Part of your application runs on your desktop machine and part of it runs on the smartphone. This RIA architecture consists of the following modules:

- **Magic xpa Server** – The core process that serves the remote clients. This is a Magic xpa Runtime engine (MgxpaRuntime.exe) functioning as an enterprise server.
- **Web Server** – A standard Web server used to handle the remote requests.
- **Magic Internet Requester** – An extension of the Web server that is used to connect the client modules to the Magic xpa server and vice versa.
- **Magic Requests Broker** – The middleware governing the flow of communication between the Magic Internet Requester and the Magic xpa Server. The broker is in charge of the load balancing by which each request is directed to the most suitable Magic xpa server process. The broker is also responsible for executing the fault tolerance policy and the capacity surge policy.
- **GigaSpaces Middleware** – GigaSpaces' XAP in-memory computing technology is the middleware that implements Magic xpa's functionality on the In Memory Data Grid. The GigaSpaces middleware is used in the deployment environment instead of the Broker.
- **Magic xpa RIA Client Module** – A module that runs on your smart device and serves as the front-end of the application.

In this course we will use the Broker as the middleware.

The image below displays Magic xpa's distributed application architecture when using the Magic Requests Broker:



A client, such as a mobile client makes a request to the enterprise server, the Magic xpa Server. It does this via the Magic Internet Requester.

The requester uses the Magic Requests Broker to communicate with the Magic xpa enterprise servers. The broker receives the request and assigns a request ID to the client's request.

The request is then sent to the Magic xpa Server. The Magic Internet Requester now works directly with the Magic xpa Server.

Automatic Logic Partitioning

Once a Rich Internet Application is deployed, the Magic xpa platform automatically partitions the logic of the application according to the nature of each operation that the platform is about to execute. While running on the client, the client will automatically turn to the server whenever it encounters a server-side only operation. And while on the server side, the application will turn to the client whenever a client-side only operation is encountered or whenever the server-side activity is completed. The automatic partitioning is the fundamental feature of the platform, which enables the defining of logic without breaking the logic into client/server/communication modules. You will learn more about this in a later lesson.

Transparent Context Management

A Magic xpa RIA developer does not need to define or maintain user sessions or user contexts. The Magic xpa platform automatically identifies the context of each request and implicitly maintains the context to follow the current session of the end user.

The developer can define and utilize context specific information and further enhance the personalization of the execution with a set of functions.

Execution Details

In previous lessons you were asked to create the application with the name **Getting Started** and to define a program with a **Public name** of **RunMe**. These are defined in a text file under the server's Scripts alias. This file is known as the execution properties file. This file contains properties that assist the communication between the client and the server, such as which program the client wants to run. You will learn more about this file in a later lesson and you will learn how to customize it.

Summary

The Rich Client architecture or Rich Internet Application involves a distributed architecture in which part of the code runs on the client, the smartphone, and part of it runs on the server.

Magic xpa takes care of the connection between the various modules including the context management. As a developer you only need to define the programming logic.

Lesson 4

Data Manipulation and Validation

One of the objectives of any program is the ability to transform data from one format to another. This includes working with numeric calculations and string manipulations.

First you will learn how to use Magic xpa arithmetic operators to perform simple arithmetic operations on your data.

You will learn more about the Logic Editor and how to use operations.

You will then learn how to set a condition on an operation and you will be introduced to the Magic xpa **IF** function. You will learn how to concatenate strings and how to trim spaces from strings.

As in any interactive program, there is a need to validate the end-user data entry in your program. You will also learn how you, as a developer, can validate end-user data entry and how you can maintain data consistency.

This lesson covers various topics including:

- Logic units
- Operations
- Calculations and conditions
- Variable Change logic unit
- Allow Parking property

Numeric Data Manipulation

Magic xpa enables you to manipulate numeric data and display the results. You can use the simple arithmetic operators, such as addition and subtraction, and you can also use the various Magic xpa internal Numeric functions. You will learn about some of these functions later on in the course. The following is a list of basic arithmetic operators that you can use for numeric values:

Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
^	Exponentiation operator (A^B returns A to the power of B)
MOD	Modulus operator Modulus returns the remainder of an integer division. For example: 17 MOD 10 returns 7

In this section you will learn how to update a variable with the result of a simple arithmetic calculation.

Logic Editor

You added variables in the Data View Editor and created the form in the Form Editor.

Now you will add logic to the program using the Logic Editor.

The Logic Editor enables you to define all of the task's logical segments within a single editor, where:

- The header line represents a logic unit or a handler. You will learn more about events and handlers in later lessons.
- The detail line represents an operation.

From the **Edit** menu, you can create a header line by clicking **Create Header Line** (CTRL+H) or a detail line by clicking **Create Line** (F4).

Magic xpa internal logic units, for example the **Control Suffix**, enable you to enter a section or unit of logic pertaining to that logic unit. In this case, you will add a series of operations that will occur during the Control Suffix stage. This means that they will occur when you leave or exit the control. You will have a better understanding of this by the end of this lesson.

The logic that you want to add is that the credit for this person (**Credit_Amount**) is double their salary (**Salary_Amount**). So if you modify the salary amount, you want to automatically modify the credit.

Operations

You create a **Header line** to define events within the Logic Editor. Within each header, you have **Detail lines** that make up that specific logic unit.

The Detail lines are made up of internal Magic xpa operations. There are nine high-level operations that you can use in the Logic Editor. You can customize these operations by adding expressions and using Magic xpa's functions, internal events, and so on. You can select an operation by selecting the operation from the combo box or by entering the operation accelerator key (as seen in the combo box). The operations are listed below:

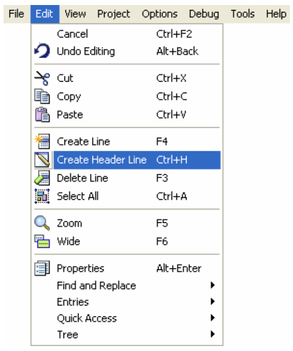
- **Remark** (default) – Enables you to enter a description of the section of code.
- **Update** – Enables you to modify or set the value of a variable.
- **Call** – Enables you to call a secondary program or other entities. You will learn more about this later in this course.
- **Invoke** – Enables you to call an external object, such as an operating system command or a Web Service.
- **Raise Event** – Enables you to raise an event. This will be discussed in a later lesson.

- **Evaluate** – Enables you to enter an expression where the return value is True or False, but you do not need to know whether the function succeeded.
- **Block** – Enables you to enter a set of operations with a single condition. There are two types: **Block If-Else** and **Block While**.
- **Verify** – Enables you to set a warning or an error as a result of a certain expression.
- **Form** – Enables you to either export data (to a printer or to a text file) or to import data.

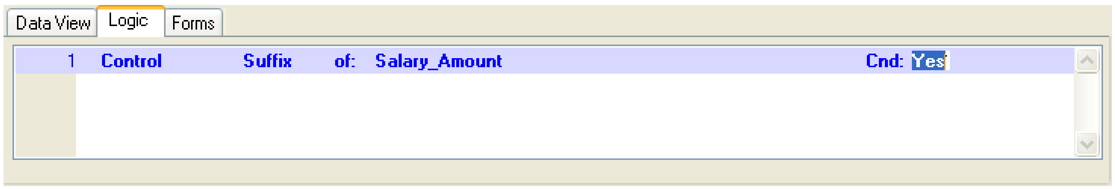
All operations defined for a single Header line are part of that logic unit. The Detail lines are indented under the Header line, as an indication that the operation is part of the designated logic unit.

In this example, you want to **update** the credit for a person where the **Credit_Amount** will be double their salary (**Salary_Amount**). So you will use the **Update** operation within the Control Suffix logic unit.

The following steps will guide you through calculating the **Credit_Amount** variable based on the **Salary_Amount**.



1. From the Program repository, zoom into **My First Program**.
2. Switch to the **Logic Editor**.
3. Create a Header line. You can do this by selecting the **Create Header Line** option from the **Edit** menu or by pressing **Ctrl+H**.
4. Define a **Control Suffix** operation.
5. Zoom from the **of:** field and the Control list opens. This is known as a selection list. In Magic xpa, you can zoom to the internal selection lists. Within the list, you park on the item you want to use in your program and either click the **Select** button or double-click the entry.
6. Park on the **Salary_Amount** control.
7. Click the **Select** button.



```
1   Control        Suffix    of:  Salary_Amount                      Cnd: Yes
```

Expression Editor

Now you need to create the following expression: **Salary_Amount * 2**. For this you will use the integrated Expression Editor.

The Expression Editor enables you to enter and display expressions.

An expression can be:

- A constant, such as a specific number, a word, or a specific variable taken from the data view.
- A formula for computing a value that consists of a sequence of operators indicating the action to be performed and operands on which the operation is performed.

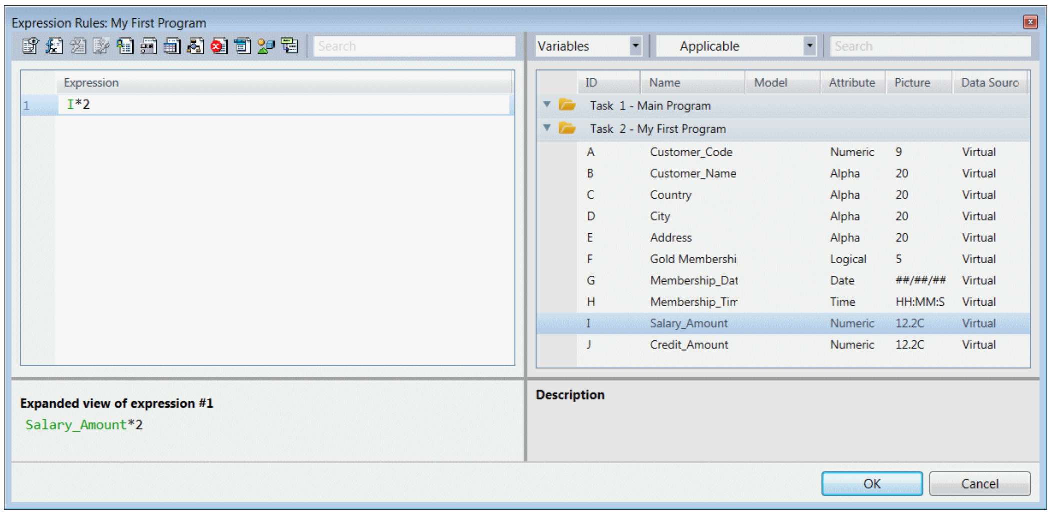
You can enter any arithmetic expression that follows conventional algebraic rules, a string, or a logical expression.

Each entry in the Expression Editor evaluates to any of the attributes that were discussed in the previous lesson such as a numeric value, an Alpha string, BLOB, or a logical value (TRUE or FALSE), depending on the type of variable and functions used. If an invalid combination of data types is used, Magic xpa will display an error message.

When you right-click in the Expression Editor, an extensive context menu will appear, where you can access the various Expression lists.

Magic xpa has many useful internal functions. You will use some of them during this course.
You will now use the Expression Editor to add the functionality.

1 Park on the **Control Suffix** logic unit line and add a line.
2. Define an **Update Variable** operation.
3. Zoom to select the **Credit_Amount** variable from the Variable list.
4 Zoom from the **With** property to the **Expression Editor**.
5. Create an expression line by pressing **F4**.
6. Use zoom (or click on the list) to move to the variable selection list on the right-hand side, so that you can select a variable.
7 Select the **Salary_Amount** variable.
8. In the expression line type: ***2**
   The expression should be: **I*2**


Expression Rules: My First Program

	Expression
1	I*2

Variables | Applicable

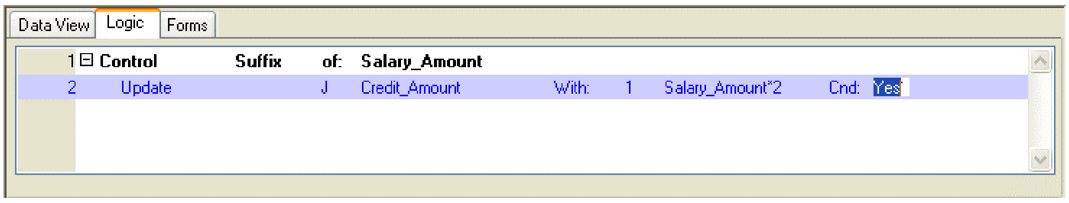
	ID	Name	Model	Attribute	Picture	Data Source
Task 1 - Main Program						
Task 2 - My First Program						
	A	Customer_Code		Numeric	9	Virtual
	B	Customer_Name		Alpha	20	Virtual
	C	Country		Alpha	20	Virtual
	D	City		Alpha	20	Virtual
	E	Address		Alpha	20	Virtual
	F	Gold Membershi		Logical	5	Virtual
	G	Membership_Dat		Date	##/##/##	Virtual
	H	Membership_Tim		Time	HH:MM:S	Virtual
	I	Salary_Amount		Numeric	12.2C	Virtual
	J	Credit_Amount		Numeric	12.2C	Virtual

Expanded view of expression #1
Salary_Amount*2

Description

9. Click **OK** to select the expression and exit the Expression Editor.

Note that the **With** column displays the expression number from the Expression Editor as well as the expression value.


Data View | Logic | Forms

```
1 ⊟ Control        Suffix    of:  Salary_Amount
2    Update                  J    Credit_Amount      With:   1   Salary_Amount*2      Cnd:  Yes
```

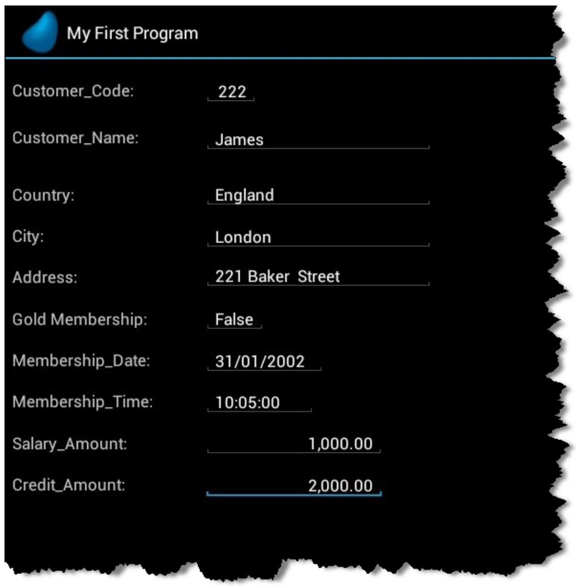
10. Save and close the program.

Now you will execute the program so that you can see the results.

11. Execute the project by pressing **CTRL+F7** and then open the Magic xpa client on your device.
12. Tap the **Salary_Amount** control. (You do not have to fill in all of the details that are shown in the image below.)
13. In the **Salary_Amount** control, type **1000** and then tap the next control.

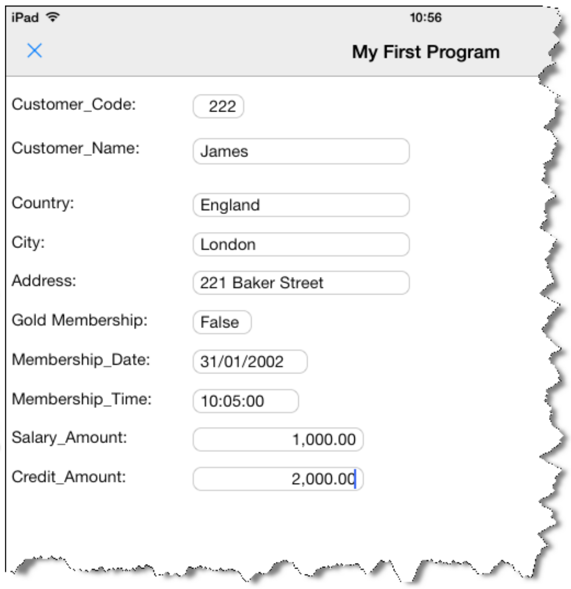
As you can see in the images below, when you leave the **Salary_Amount** control, the **Credit_Amount** value is updated by the **Salary_Amount** value, multiplied by **2**.

Android


My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2,000.00

iOS



My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2,000.00

Explaining the Results

The Control Suffix logic unit is defined for a specific control. In this example it was defined for the **Salary_Amount** control.

The Control Suffix logic unit is executed immediately when the end user leaves the control.

Within the logic unit, you used the **Update** operation, which enabled you to update a variable value with an expression.

You created an expression in the Expression Editor and used the arithmetic operator (*****) to multiply the **Salary_Amount** value by **2**.

In the same way, you can use all of the other arithmetic operators that were mentioned in the beginning of this lesson.

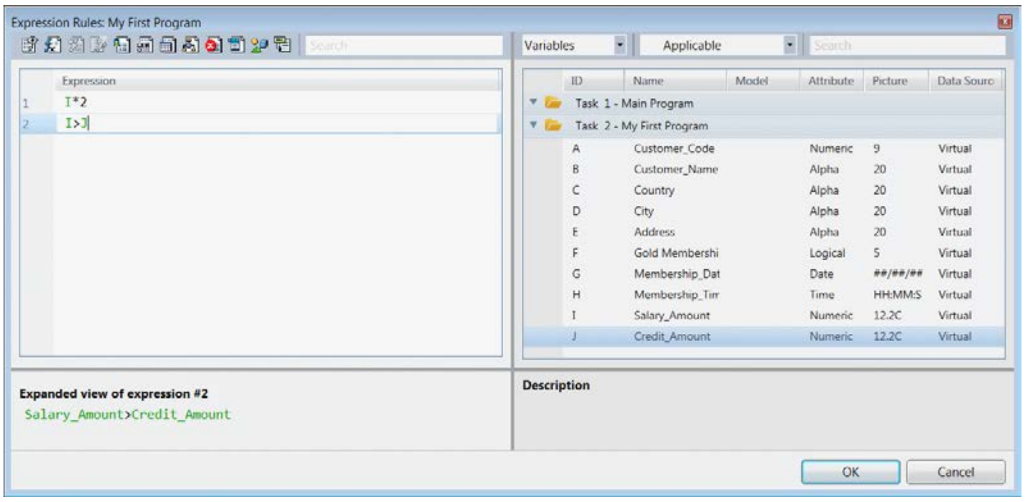
Conditional Calculations

Magic xpa allows you to condition the execution of an operation. The condition is a logic expression, which returns True or False.

Now you will add a condition to the **Update** operation, so that the **Credit_Amount** variable will be updated by the **Salary_Amount*2**, but only when the **Salary_Amount** value is greater than the **Credit_Amount** value.

1. Zoom to **My First Program**.
2. Switch to the Logic Editor.

3. Park on the **Update** operation line.
4. From the **Cnd** column, zoom to the Expression Editor.
5. Create an expression line.
6. Enter the following expression **Salary_Amount > Credit_Amount** (the expression will be **I>J**).
   Note: You can zoom to select a variable from the Variable list to be used within the expression.



Expression Rules: My First Program

	Expression
1	I*2
2	I>J

Variables · Applicable

ID	Name	Model	Attribute	Picture	Data Sourc
Task 1 - Main Program					
Task 2 - My First Program					
A	Customer_Code		Numeric	9	Virtual
B	Customer_Name		Alpha	20	Virtual
C	Country		Alpha	20	Virtual
D	City		Alpha	20	Virtual
E	Address		Alpha	20	Virtual
F	Gold Membershi		Logical	5	Virtual
G	Membership_Dat		Date	##/##/##	Virtual
H	Membership_Tim		Time	HH:MM:S	Virtual
I	Salary_Amount		Numeric	12.2C	Virtual
J	Credit_Amount		Numeric	12.2C	Virtual

Expanded view of expression #2

```
Salary_Amount>Credit_Amount
```

Description

7. Save and close the program.
8. Execute the project by pressing **CTRL+F7** and then open the Magic xpa client on your device.
9. In the **Salary_Amount** control, type: **100**.
10. Tap any other control.

You will see that the **Credit_Amount** value was updated to **200**, since the condition evaluated to **True.** The **Salary_Amount** value (**100**) was greater than the **Credit_Amount** value (**0**).

Now, try a scenario in which the condition is not met.

11. In the **Salary_Amount** control, type **150**. Remember that the amount in the **Credit_Amount** control is set to **200**.
12. Tap any other control.

You will see that the **Credit_Amount** value was not updated, since the condition evaluated to **False**.

The **Salary_Amount** value (**150**) was less than the **Credit_Amount** value (**200**).

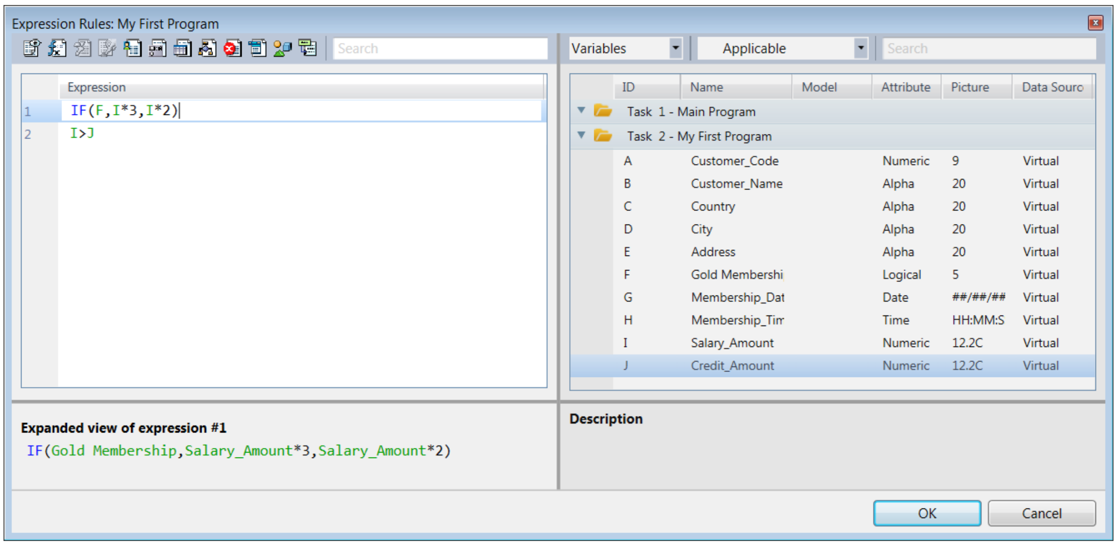
In the previous section you learned how to set a condition for an operation. In some cases you need to control the returned value as well as the operation execution. In these cases, you need to condition the result.

The Internal "IF" Function

When you need to condition an operation, you can use the **IF** function. This function evaluates a Boolean expression and returns one value if True and another value if False. In the following example you will use the **IF** function to control the multiplication factor (from the last example), so that if the customer has a gold membership, the customer receives better credit (the multiplication factor will be 3 instead of 2).

1. Zoom to **My First Program**.
2. Switch to the Logic Editor.
3. Park on the **Update** operation line.
4. From the **With** column, zoom to the Expression Editor.
5. Change the expression as follows: **IF(F,I*3,I*2)**
   Where:
   F is the **Gold_Membership** variable and
   I is the **Salary_Amount** variable.

   This expression evaluates to:
   IF (Gold Membership is True, then multiply Salary_Amount by 3, else multiply Salary_Amount by 2).


Expression Rules: My First Program

	Expression
1	IF(F,I*3,I*2)
2	I>J

Variables | Applicable

ID	Name	Model	Attribute	Picture	Data Sourc
Task 1 - Main Program					
Task 2 - My First Program					
A	Customer_Code		Numeric	9	Virtual
B	Customer_Name		Alpha	20	Virtual
C	Country		Alpha	20	Virtual
D	City		Alpha	20	Virtual
E	Address		Alpha	20	Virtual
F	Gold Membershi		Logical	5	Virtual
G	Membership_Dat		Date	##/##/##	Virtual
H	Membership_Tir		Time	HH:MM:S	Virtual
I	Salary_Amount		Numeric	12.2C	Virtual
J	Credit_Amount		Numeric	12.2C	Virtual

Expanded view of expression #1

```
IF(Gold Membership,Salary_Amount*3,Salary_Amount*2)
```

Description

When using a Logical variable, it is not necessary to use the expression **F=True** meaning **Gold Membership=True**. It is adequate to simply use the variable. In this case, you could write **F** (instead of F=True), which is the variable **Gold Membership**.

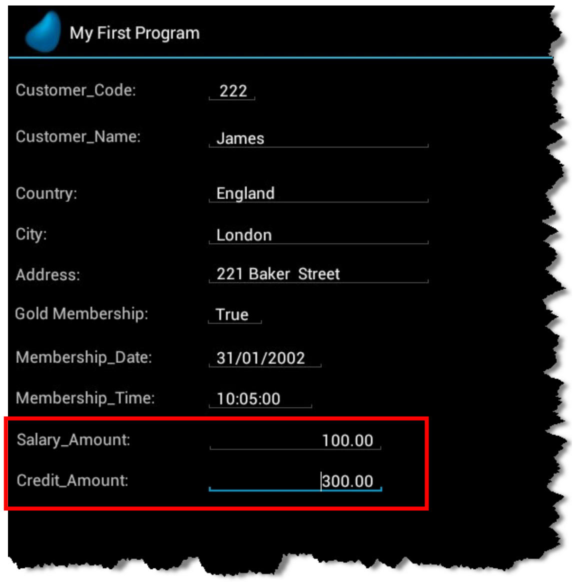
6. Save and close the program.
7. Execute the project by pressing **CTRL+F7** and then open the Magic xpa client on your device.
8. In the **Gold_Membership** control, type: **True**.

Once you type the first letter of the words True or False (T/F) in Logical variables, Magic xpa will automatically complete the rest of the word once you move off the field.

9. In the **Salary_Amount** control, type **100** and press the TAB key.

As you can see in the images below, the **Credit_Amount** value was updated with **300**, since the customer has a **Gold_Membership**.

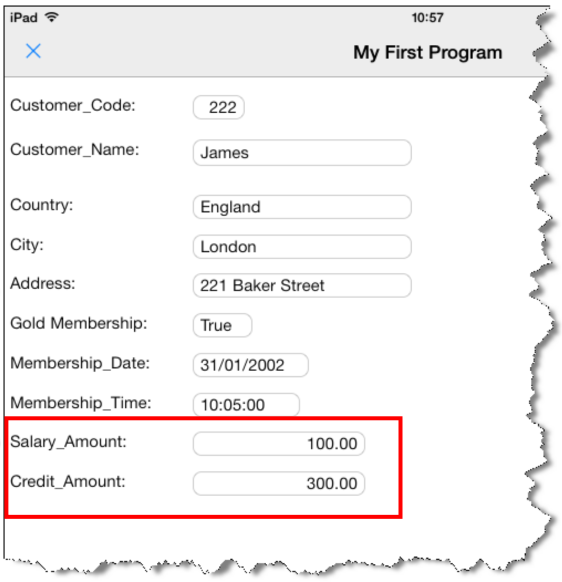
Android



My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	True
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	100.00
Credit_Amount:	300.00

iOS



My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	True
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	100.00
Credit_Amount:	300.00

Now you will try a different scenario in which the customer does not have a gold membership.

10. In the **Gold_Membership** control, type **False**.
11. In the **Salary_Amount** control, type **2000** and press the **TAB** key.

The **Credit_Amount** value was updated with **4000**, since the customer does not have a **Gold_Membership**.

Don't forget that the Update operation condition (Salary Amount > Credit Amount) still exists. Make sure that the Salary Amount is larger than the Credit Amount.

Alphanumeric Data Manipulation

Up until now you have learned and practiced Numeric value manipulations. Magic xpa also allows you to manipulate alphanumeric data.

In this section you will learn how to use the string concatenation operator to concatenate several strings.

The ampersand (&) operator is used in Magic xpa to concatenate Alpha strings.

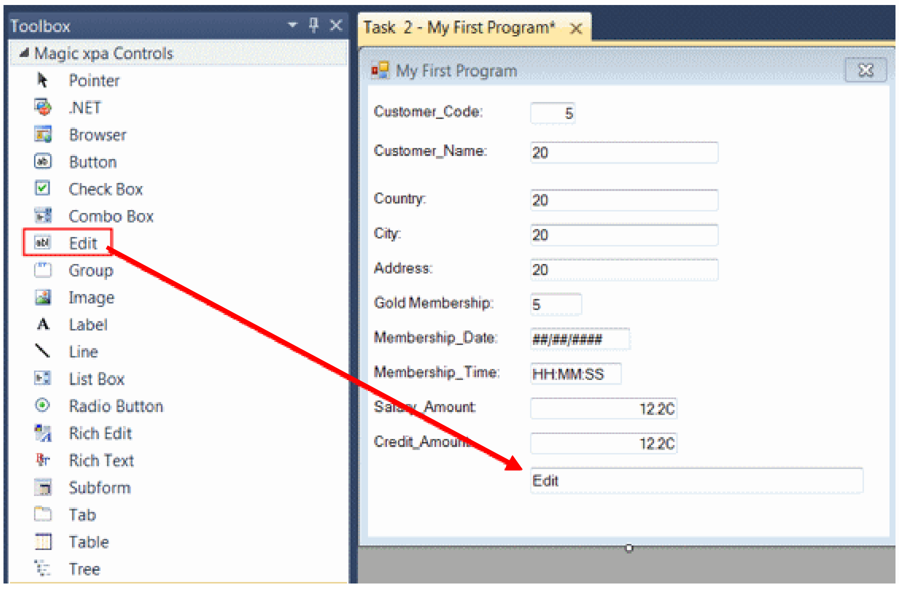
In this section you will learn about the string concatenation operator and the **Trim** function. You will learn additional information about these functions later on in the course.

In this example you will display a customer's complete address, which includes a concatenation of the customer's address, city, and country.

1. Zoom to **My First Program**.
2. Switch to the Form Editor and zoom to the **My First Program** form.

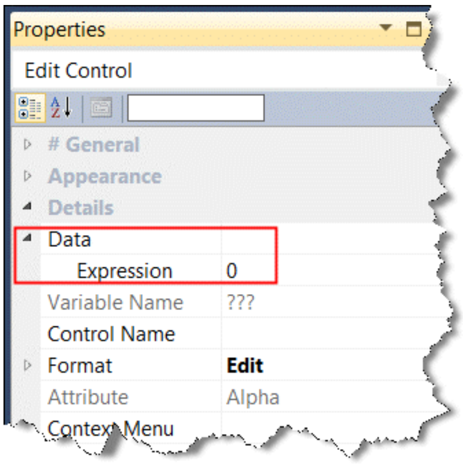
Now you will add an Edit control to the form to display the complete address string.

3. Increase the size of the form by dragging the bottom of the form.
4. Drag the **Edit** control icon from the Toolbox and place it on the form as shown in the image below.



Park on the new Edit control and open the Control Properties sheet (**Alt+Enter**) if it's not already open. In Magic xpa, the keyboard combination of Alt+Enter will access a property sheet. You can also access the same sheet by selecting the **Properties** option in the context menu for that specific item.

5. Expand the **Data** property and zoom from the **Expression** line to assign an expression for the control.



Data	
Expression	0
Variable Name	???
Control Name	
Format	**Edit**
Attribute	Alpha
Context Menu	

The Edit control can be assigned to a variable or to an expression. In previous sections you added Edit controls that were assigned to variables. Now, you will be adding an Edit control that will be assigned to an expression.

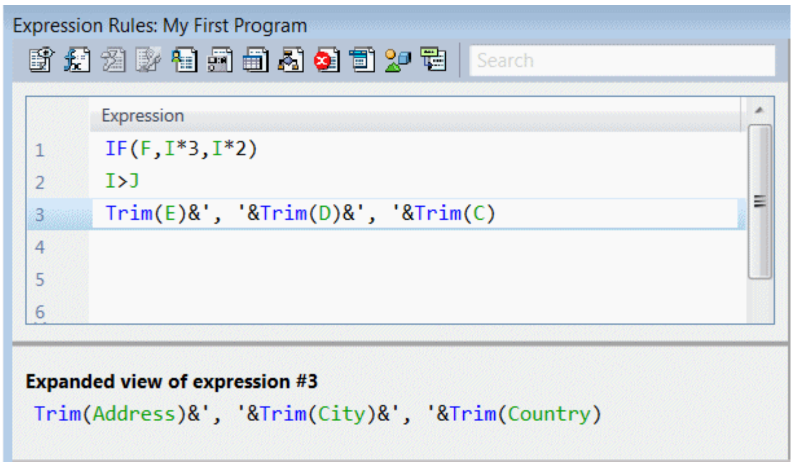
6. The Expression Editor opens.

You will now use the **Trim** function. This function removes trailing or preceding blanks from a string.

Syntax	**Trim (*string*)**
Parameters	*string*: An Alpha string to be trimmed. You can type a string between single straight quotation marks (') or type the letter of an Alpha variable.
Returns	The function returns the string without the leading and trailing blanks.
Example	Trim(' hello ') returns the string: 'hello' without leading and trailing spaces.
Note	The apostrophes will not appear in the result.

7. Create an expression line.
8. Type the following expression: **Trim(E)&', '&Trim(D)&', '&Trim(C)**
   Where:
   E is the **Address** variable.
   D is the **City** variable.
   C is the **Country** variable.
   The string **', ' adds** a comma separator.

In this example you are concatenating various strings together while at the same time you are removing the extra blanks.



```
IF(F,I*3,I*2)
I>J
Trim(E)&', '&Trim(D)&', '&Trim(C)
```

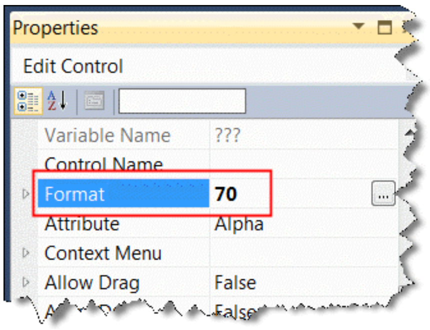
Expanded view of expression #3

```
Trim(Address)&', '&Trim(City)&', '&Trim(Country)
```

9. Click **OK** to select the expression for the **Data** property.

The Edit control that you placed on the form should contain information from three variables. Each variable may contain up to 20 characters. To display all of the information, including the three variables' content, and the spaces and commas between the variables, you need to change the Edit control format to 70.

10. In the Edit Control properties' **Format** property, type **70**.

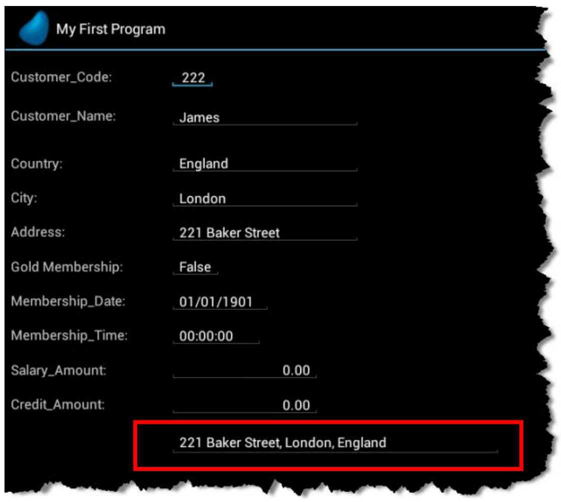

Properties
Edit Control

Variable Name	???
Control Name	
Format	70
Attribute	Alpha
Context Menu	
Allow Drag	False

11. Close the program and save the changes.
12. Execute the project by pressing **CTRL+F7** and then open the Magic xpa client on your device.
13. Type in the same customer details, as you did before.

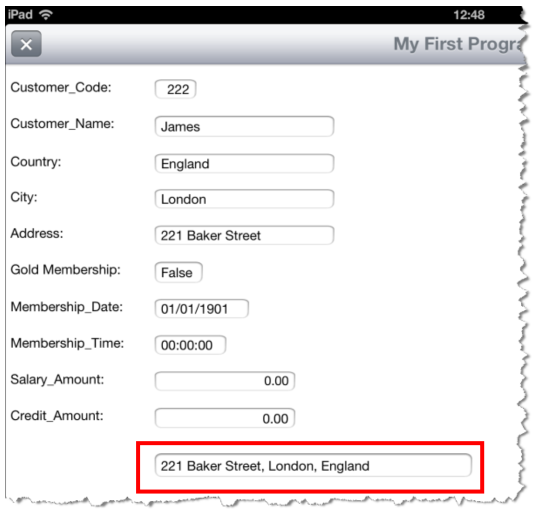
As you can see, the new Edit control displays the complete customer address, which contains the address, city, and country separated by commas.

Android


My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	01/01/1901
Membership_Time:	00:00:00
Salary_Amount:	0.00
Credit_Amount:	0.00
	221 Baker Street, London, England

iOS



My First Progra

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	01/01/1901
Membership_Time:	00:00:00
Salary_Amount:	0.00
Credit_Amount:	0.00
	221 Baker Street, London, England

Magic xpa Internal Data Validation

In a previous lesson you were introduced to two examples that presented part of Magic xpa's internal data validation mechanism. For example, you saw that an end user cannot type Alpha characters into a Numeric variable. In Magic xpa, the variable attribute, in this case the Numeric variable, prevents the end user from entering unacceptable data.

The variable picture is a more specific way to define the data type and format of the variable. For example, a variable with an Alpha attribute can contain Alpha and Numeric characters. Using the **Picture** property, you can specify the exact positions within the variable that can contain Alpha characters and the ones that can contain Numeric characters. The following table lists examples of variable definitions and the allowed values that can be used.

Attribute	Picture	Description
Numeric	N5	A 5-digit number between -99999 to +99999. The N directive allows the number to be negative.
Alpha	##UXX	A 5-character string where the first two positions must be Numeric characters, the next position an upper case character and the last two positions can be any character. The # directive within the picture means that the data in that position can only be Numeric. The U directive within the picture means that Magic xpa will automatically change the value to an upper case value. **Note:** In Android systems, auto capitalization is a setting on each keyboard. Some keyboards may have this property set to off. Example: 78Mag
	###-#######	A common way to define phone number variables. The variable contains three numbers, which are the area code, a separation character (-), and then the phone number.

Attribute	Picture	Description
Date	##/##/####	This format is the default format for a date. You can also use DD/MM/YYYY. The DD directives mean that the first two positions can only be a number from 01 to the number of the days in the specified month. The MM directives mean that the end user can only enter a number between 01 to 12, representing the month. The YYYY directives mean that the end user can only enter 4-digit numbers, representing the year in full format.
Logical	5	A 5-character string that can only be True or False. This variable format means that the end user can only enter True or False. Other values will not be accepted.
Time	HH:MM:SS	This is the default picture for defining a time variable.

This section provides you with a number of examples that demonstrate the Magic xpa internal data validation mechanism.

Numeric Attribute Validation

As you learned in a previous lesson, when you have a Numeric field, the numeric keyboard is displayed. When you try to enter a character in a Numeric field, the input is rejected. As an example:

1. Execute the project and in the **Customer_Code** field, type a character, such as: **a**.

In a later lesson you will learn how to add a status bar so that you can pass information to the end user. The end user will then receive an "Invalid number" error message.

Alpha Attribute Validation

Magic xpa lets you enter any character in an Alpha variable.

The variable picture also defines the length of the variable. In this case, the name can contain only 20 characters, since Magic xpa constrains the length of the string to 20 characters.

2. In the **Customer_Name** field, type: **Sherlock lives in 221A Baker Street**.

You will see that you will not be able to enter the entire text.

Logical Attribute Validation

Magic xpa does not allow you to enter anything but **True** or **False** in Logical variables. If you enter anything else, you will not be able to continue and you will receive an error message.

3. In the **Gold_Membership** field, type: **aaa**.
4. Tap another field. You will receive an error and in the text of the error you will see the valid values that you can enter in this field.

Developer Validation

Up until now the Magic xpa internal validation mechanism was discussed. This section will explain how you as a developer can validate end-user data entry.

In this example you will ensure that the end user only enters dates that are <u>not</u> later than the current date.

1. Zoom to **My First Program**.
2. Switch to the Logic Editor.
3. From the **Edit** menu, select **Create Header Line** (**Ctrl+H**) to create a Header line for the **Control Suffix** logic unit. Define this Header line for the **Membership_Date** control.

For this example you will use the **Verify** operation. The Verify operation displays an Error or Warning message whenever Magic xpa executes the operation and its condition evaluates to True. The message will be displayed in an error box.

The Verify operation's Mode property provides the following options:

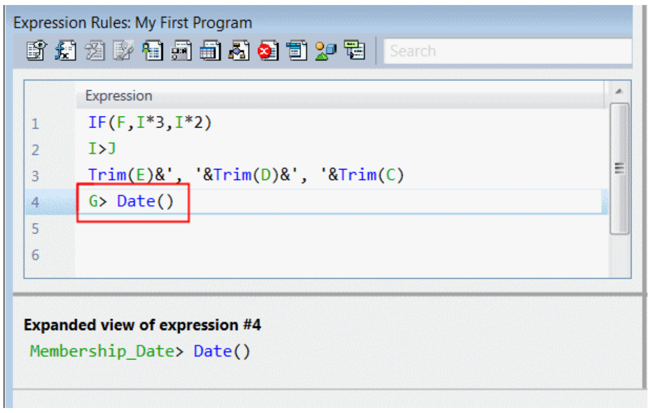
- **Warning** (default) – Magic xpa beeps and warns the end user by displaying the specified message, but does not stop the program execution. The user can ignore the message, click the OK button in the message box, and continue.
- **Error** – Magic xpa beeps and alerts the end user by displaying the specified message. The end user cannot ignore the message. The end user must correct the input according to the constraints in order to continue the program.
- **Revert** (for advanced usage) – An error message is displayed. The cursor returns to the handled control by executing the operations that precede the Verify operation in reverse order back to the first operation in the logic unit.

4. Park on the **Control Suffix** logic unit line and create a Detail line.
5. Define a **Verify Error** operation.

6. In the property sheet, go to the **Text** property and type: **The date is later than the current date**.

The **Display** property enables you to display the error message in an error box or in the form's status line. On a mobile device, Magic xpa does not display a status line and therefore the error message will always be displayed in a message box.

7. Park on the **Cnd** property and zoom to the Expression Editor by pressing **F5**.
8. Create a line in the Expression Editor by pressing **F4**. The order that expressions appear in the editor have no relevance to the program.
9. Select the **Membership_Date** variable. This should be the variable **G** assuming that you have not added any of your own variables.
10. Complete the expression as follows:
    G> Date().
11. Click **OK**.



```
IF(F,I*3,I*2)
I>J
Trim(E)&', '&Trim(D)&', '&Trim(C)
G> Date()
```

Expanded view of expression #4

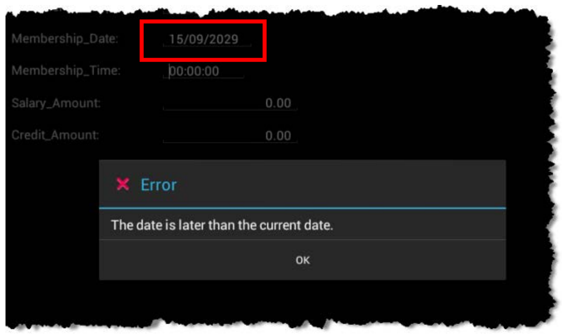
```
Membership_Date> Date()
```

Now you will execute the program to view the results.

12. Execute the project by pressing **CTRL+F7** and then open the Magic xpa client on your device.
13. In the **Membership_Date** field, select a later day than the current date, such as 15/09/2029.
14. Move to a different field.

As you can see, when the condition of the Verify operation is met (the typed date is later than the current date), an error message is displayed. After you confirm the error message, by tapping **OK**, Magic xpa parks on the **Membership_Date** field, enabling you to set a different value.

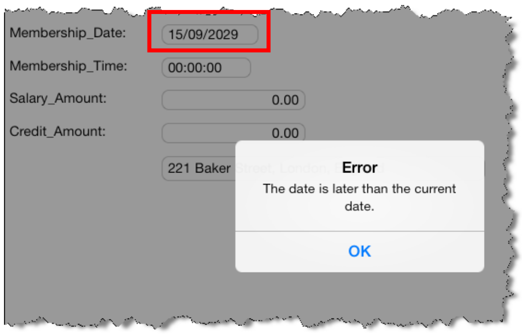
Android:



Membership_Date: 15/09/2029
Membership_Time: 00:00:00
Salary_Amount: 0.00
Credit_Amount: 0.00

Error
The date is later than the current date.

iOS:



Membership_Date: 15/09/2029
Membership_Time: 00:00:00
Salary_Amount: 0.00
Credit_Amount: 0.00
221 Baker

Error
The date is later than the current date.

You must enter a date that meets the condition you entered to continue, or exit the program.

Data Consistency

During project development, there are common scenarios in which you need to maintain data consistency rules.

In this example, you need to maintain the data consistency while the end user types in the address details. For example, take a scenario where the end user types a country and then a certain city that belongs to the country. If the end user then changes the country, the city content should be cleared since the typed city most probably does not belong to the new country. So you need to handle the change of the value. For this you will use a logic unit called **Variable Change**.

Variable Change Logic Unit

The Variable Change logic unit handles the change of a variable's value. In Magic xpa there are two ways that a variable value can be changed:

- Interactively by the user – The change is a result of editing a value of the control on the form.
- Non-interactively – The change is a result of a non-interactive change, such as the Update operation.

The Variable Change logic unit is triggered whenever the variable value is changed in one of the above ways.

Parameters

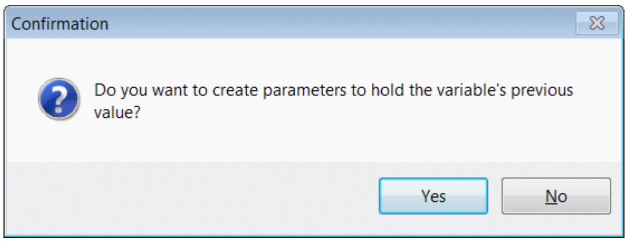
When you create a Variable Change logic unit, you are prompted to approve the creation of the following parameters. You will learn more about parameters later in the course.

Parameter	Description
CHG_REASON_ parameter	This parameter stores a number (0 or 1) that represents the reason the variable was changed. 0 – indicates an interactive change, such as editing the control 1 – indicates a non-interactive change, such as from an Update command This parameter must be a Numeric attribute.
CHG_PRV_ parameter	This parameter stores the variable's value before the change. This parameter must be of the same attribute as the variable chosen in the variable change.

This section will show you how to maintain the data consistency of the address details, using the Variable Change logic unit.

1. Zoom to **My First Program** and switch to the Logic Editor.
2. Create a Header line (**Ctrl+H**) and define a **Variable Change** logic unit for the **Country** variable.

A confirmation box is displayed, like the image on the right, asking you if you want to create parameters to hold the previous value.

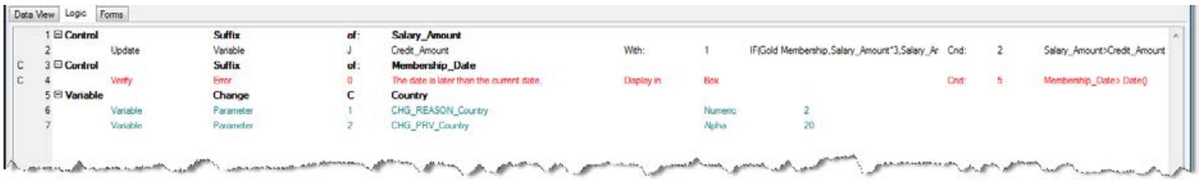


Do you want to create parameters to hold the variable's previous value?

3. Click the **Yes** button in the Confirmation box.

If you use the **Country** variable's value within this logic unit, the value will contain the updated value. The previous value is passed in the **CHG_PRV_ parameter**. As an example, if you changed the data in the **Country** variable from England to Canada, then in the Variable Change logic unit, Country will be Canada and **CHG_PRV_Country** will be England.
It is useful to use this within the Variable Change logic unit; for example, to compare the variable's current and previous values.

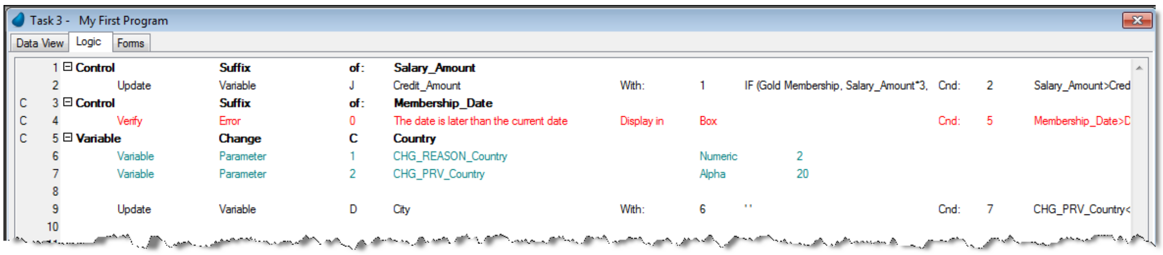
Your program should now look similar to the image below.



```
Data View  Logic  Forms
     1 ⊟ Control              Suffix      of:  Salary_Amount
     2          Update        Variable    J    Credit_Amount                     With:       1        IF(Gold Membership,Salary_Amount*3,Salary_Ar  Cnd:  2  Salary_Amount>Credit_Amount
C    3 ⊟ Control              Suffix      of:  Membership_Date
C    4          Verify        Error       0    The date is later than the current date  Display in  Box                                    Cnd:  5  Membership_Date> Date()
     5 ⊟ Variable             Change      C    Country
     6          Variable      Parameter   1    CHG_REASON_Country                            Numeric     2
     7          Variable      Parameter   2    CHG_PRV_Country                               Alpha       20
```

The letter **C** to the right of lines 3-4 is an indication that these operations are occurring on the **client**. This will be discussed in detail in a later lesson.

4. In the **Variable Change** logic unit, create a Detail line (**F4**).
   Note: For usability and readability, it is recommended to define new operations below the Parameters.
5. Create an **Update** operation for the **City** variable.
6. Update the **City** variable with the value **' '** (apostrophe, space, apostrophe). This is equivalent to resetting an Alpha variable. You are updating the **City** with a blank value.
7. You need to update this variable when the previous value had an actual value; in other words, it did not have a blank value. This means that the expression should be **CHG_PRV_Country <>' '**



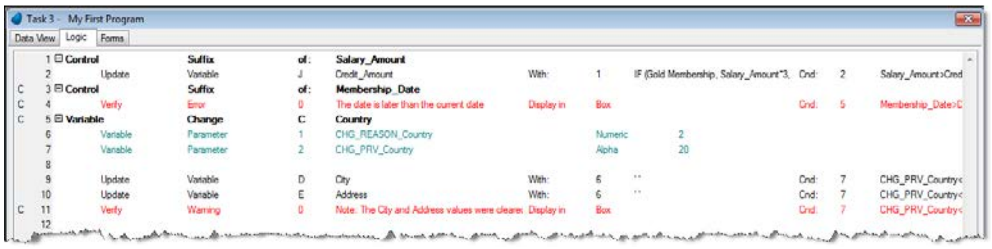
```
Task 3 -   My First Program
Data View  Logic  Forms
     1 ⊟ Control              Suffix      of:  Salary_Amount
     2          Update        Variable    J    Credit_Amount                     With:       1        IF (Gold Membership, Salary_Amount*3,  Cnd:  2  Salary_Amount>Cred
C    3 ⊟ Control              Suffix      of:  Membership_Date
C    4          Verify        Error       0    The date is later than the current date  Display in  Box                              Cnd:  5  Membership_Date>D
C    5 ⊟ Variable             Change      C    Country
     6          Variable      Parameter   1    CHG_REASON_Country                            Numeric     2
     7          Variable      Parameter   2    CHG_PRV_Country                               Alpha       20
     8
     9          Update        Variable    D    City                              With:       6        ''                               Cnd:  7  CHG_PRV_Country<
    10
```

8. Repeat the above steps for the **Address** variable. However, this time instead of entering new expressions, select the existing ones.

Now you will add a **Verify** operation to alert the end user that the values of the city and address were cleared.

9. In the **Variable Change** logic unit line, create a line and add a **Verify Warning** operation.
   The warning will be: **Note: The City and Address values were cleared!**
10. The condition for the operation should be the same condition as the Update operations: **CHG_PRV_Country <>' '**
11. Close the program and save the changes.

The program should look similar to this:

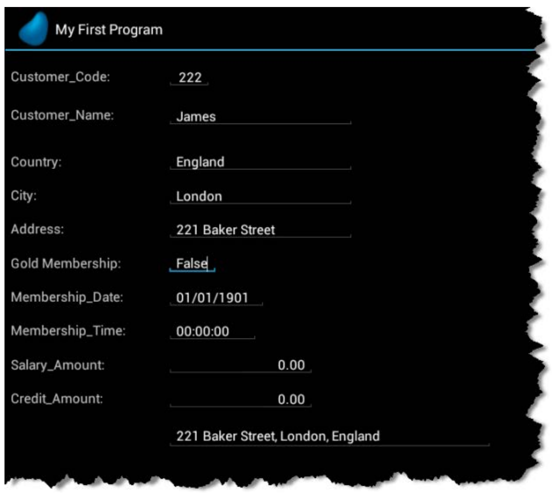


```
Task 3 - My First Program
Data View | Logic | Forms

   1 ⊟ Control         Suffix     of:  Salary_Amount
   2       Update      Variable   J    Credit_Amount                          With:       1   IF (Gold Membership, Salary_Amount*3,  Cnd:  2  Salary_Amount>Cred
C  3 ⊟ Control         Suffix     of:  Membership_Date
C  4       Verify      Error      0    The date is later than the current date Display in Box                                     Cnd:  5  Membership_Date>C
C  5 ⊟ Variable        Change     C    Country
   6       Variable    Parameter  1    CHG_REASON_Country                                 Numeric    2
   7       Variable    Parameter  2    CHG_PRV_Country                                    Alpha      20
   8
   9       Update      Variable   D    City                                   With:       6   ''                                     Cnd:  7  CHG_PRV_Country<
  10       Update      Variable   E    Address                                With:       6   ''                                     Cnd:  7  CHG_PRV_Country<
C 11       Verify      Warning    0    Note: The City and Address values were cleare Display in Box                               Cnd:  7  CHG_PRV_Country<
  12
```

Now you will run the program to view the results.

12. Execute the project and then open the Magic xpa client on your device.
13. Type the same customer details as you did earlier in this lesson.

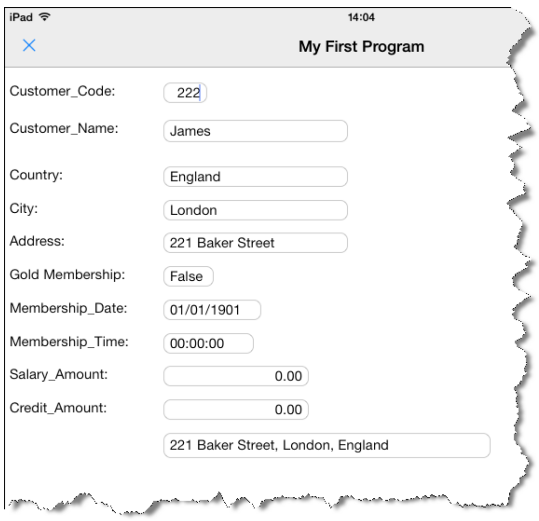
Android



My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	01/01/1901
Membership_Time:	00:00:00
Salary_Amount:	0.00
Credit_Amount:	0.00
	221 Baker Street, London, England

iOS



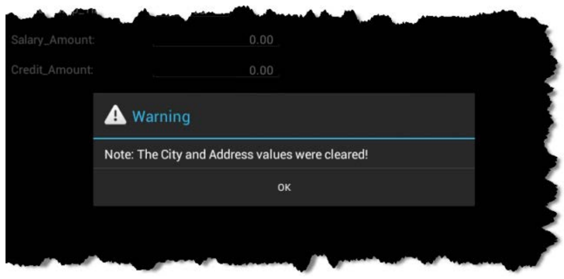
iPad 14:04

My First Program

Customer_Code:	222
Customer_Name:	James
Country:	England
City:	London
Address:	221 Baker Street
Gold Membership:	False
Membership_Date:	01/01/1901
Membership_Time:	00:00:00
Salary_Amount:	0.00
Credit_Amount:	0.00
	221 Baker Street, London, England

14. Change the **Country** value to: **Spain.**
15. Tap the **Customer_Name** control.

Android


Salary_Amount: 0.00

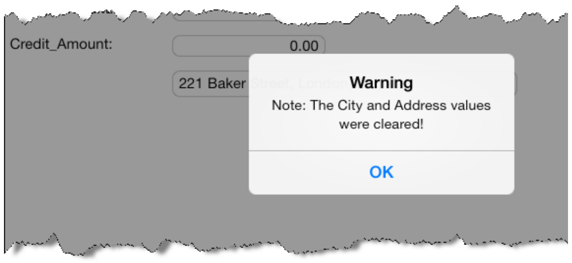
Credit_Amount: 0.00

Warning

Note: The City and Address values were cleared!

OK

iOS


Credit_Amount: 0.00

221 Baker

Warning

Note: The City and Address values were cleared!

OK

As you can see, when you changed the **Country** value and left the control, the values of the **City** and **Address** were cleared and a Warning box appeared to alert the end user of the changes.

Data Consistency – Short Summary

In the previous sections, you learned how to maintain data consistency.

You used the Variable Change logic unit to handle the change of the Country value.

Within the logic unit you defined Update operations to clear the City and Address values, and a Verify operation to alert the end user when the values were cleared.

You set a condition for the operations within the Variable Change logic unit: **Country Previous value <>''**.

This condition is necessary to make sure that the alert to the end user appears only if the Country value is changed from one value to another, and not when the Country value is empty and the end user enters a value for the first time.

You also saw three operations based on the same condition. This is inefficient coding as the condition is evaluated for each operation separately. In a later lesson you will learn about the Block operation and how to condition sections of code to make this example more efficient.

Parking Condition

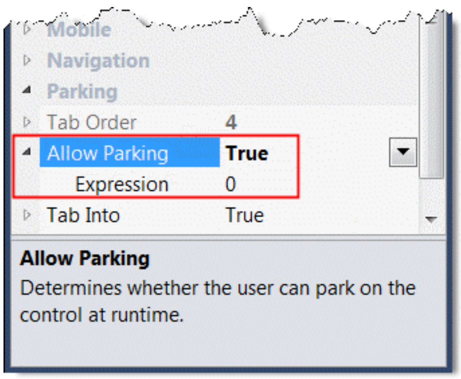
In the previous sections you learned one way to maintain data consistency; you handled the content of a variable as a result of a change to another variable's data. Now you will learn another way to maintain data consistency. In this section, you will control the parking ability of a control as a result of another variable's content.

In this example, the **City** control will be parkable only if the **Country** variable has a value, meaning that the variable is not empty. In addition, the **Address** control will be parkable only if both the **Country** and the **City** variables have values.

1. Zoom to the **My First Program** form of **My First Program**.

Now you will set a parking condition for the **City** Edit control, so that the control will be parkable only if the **Country** variable has a value.

2. Park on the **City** Edit control and open the **Control Properties** sheet (Alt+Enter).
3. Park on the **Allow Parking** property and set it to **True**.



Tab Order	4
Allow Parking	True
Expression	0
Tab Into	True

Allow Parking
Determines whether the user can park on the control at runtime.

The **Allow Parking** property enables you to define whether the user can park on a certain control. On a mobile device this means that the property controls whether you can tap that field and enter data.

When this property is set to **False**, you cannot tap on the field or move to it by any other means.

The **Allow Parking** property is enabled only when a variable is assigned to the control in the **Data** property.

Define an expression in the Expression Editor (zoom from the **Expression** line).

4. Enter the expression: **Country <>''**
5. Repeat this by setting a parking condition for the **Address** Edit control, so that the control will be parkable only if both the **Country** and **City** variables have values. Use the following expression: **Country <>'' AND City <>''**

The **AND** operator enables you to define a Boolean expression in which both expressions need to be evaluated to True in order for it to be valid.

6. Run the project.
7. Type a value for the **Customer_Code** and the **Customer_Name** values as before.
8. Tap the **City** control or the **Address** control.

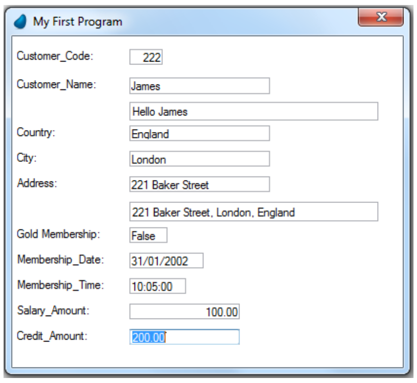
You will notice that focus passed to different controls.

9. Type in a value for the **Country** field, and tap the **City** field. You will now see that you can enter a value. You will still not be able to enter a value in the **Address** field.
10. Type in a value for the **City** variable, and tap the **Address** field. The **Address** field is now parkable.

Exercise

1. The USA banks decided to give their clients a benefit. Each USA client will receive an additional credit amount, which is 10 percent of the client's salary.
   - This calculation will be done after the end user types in the salary amount (in the Control Suffix of the **Salary_Amount** that you already defined).
   - In the **Update** operation of the **Credit_Amount** use the **IF** function to check if the customer's **Country=USA**.

2. To make your project friendlier, add a personal welcome announcement to the form.
   - After the **Customer_Name** field, add a **Hello** customer message, such as Hello George.
   - Move the Customer Address concatenated field that you added during this lesson to below the **Address** Edit control.
   - You can see an example in the image on the right.



My First Program

Customer_Code:	222
Customer_Name:	James
	Hello James
Country:	England
City:	London
Address:	221 Baker Street
	221 Baker Street, London, England
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	100.00
Credit_Amount:	200.00

Now you will practice what you learned about validation.

3. Add a validation to **My First Program** so that the end user will have to type in a **Customer_Code** before tapping a different control.
4. Maintain the data consistency of the **Membership_Date** and **Membership_Time** variables, so that if the date is changed, the time is cleared.
5. Do not allow the cursor to park on the **Membership_Time** field if the **Membership_Date** is empty. In Magic xpa, an empty date value is initially set to **'01/01/1901'Date**.
   Date is a **literal** and you will learn more about literals later in this course.

Summary

In this lesson you practiced some of the routine Magic xpa actions, such as zooming to programs and using the Expression Editor.

You learned about:

- Magic xpa arithmetic operators and how to use them to make calculations.
- The internal logic units and you used the Verify operation and the Update Variable operation. You were introduced to the Variable Change logic unit and learned how to use it within your project.
- The **IF** function and how to use it within expressions.
- Ways to manipulate alphanumeric data. You learned about the Magic xpa alphanumeric concatenation operator and how to use it to concatenate several strings.
- The **Trim** function, which removes spaces within strings.
- Magic xpa's internal data validation mechanism and its benefits.
- Validating the end-user data entry and how to maintain data consistency using the Verify operation.

Lesson 5

Setting Initial Values

Initializing variables with a default value is commonly used by developers during project development to save the end user the trouble of entering constant values, such as the current date.

Magic xpa enables you to set initial values for variables in different levels of the task.

Magic xpa provides you with a number of ways to initialize a variable value, such as the Update operation and the Init property.

This lesson covers various topics including:

- Init property
- Task Prefix logic unit
- Task Suffix logic unit
- Control logic unit

Update in Task Prefix

One of the ways to initialize variables with a value is by using the Update operation.

You can use the Update operation in each Task logic unit.

In Magic xpa you can define two **task** logic units:

- **Task Prefix** – In this logic unit, you can specify operations that Magic xpa executes at the beginning of the task execution.
  The operations stored in this logic unit are used as initialization procedures, such as initializing local variable values.
- **Task Suffix** – In this logic unit, you can specify operations that Magic xpa executes at the end of the task execution.
  The operations stored in this logic unit are used as task completion procedures, such as updating parameters.

In this section, you will use the Update operation in the Task Prefix logic unit to set variable values.

System Date and Time

Here you will update the **Membership_Date** variable with the **Date** function, which returns the system date (the current date), and update the **Membership_Time** variable with the **Time** function, which returns the system time (the current time).

You will use the Update operation in the Task Prefix logic unit so that the updating process will be executed at the **beginning** of the task execution.

1. In **My First Program** create a **Task Prefix** logic unit.
2. In this logic unit, update the **Membership_Date** variable with the **Date** () function and the **Membership_Time** variable with the **Time** () function.
3. Execute the project and run the Magic xpa client.

When you run the program, you will see that the **Membership_Date** field is updated with the current date value and the **Membership_Time** field is updated with the current time value.

Variable Initialization

In the last section, you learned how to set a value for a variable using the **Update** operation.

In some cases, there is a need to initialize a variable with a certain value.

There are some differences between the **Update** operation and variable initialization.

The **Update** operation is a procedural operation that is executed each time Magic xpa reaches the **Update** command.

The **variable initialization** is divided into two major phases:

- The procedural phase is executed when a record is created. The behavior depends on whether you have Virtual variables or columns:
    - Virtual variable values are initialized in Create and Modify mode.
    - Column values are initialized in Create mode.
- The **non-procedural** phase is executed when the Init expression includes other variables within it.
    - The value of the Virtual variables and the columns with an Init expression will be **recomputed** immediately after one (or more) of the expression's variables have been changed.

You will learn about the table columns and task modes later in this course.

The recompute mechanism is active only when a variable, which was defined prior to the current variable in the data view, was changed. This means that changes in variable A will affect variable B only if variable A is defined before variable B in the data view.

During the previous lessons, you probably found it tedious to type in the same information for every example. To handle this you can use the **Init** property to initialize the Virtual variable values.

The execution side of the **Init** value can affect performance. This will be discussed further in a later lesson.

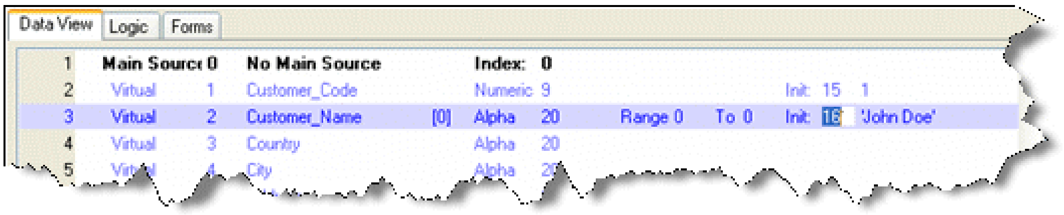
In this example, you will initialize the **Customer_Code** variable with a value of **1** and the **Customer_Name** variable with the **John Doe** value. You will use the **Init** property to initialize the variable value, so that the variable value will be updated before the variable is displayed on the form.

1. In **My First Program**, go to the Data View Editor and park on the **Customer_Code** variable definition line.
2. Zoom from the **Init** property and enter an expression for the value: **1**

The **Customer_Code Init** expression displays the expression number and the expression value is displayed to the right of it.

3. Park on the **Customer_Name** variable and enter the following expression for the **Init** property: '**John Doe**'

For an Alpha expression, you must use single straight quotation marks at the beginning and the end of the string expression (' ').

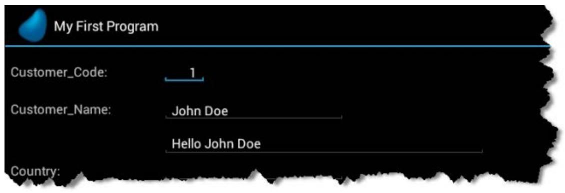


```
1  Main Sourc 0  No Main Source          Index:  0
2  Virtual    1  Customer_Code           Numeric 9                      Init: 15  1
3  Virtual    2  Customer_Name      [0]  Alpha   20   Range 0   To 0   Init: 16  'John Doe'
4  Virtual    3  Country                 Alpha   20
5  Virtual    4  City                    Alpha   20
```

4. Execute the project and run the Magic xpa client.

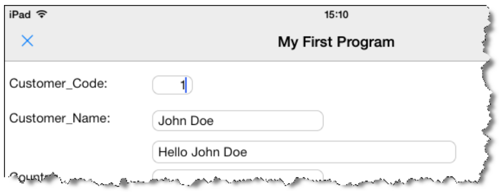
As you can see, when you run the program, the customer code and name values are initialized with the expressions that you set.

Android



Customer_Code: 1
Customer_Name: John Doe
Hello John Doe
Country:

iOS



Customer_Code: 1
Customer_Name: John Doe
Hello John Doe

Update in Control Prefix

Up until now, you updated the variable value in the Task Prefix logic unit and in the Control Suffix logic unit. You can also update a variable in the Control Prefix logic unit.

The Control logic unit enables you to handle the behavior of a specific control. You can use the Control logic unit for various purposes, such as setting values for variables, and checking input before leaving the control.

When you tap a control, the Control Prefix logic unit operations for that control are executed before the focus is moved to the control.

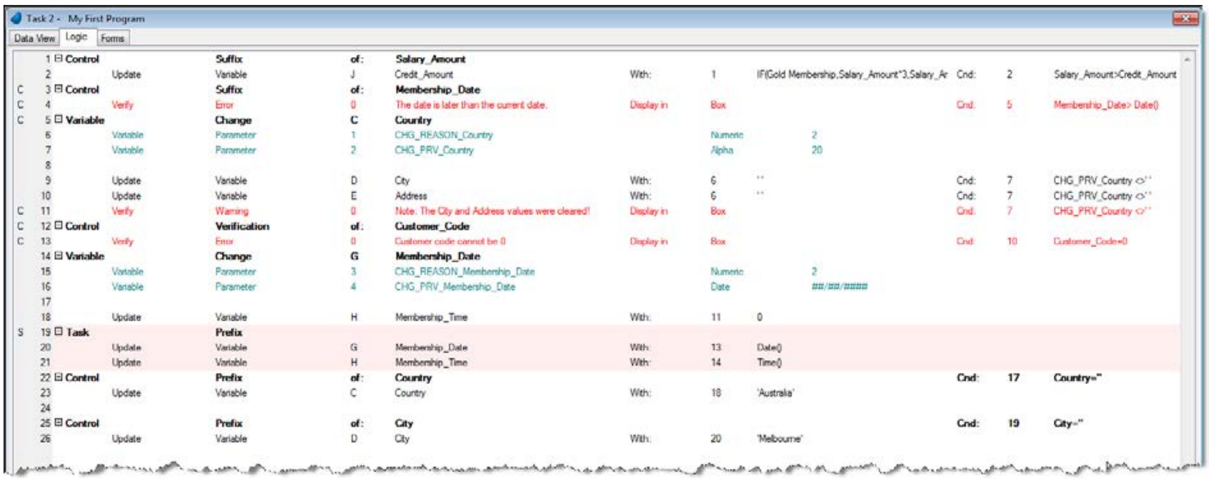
Control Suffix logic unit operations are executed before the focus is moved from the particular control to a different control. This logic unit was discussed in a previous lesson.

If you set the Update operation within the Control Prefix logic unit, the operation will be executed just before the cursor is moved to the control.

In this example, you will update the **Country** variable with the value: **'Australia'** and the **City** variable with the **'Melbourne'** value.

You will use the Update operation in the Control Prefix logic unit.

1. Add a **Control Prefix** logic unit for the **Country** variable.
2. The condition for this logic unit is: **Country** variable is blank (**Country=''**).
3. In this logic unit, update the **Country** variable with: **'Australia'**
4. Add a **Control Prefix** logic unit for the **City** variable.
5. The condition for this logic unit is: **City** variable is blank (**City=''**).
6. In this logic unit, update the **City** variable with: **'Melbourne'**


Task 2 - My First Program

Data View | Logic | Forms

```
     1 ⊟ Control            Suffix        of:  Salary_Amount
     2            Update    Variable      J    Credit_Amount          With:      1         IF(Gold Membership,Salary_Amount*3,Salary_Ar  Cnd:  2   Salary_Amount>Credit_Amount
C    3 ⊟ Control            Suffix        of:  Membership_Date
C    4            Verify    Error         0    The date is later than the current date.   Display in  Box    Cnd:  5   Membership_Date> Date()
C    5 ⊟ Variable           Change        C    Country
     6            Variable  Parameter     1    CHG_REASON_Country                Numeric    2
     7            Variable  Parameter     2    CHG_PRV_Country                   Alpha      20
     8
     9            Update    Variable      D    City                   With:      6         ''        Cnd:  7   CHG_PRV_Country <>''
    10            Update    Variable      E    Address                With:      6         ''        Cnd:  7   CHG_PRV_Country <>''
C   11            Verify    Warning       0    Note: The City and Address values were cleared!   Display in  Box   Cnd:  7   CHG_PRV_Country <>''
C   12 ⊟ Control            Verification  of:  Customer_Code
C   13            Verify    Error         0    Customer code cannot be 0   Display in  Box              Cnd:  10  Customer_Code=0
    14 ⊟ Variable           Change        G    Membership_Date
    15            Variable  Parameter     3    CHG_REASON_Membership_Date        Numeric    2
    16            Variable  Parameter     4    CHG_PRV_Membership_Date           Date       ##/##/####
    17
    18            Update    Variable      H    Membership_Time        With:      11        0
S   19 ⊟ Task               Prefix
    20            Update    Variable      G    Membership_Date        With:      13        Date()
    21            Update    Variable      H    Membership_Time        With:      14        Time()
    22 ⊟ Control            Prefix        of:  Country                                               Cnd:  17  Country=''
    23            Update    Variable      C    Country                With:      18        'Australia'
    24
    25 ⊟ Control            Prefix        of:  City                                                  Cnd:  19  City=''
    26            Update    Variable      D    City                   With:      20        'Melbourne'
```

7. Execute the project.
8. Tap the **Country** field.
9. Tap the **City** field.

As you can see:

- The Membership Date and Time values are updated when the program begins.
- The Customer Code and Name values are initialized when the program begins.
- The Country value is updated only when you park on the **Country** field.
- The City value is updated only when you park on the **City** field.

Note that if you change the country name and then tap another field, the Variable Change logic unit will be active and you will get a message that the City and Address fields were cleared. However, if you tap the **City** field, Magic xpa will invoke the Control Prefix logic unit and update the City with **"Melbourne"**.

Exercise

1. Update the **City** variable to Melbourne only if the country is Australia.
2. If the country is Tartarus, then the initial credit will be zero. Use the **Init** property for this.
3. The **Country** variable is an Alpha field. Are the values Tartarus, tartarus or TARTARUS all the same? If not, how can you check what the user entered so that you can use it in an expression? How would you change the expression you set in question 2?
4. If the country is Xanadu and the customer has a gold membership, then set the initial credit to 1000.

Summary

In this lesson you learned how to initialize values for variables in several ways.

You set variable values using:

- The Update operation in different logic units. You used the Update operation in this lesson and previous lessons.
- The variable's **Init** property.

You learned about the behavior of each of the above options.

This lesson introduced you to the Task logic unit and the Control Prefix logic unit.

Lesson 6

Setting the Form's Appearance

In previous lessons you were introduced to the **Form Designer**. You learned about the main form and designed a basic form to display the task variables.

Magic xpa provides you with various tools to set the form's appearance, such as the form's background color and wallpaper as well as the controls' color, font, style, size, and location.

This lesson covers various topics including:

- Color, font, and style
- Wallpaper

In the Appendix of this course, you can find a best practices guide about form design for mobile devices.

Adding Colors to the Color Repository

Adding a Text Caption Color

Magic xpa provides you with a specific color file for each environment:

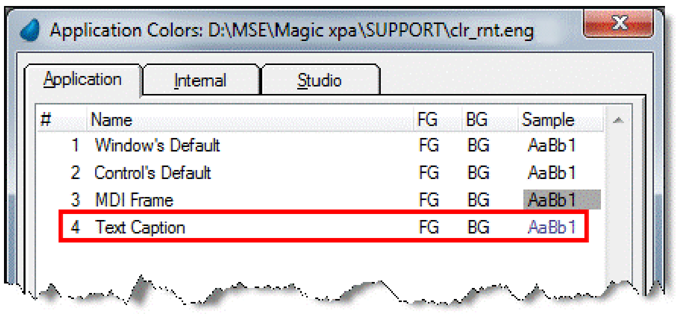
- **Application** – The default Application color file is **clr_rnt.eng**. You can add, delete, and modify any entry in these files.
- **Internal** – The default Internal color file is **clr_int.eng**. This includes colors that are used at runtime to display built-in screens, such as the environment setting dialog boxes and the Range/Locate dialog boxes. You cannot add new colors and you cannot rename existing colors. However, the entries' color settings can be modified.
- **Studio** – The default Studio color file is **clr_std.eng**. This includes only the colors that affect the Studio screens. You cannot add new colors and you cannot rename existing colors. However, the entries' color settings can be modified.

Each file contains a list of color definitions, which include a foreground color (FG) and a background color (BG).

It is good practice not to delete entries in the Magic xpa files, but only in the application file.

Now you will add two color definitions to the Color repository.

1. From the **Options** menu, select **Settings** and then **Colors**.
2. Click the **Application** tab.
   a. Park on the last color line.
   b. Create a line by pressing **F4**.
3. In the **Name** column, type: **Text Caption**.
4. Zoom from the **FG** column.
5. Select the first (empty) entry from the **System** drop-down list.
6. Set the following RGB colors:
   - Red = 68
   - Green = 68
   - Blue = 138
7. Click **OK**.
8. Zoom from the **BG** column.
9. Select the first (empty) entry from the **System** drop-down list.
10. Check the **Transparent** check box.
11. Click **OK**.


Application Colors: D:\MSE\Magic xpa\SUPPORT\clr_rnt.eng

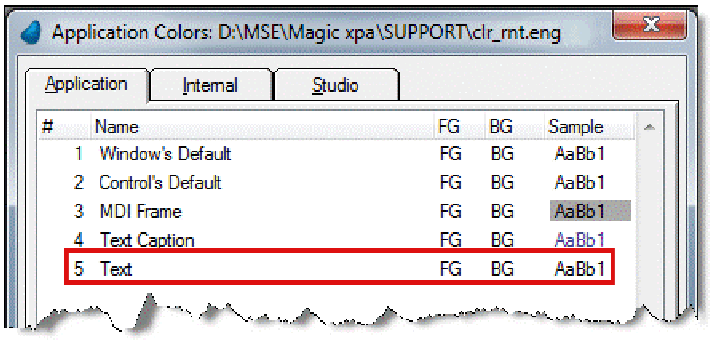
#	Name	FG	BG	Sample
1	Window's Default	FG	BG	AaBb1
2	Control's Default	FG	BG	AaBb1
3	MDI Frame	FG	BG	AaBb1
4	Text Caption	FG	BG	AaBb1

You gave the color a meaningful name, **Text Caption**. This will help you maintain consistency in your project if you assign this color to all the Text controls that are used as captions.

Adding a Text Color

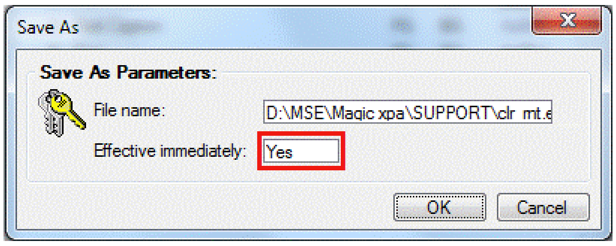
In a similar manner you will add a color that will be used for the text.

1. Create a line.
2. In the **Name** column, type: **Text**
3. Zoom from the **BG** column.
4. In the **Text: BG** window, select the first (empty) entry from the **System** drop-down list.
5. Check the **Transparent** check box.
6. Click **OK**.


Application Colors: D:\MSE\Magic xpa\SUPPORT\clr_rnt.eng

#	Name	FG	BG	Sample
1	Window's Default	FG	BG	AaBb1
2	Control's Default	FG	BG	AaBb1
3	MDI Frame	FG	BG	AaBb1
4	Text Caption	FG	BG	AaBb1
5	Text	FG	BG	AaBb1

7. Click **OK** to close the Color repository.
   A **Save As** dialog box will appear.
8. From the **Effective immediately** parameter, select **Yes**.
9. Click **OK**.


Save As

Save As Parameters:

File name: D:\MSE\Magic xpa\SUPPORT\clr_rnt.e

Effective immediately: Yes

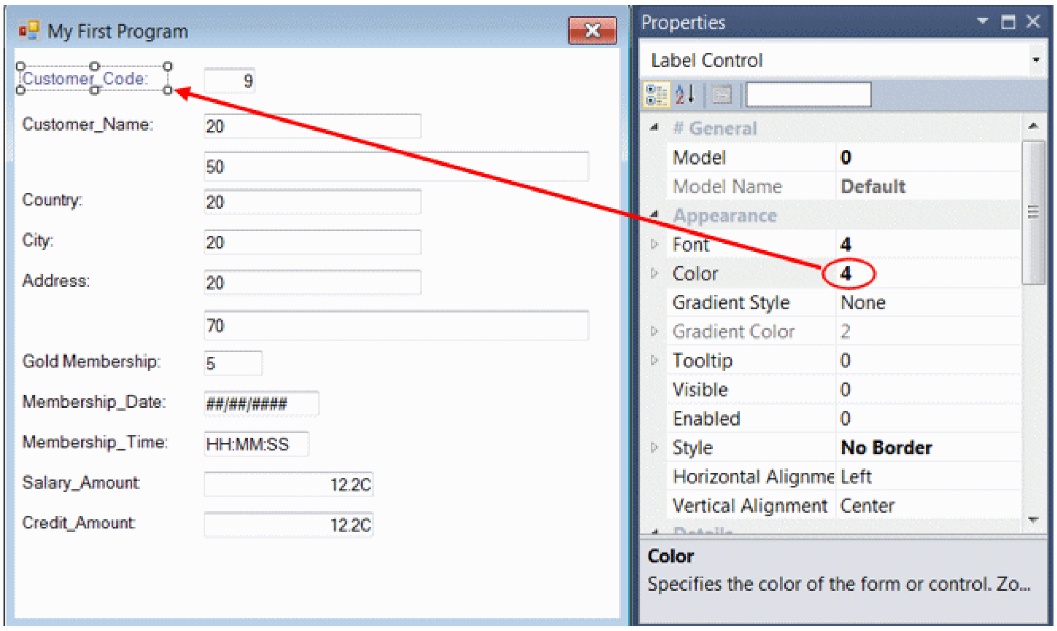
Changing a Control's Color

Magic xpa enables you to set the appearance of each control placed on the form. In this section, you will learn how to change the color of a control.

In the Color repository, you defined two colors: **Text Caption (color # 4)** and **Text (color # 5)**. Now, you will change the color of the Label controls to the **Text Caption** color.

1. Zoom to the My First Program form of My First Program.
2. Select the **Customer_Code** Label control by clicking on it.
3. If the Label control's Control Properties sheet is not open, right-click on the control and select **Properties** (Alt+Enter).
4. From the **Color** property, zoom to the Color repository and select the entry **Text Caption (# 4)**.

As you can see, the color of the **Customer_Code** Label control has changed to blue.



Customer_Code:	9
Customer_Name:	20
	50
Country:	20
City:	20
Address:	20
	70
Gold Membership:	5
Membership_Date:	##/##/####
Membership_Time:	HH:MM:SS
Salary_Amount	12.2C
Credit_Amount	12.2C

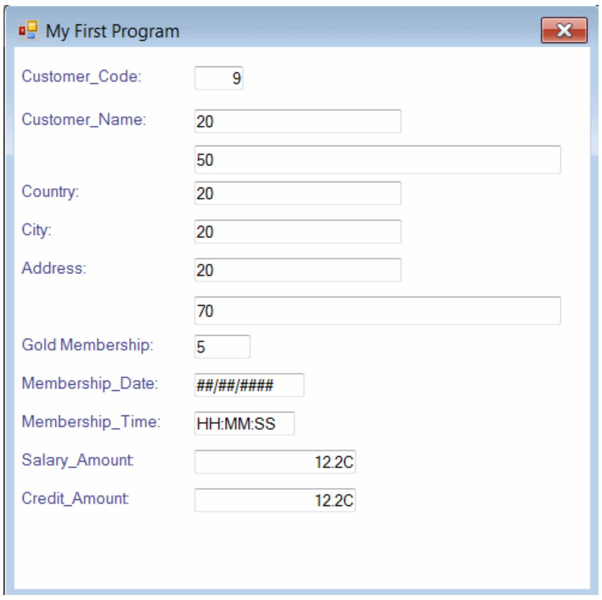
# General	
Model	0
Model Name	Default
Appearance	
Font	4
Color	4
Gradient Style	None
Gradient Color	2
Tooltip	0
Visible	0
Enabled	0
Style	No Border
Horizontal Alignme	Left
Vertical Alignment	Center

Color
Specifies the color of the form or control. Zo...

5. Repeat stages 2-4 for all of the Label controls to change their color to the **Text Caption** color.
6. Close the program and save the changes.

It is possible to select a number of controls at the same time and then to set the same color for all of the select controls. You do this using one of the following methods:

- Press the **Ctrl** key and then click each control.
- Click above the first control, keep the left mouse button pressed down, and move the mouse down until all of the controls are marked.



Customer_Code:	9
Customer_Name:	20
	50
Country:	20
City:	20
Address:	20
	70
Gold Membership:	5
Membership_Date:	##/##/####
Membership_Time:	HH:MM:SS
Salary_Amount	12.2C
Credit_Amount	12.2C

Adding Fonts to the Font Repository

Adding a Text Caption Font

Similar to the colors, Magic xpa provides you with a specific Font file for each environment:

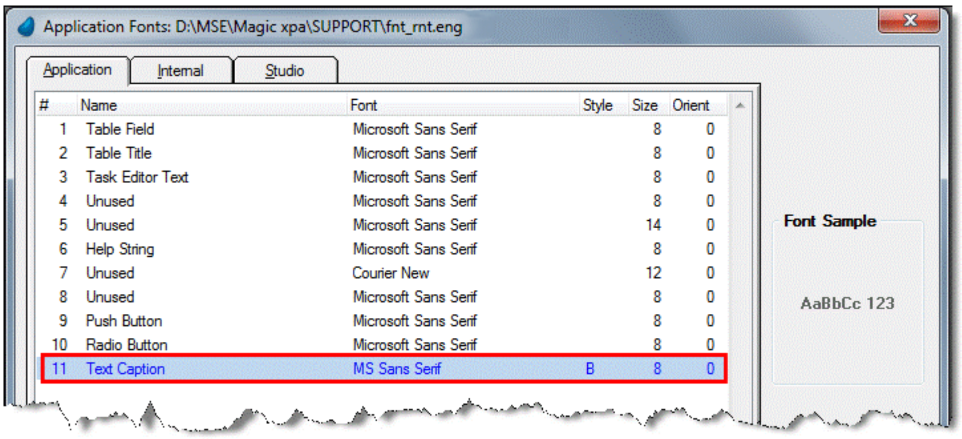
- **Aplication – fnt_rnt.eng** is the default Application font file, which contains a list of 10 default font definitions. You can add an unlimited number of application font definitions to the list, which you can then use to design your task forms and controls.
- **Internal – fnt_int.eng** is the default Internal font file, which contains a list of 100 fixed font definitions.
- **Studio – fnt_std.eng** is the default Studio font file, which contains a list of 100 fixed font definitions.

The list contains font definitions that are used within the tool (for windows, operations, expressions, and so on).

Now you will add two font definitions to the Font repository.

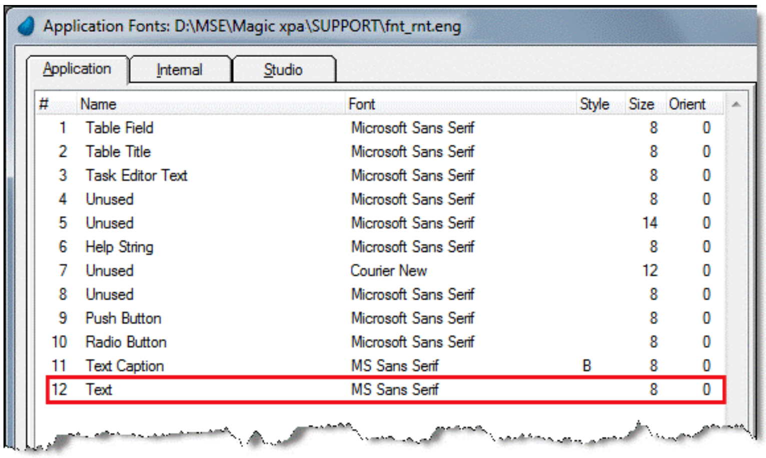
1. From the **Options** menu, select **Settings** and then **Fonts**.
2. Click the **Application** tab.
3. Create a line and in the **Name** column, type: **Text Caption**.
4. Zoom from the **Font** column.
5. From the **Font Style** list, select **Bold**.

The default font in the Studio is MS Sans Serif. This font may not be a valid font in the device that you are using. The device will reset the font type to its own default setting. It will use the Font Style that you defined. In a later lesson, you will learn how to use device specific fonts.


Application Fonts: D:\MSE\Magic xpa\SUPPORT\fnt_rnt.eng

#	Name	Font	Style	Size	Orient
1	Table Field	Microsoft Sans Serif		8	0
2	Table Title	Microsoft Sans Serif		8	0
3	Task Editor Text	Microsoft Sans Serif		8	0
4	Unused	Microsoft Sans Serif		8	0
5	Unused	Microsoft Sans Serif		14	0
6	Help String	Microsoft Sans Serif		8	0
7	Unused	Courier New		12	0
8	Unused	Microsoft Sans Serif		8	0
9	Push Button	Microsoft Sans Serif		8	0
10	Radio Button	Microsoft Sans Serif		8	0
11	Text Caption	MS Sans Serif	B	8	0

6. Add a new Magic xpa font named **Text**, using the default values.


Application Fonts: D:\MSE\Magic xpa\SUPPORT\fnt_rnt.eng

#	Name	Font	Style	Size	Orient
1	Table Field	Microsoft Sans Serif		8	0
2	Table Title	Microsoft Sans Serif		8	0
3	Task Editor Text	Microsoft Sans Serif		8	0
4	Unused	Microsoft Sans Serif		8	0
5	Unused	Microsoft Sans Serif		14	0
6	Help String	Microsoft Sans Serif		8	0
7	Unused	Courier New		12	0
8	Unused	Microsoft Sans Serif		8	0
9	Push Button	Microsoft Sans Serif		8	0
10	Radio Button	Microsoft Sans Serif		8	0
11	Text Caption	MS Sans Serif	B	8	0
12	Text	MS Sans Serif		8	0

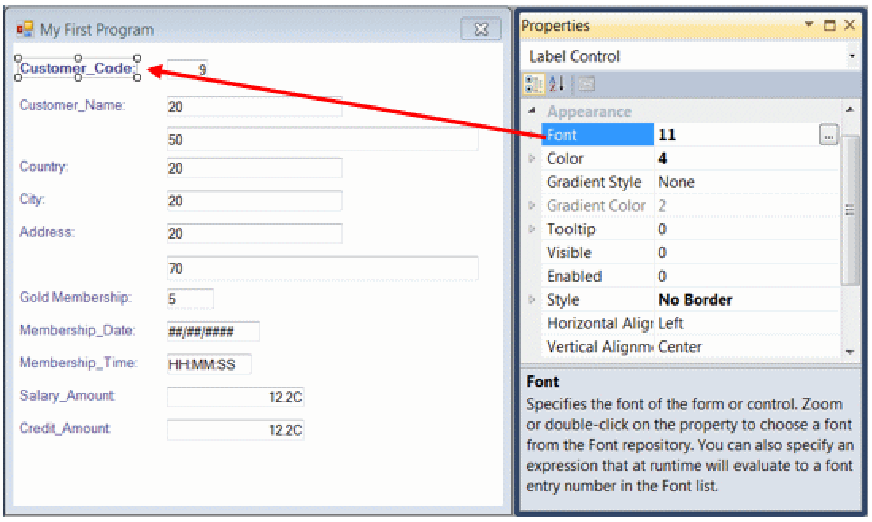
7. Click **OK** to close the Font repository.
   The **Save As** dialog box will appear.
8. From the **Effective immediately** parameter, select **Yes**.
9. Click **OK**.

Changing a Control's Font

Magic xpa enables you to control the appearance of each control placed on the form. In this section, you will learn how to change the control's font.

You previously added two new font definitions to the Font repository, **Text Caption (font # 11)** and **Text (font # 12)**. You will now change the font of the Label controls to the **Text Caption** font.

10. Zoom to the **My First Program** form (of **My First Program**).
11. Select the **Customer_Code** Label control by clicking on it and open the Control Properties sheet.
12. From the **Font** property, zoom to the Font repository and select the entry named **Text Caption**.
13. In the same way that you changed the **Customer_Code** Label control, change the font of all the Label controls on the form to the **Text Caption** font.



Customer_Code	9
Customer_Name:	20
	50
Country:	20
City:	20
Address:	20
	70
Gold Membership:	5
Membership_Date:	##/##/####
Membership_Time:	HH:MM:SS
Salary_Amount:	12.2C
Credit_Amount:	12.2C

Font	11
Color	4
Gradient Style	None
Gradient Color	2
Tooltip	0
Visible	0
Enabled	0
Style	No Border
Horizontal Aligı	Left
Vertical Alignm	Center

Font
Specifies the font of the form or control. Zoom or double-click on the property to choose a font from the Font repository. You can also specify an expression that at runtime will evaluate to a font entry number in the Font list.

Remember: You can select multiple controls by pressing **Ctrl** and clicking on the controls.

Move a Control

You may need to move controls to make the form look the way you want. You can move controls using the mouse or the keyboard as follows:

- By mouse: Click on the middle of the control and drag the control on the form to the requested position.
- By keyboard: Select the control and then move the control using the arrow keys.

The following instructions show you how to move the **Customer_Code** Edit control to the right, in order to uncover the **Customer_Code** Label control's content.

Using the Mouse

- Click on the **Customer_Code** Edit control and drag it to the right.
- Position it so that the entire **Customer_Code** Label control is displayed.

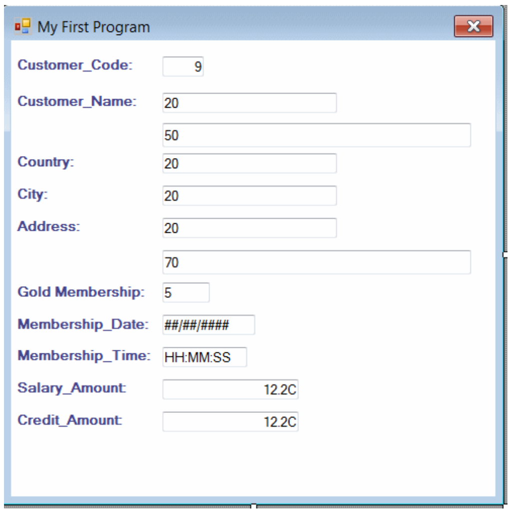
Using the Keyboard

- Click on the **Customer_Code** Edit control.
- Press the right arrow key to move the Edit control to the right.
- Position it so that the entire **Customer_Code** Label control is displayed.

Reset the Control Size

The font you have chosen is bigger than the existing font. Therefore, you can see only part of the text for some of the Label controls. You can manually resize the control by dragging it to the right. Or, you can use Magic xpa's **Fit Control Size** tool to adjust the control's size to the control's picture size.

14. Select the **Customer_Name** Label control by clicking on it.
15. In the toolbar, click the **Fit Control Size** icon .
16. Change the positions of the rest of the Edit controls so that the Label controls are displayed in their entirety.



Customer_Code:	9
Customer_Name:	20
	50
Country:	20
City:	20
Address:	20
	70
Gold Membership:	5
Membership_Date:	##/##/####
Membership_Time:	HH:MM:SS
Salary_Amount:	12.2C
Credit_Amount:	12.2C

Wallpaper

Magic xpa enables you to set wallpaper for a form. The wallpaper is a file that serves as the background of the window. The wallpaper is one of a form's appearance properties, which enable you to control the form's appearance. In this section, you will set a wallpaper file to be used as the form background and you will be introduced to some of the form properties.

Copying Files

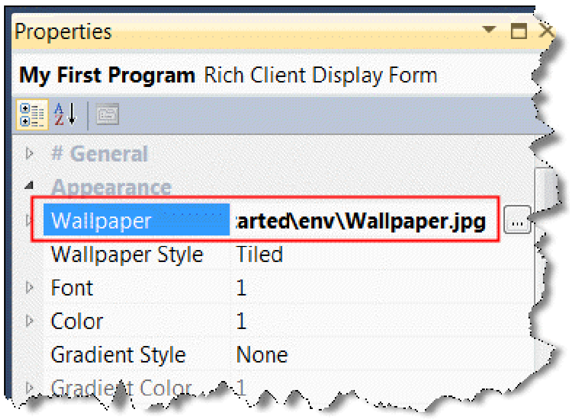
There are several directories and files that were prepared for you to use during the course.

Please copy the following folders from the **CourseData** directory into your project (**Getting Started**) directory.

- data
- env
- images
- products
- Text

Setting a Wallpaper File in the Form Properties

1. Zoom to the **My First Program** form.
2. Open the Form Properties.
3. From the **Wallpaper** property, click the **Zoom** button to browse for a file.
4. Select the **Wallpaper.jpg** file (located in: **[Magic xpa installation]\Projects\Getting Started\env**) and click the Open button.


Properties

My First Program Rich Client Display Form

General

Appearance

Wallpaper	arted\env\Wallpaper.jpg
Wallpaper Style	Tiled
Font	1
Color	1
Gradient Style	None
Gradient Color	1

Dynamic Variable

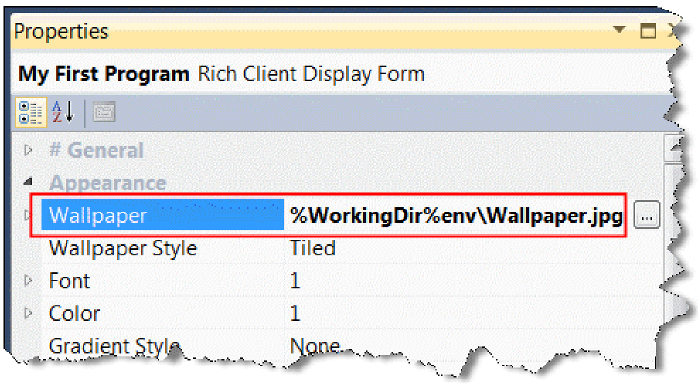
You have just set a file name and path as the form's wallpaper.

Using a specific file name and its path within a project may have some disadvantages later on. For example, if you use the **Wallpaper.jpg** file within several programs as its form's wallpaper, and then you change the directory name or just move the file to another location, you need to change the file name in every program that uses it as wallpaper. This may be very complicated in multi-program projects. Moreover, the same project can be executed on different computers with different file locations, so that the file references must be adapted to each computer environment.

To solve this problem, Magic xpa provides you with **Logical Names**. You will learn more about these special variables in a later lesson. For now you will learn about a useful internal Logical Name:

- **%WorkingDir%** – This logical name contains the value of the path of the working directory. In the current example that means:
  <Magic xpa installation path>\projects\Getting Started.

5. Park on the **Wallpaper** property.
6. Edit the property to: **%WorkingDir%env\Wallpaper.jpg**.


Properties

My First Program Rich Client Display Form

# General	
Appearance	
Wallpaper	%WorkingDir%env\Wallpaper.jpg
Wallpaper Style	Tiled
Font	1
Color	1
Gradient Style	None

You can open the Form Designer and see that the wallpaper image is still there.

Transparent Color for a Control

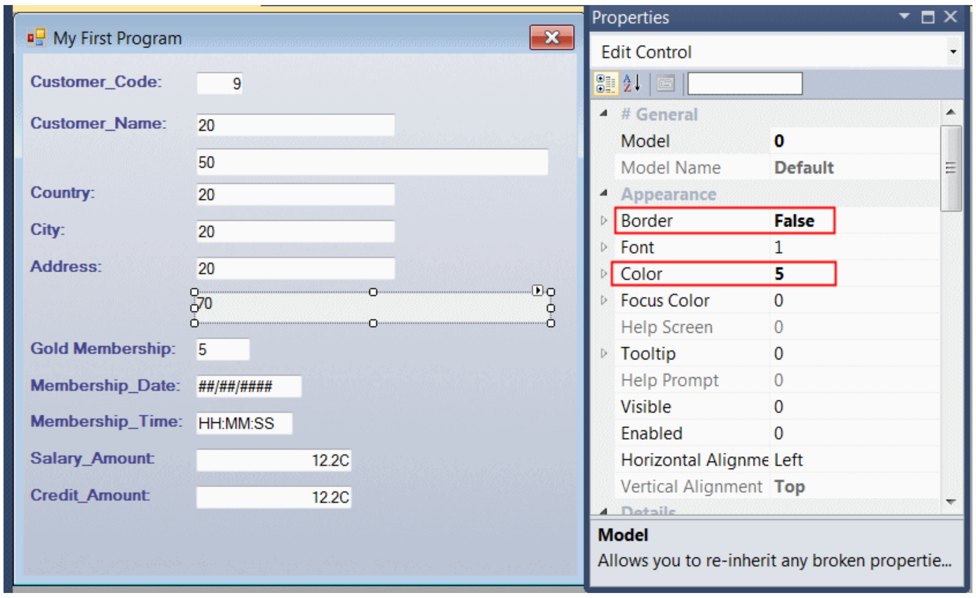
When setting wallpaper, sometimes you want the controls to have a transparent color so that they fit in with the background.

To make a control transparent on its background, the control needs a color that has a transparent background.

In the following example you will set the full address Edit control as a transparent control.

1. In the Form Designer, park on the full address Edit control and open the control properties.
2. Set the **Border** property to **False**.
3. From the **Color** property, select the **Text** entry.

In the image below you can see that an image is set for the form as a result of the wallpaper setting and the full address control appears as a transparent control on the wallpaper background.



My First Program

Customer_Code:	9
Customer_Name:	20
	50
Country:	20
City:	20
Address:	20
	70
Gold Membership:	5
Membership_Date:	##/##/####
Membership_Time:	HH:MM:SS
Salary_Amount:	12.2C
Credit_Amount:	12.2C

Properties

Edit Control

# General	
Model	0
Model Name	Default
Appearance	
Border	False
Font	1
Color	5
Focus Color	0
Help Screen	0
Tooltip	0
Help Prompt	0
Visible	0
Enabled	0
Horizontal Alignme	Left
Vertical Alignment	Top
Details	

Model
Allows you to re-inherit any broken propertie...

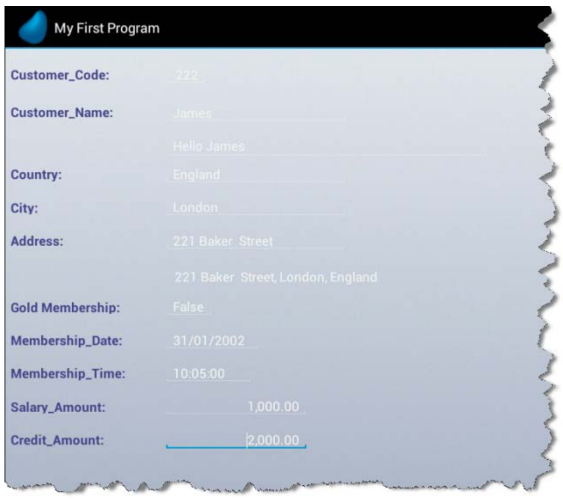
4. Close the program and save the changes.

Running the Program

You completed the form design. The program is ready for execution. Now, you will run the program to view the results.

1. Run **My First Program**.
2. Type the customer details as shown in the image below.

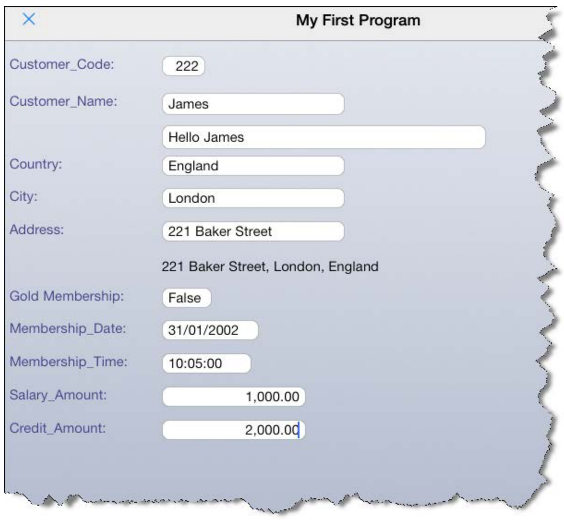
Android



My First Program

Customer_Code:	222
Customer_Name:	James
	Hello James
Country:	England
City:	London
Address:	221 Baker Street
	221 Baker Street, London, England
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2,000.00

iOS



My First Program

Customer_Code:	222
Customer_Name:	James
	Hello James
Country:	England
City:	London
Address:	221 Baker Street
	221 Baker Street, London, England
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2,000.00

As you can see in the images above, the use of wallpaper provides a more enhanced look than the original black background. The program running on the Android device and the iOS device have a similar look. As you can see from the image on the left (the Android image), the Android system's edit control default color does not look good on the wallpaper image. You will change this in the exercise.

Placement

Placement is a very important issue when executing your program on a mobile device. You will see the reason with a small example.

1. Open your program, **My First Program**, and add a new Virtual variable named **Address Label**.
2. Make this an Alpha variable with a length of **200**.

In an address label, you would have the name of the person on one line and then the address on another and so forth. You need to have a way of moving the text to a new line.

Syntax	**ASCIIChr (*numeric*)**
Parameters	*Numeric*: A number from 0 to 255 which represents an ASCII character.
Returns	A character representing the ASCII value.
Note	ASCII 10 represents a line feed.

3. Set the **Init** property of the **Address Label** to the following expression:

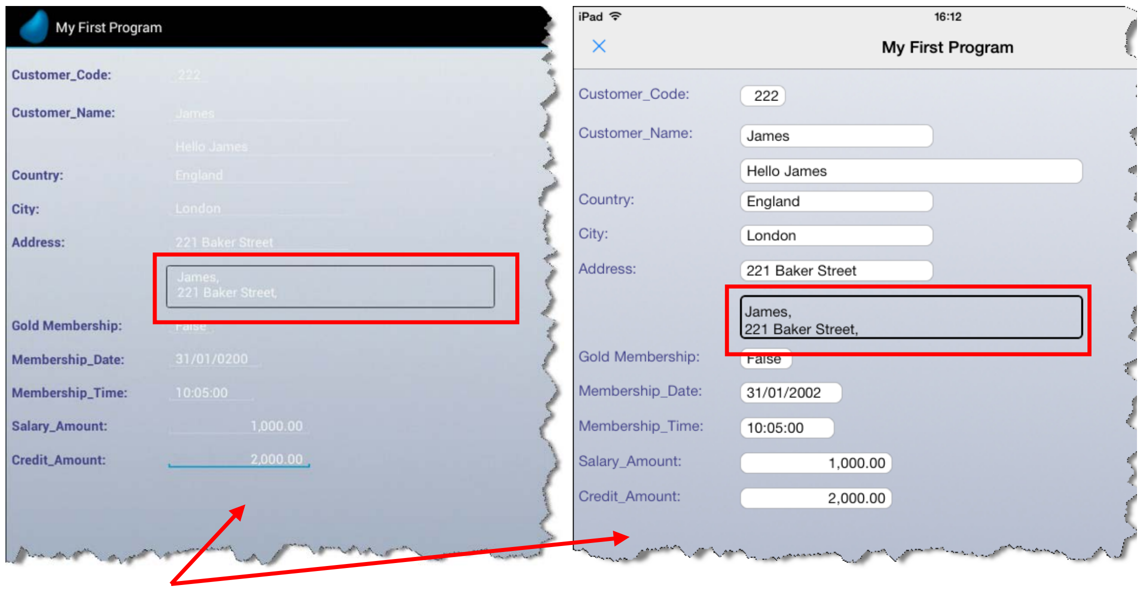
   Trim (B)&',' & ASCIIChr (10)&Trim(E)&',
   '&ASCIIChr (10)&Trim(D)&','&ASCIIChr (10)&Trim(C)

   where **B** is **Customer Name**, **E** is **Address**, **D** is **City** and **C** is **Country**.
4. Zoom into the form and move all the controls from **Gold_Membership** onwards to further down on the form.
5. Park on the customer address concatenated Edit control that you added on the form.
   a. Zoom into the **Data** property and change the value to the **Address Label** variable.
   b. Right-click on the the **Format** property and re-inherit the value by right-clicking on the selecting **Inherit**. The value should now show **200** and not **70**.
   c. In the **Appearance** section, set the **Vertical Alignment** property to **Top**.
   d. Reset the **Border** property to **True**.
   e. In the **Input** section, set the **Multiline Edit** property to **True**.
   f. In the **Navigation** section, set the **Height** to **2.5**.

g. Make sure that the height does not overlap the **Gold Membership** Edit control. If it does, move the **Gold_Membership** control downwards.

Now you can execute the program and see the result.

Android iOS



My First Program

Customer_Code: 222
Customer_Name: James
Hello James
Country: England
City: London
Address: 221 Baker Street
James,
221 Baker Street,
Gold Membership: False
Membership_Date: 31/01/0200
Membership_Time: 10:05:00
Salary_Amount: 1,000.00
Credit_Amount: 2,000.00

iPad 16:12
My First Program

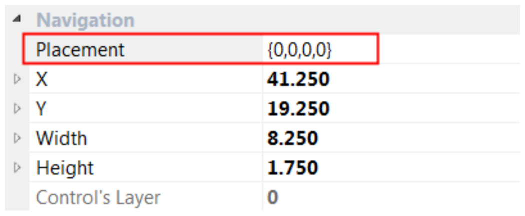
Customer_Code: 222
Customer_Name: James
Hello James
Country: England
City: London
Address: 221 Baker Street
James,
221 Baker Street,
Gold Membership: False
Membership_Date: 31/01/2002
Membership_Time: 10:05:00
Salary_Amount: 1,000.00
Credit_Amount: 2,000.00

As you can see in the image above, the **Address Label** shows only part of the data, yet at the bottom of the form there is more than enough room to display the data. This is true when holding the tablet in Portrait mode. Is it true when you hold it in Landscape mode? What happens when you run the same program on a smartphone or a tablet with a smaller screen? This is where placement comes in.

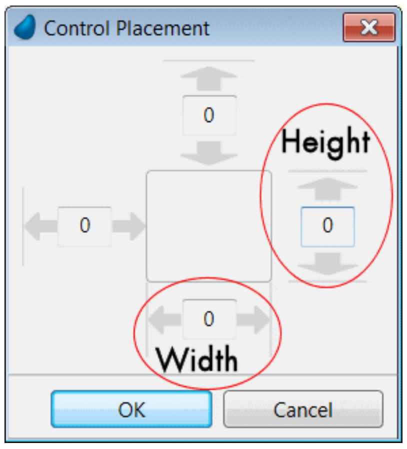
Placement is a property that determines whether or not controls are resized when the form is resized. When a control's **Placement** property equals zero, the relative size of the control does not change when the size of the form is changed in runtime.

When the **Placement** property is larger than zero, the relative size changes proportionally when the size of the form changes in runtime.

The placement of a control is defined by four values.



Navigation	
Placement	{0,0,0,0}
X	41.250
Y	19.250
Width	8.250
Height	1.750
Control's Layer	0



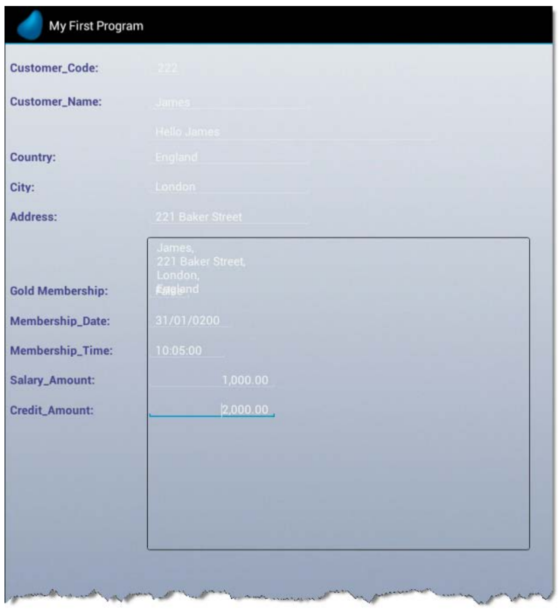
In the **Placement** property, the **Width** and **Height** values determine how the control resizes itself when the form is resized. The value is in percentages. When you set the value as zero, you are defining that the control is not resized. When you set a value of 100, you are saying that the control resizes itself with the form.

As an example, you will increase the height and width of the **Address Label** when the form is increased. To do this:

1. Zoom into **My First Program** and zoom into the Form Designer.
2. Park on the **Address** Label control and open the control property sheet.
3. Click on the **Placement** property and then click the Zoom button.
4. Set the **Width** property to **100** in and the **Height** property to **100**.
5. Click **OK**. The **Placement** property will show: **0,100,0,100**.
6. Execute the program.

You can see from the image below that the **Address Label**'s size increased both horizontally and vertically. The problem is that the controls below the **Address Label** are distorted because the controls are displayed one on top of the other. The other controls need to be moved.

Android



My First Program

Customer_Code: 222

Customer_Name: James

Hello James

Country: England

City: London

Address: 221 Baker Street

James,
221 Baker Street,
London,
England

Gold Membership: False

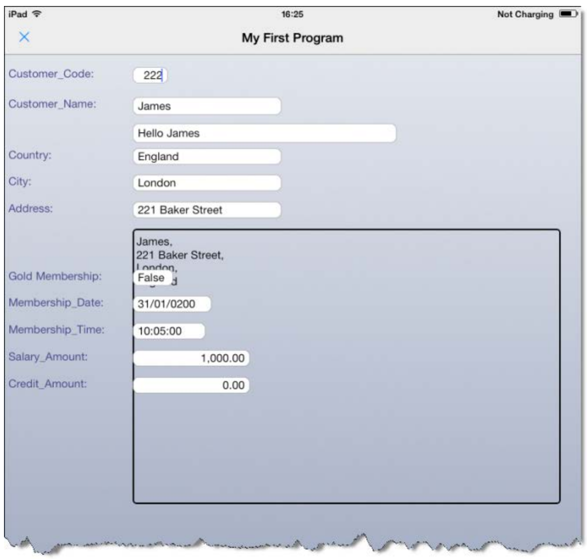
Membership_Date: 31/01/0200

Membership_Time: 10:05:00

Salary_Amount: 1,000.00

Credit_Amount: 2,000.00

iOS



iPad 16:25 Not Charging

My First Program

Customer_Code: 222

Customer_Name: James

Hello James

Country: England

City: London

Address: 221 Baker Street

James,
221 Baker Street,
London,

Gold Membership: False

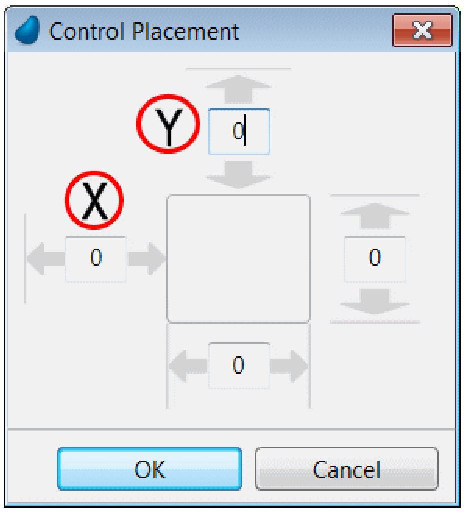
Membership_Date: 31/01/0200

Membership_Time: 10:05:00

Salary_Amount: 1,000.00

Credit_Amount: 0.00

The **X** and **Y** values of the placement solve the problem. They determine how a control moves when the form is resized. The value is also in percentages. When you set the value as zero, you are defining that the control stays in place. When you set a value of 100, you are saying that the control moves with the form for its full placement.



You will define that the **Gold Membership** control moves according to the placement. This means that it will always appear after the **Address Label,** no matter how much that control's height grows. To do this:

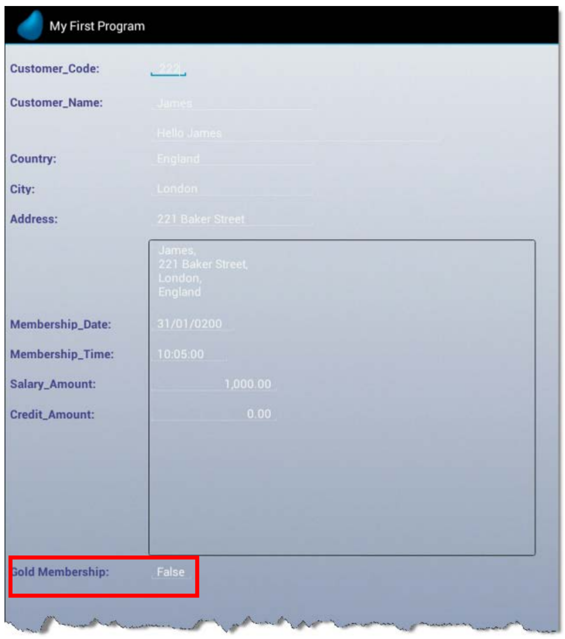
1. Zoom into **My First Program** and zoom into the Form Designer.
2. Park on the **Gold Membership** control and open the control property sheet.
3. Click on the **Placement** property and then click the Zoom button.
4. Set the **Y** property to **100**.
5. Click **OK**. The **Placement** property will show: **0,0,100, 0**.

Remember that if you change the Edit control, its label must move with it:

6. Park on the **Gold Membership** Label control and open the control property sheet.
7. Click on the **Placement** property and then click the Zoom button.
8. Set the **Y** property to **100**.
9. Click **OK**. The **Placement** property will show: **0,0,100, 0**.

Execute the program.

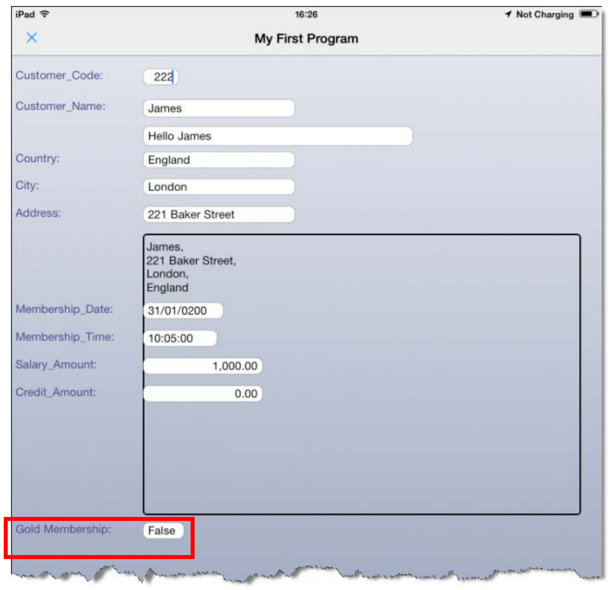
Android



My First Program

Customer_Code: 222
Customer_Name: James
Hello James
Country: England
City: London
Address: 221 Baker Street
James,
221 Baker Street,
London,
England
Membership_Date: 31/01/0200
Membership_Time: 10:05:00
Salary_Amount: 1,000.00
Credit_Amount: 0.00
Gold Membership: False

iOS



iPad 16:26 Not Charging

My First Program

Customer_Code: 222
Customer_Name: James
Hello James
Country: England
City: London
Address: 221 Baker Street
James,
221 Baker Street,
London,
England
Membership_Date: 31/01/0200
Membership_Time: 10:05:00
Salary_Amount: 1,000.00
Credit_Amount: 0.00
Gold Membership: False

As you can see from the image above, the **Address Label** increased in both width and height and the **Gold Membership** control moved accordingly.

Exercise

Now you will practice what you learned during this lesson.

1. Define a new color, **Edit controls** (Green = 90, Blue = 146). This text is an example of what it will look like. Remember that the background must be transparent.
2. Change the color of all the Edit controls to Edit control.
3. Change the font of all the Edit controls to **Text**.
4. Define the placement of **Membership_Date**, **Membership_Time**, **Salary_Amount**, **Credit_Amount** so that they move together with the **Gold Membership**.
5. Define **Customer_Name**, **welcome customer** and **Address** controls so that their width increases as the form increases but they remain in place.

Make the color and font files specific for this project.

6. Copy the color and font file to your project directory under the folder named **Env** and set the color and font files in the **Application properties** using the **%WorkingDir%** logical name.

You will not see any changes in your project. However, from now on, all changes that you make to the color or the font files will apply to the files belonging to your project and not to the generic files in the **Support** directory.

Summary

This lesson introduced you to some of Magic xpa's form design options.

You learned how to define colors and fonts to be used within your project.

You also learned how to change a control's color, font, size, location, and style.

You then learned about setting wallpaper for a form and about additional form appearance properties.

You were introduced to the Logical Names option and used the **%WorkingDir%** logical name.

You learned about placement. Remember that when resizing the form:

- Defining values of X=100 and Y=100 will keep the control in the bottom right position.
- Defining values of Width=100 will **resize** the control horizontally. This is mostly used when the control size is smaller than its content.
- Defining values of X=100 will **move** the control horizontally. This is mostly used when this control is displayed after a control that has a Width placement of 100%.

Lesson 7

Viewing Data Source Content

Saving data is a fundamental requirement for a data-related application. Magic xpa supports the saving of data to various databases, such as: Oracle, DB2, Microsoft SQL (MSSQL), Pervasive, and ODBC.

This course uses SQLite as the database. To work with other SQL databases, refer to the Magic xpa documentation.

Up until now you created programs with **Virtual** variables. Virtual variables exist as long as the program is running. This means that data has to be entered every time the program is executed. There is no stage where you can save Virtual variables.

In this lesson you will learn how to create an entry (a data source) in the Data repository.

This lesson covers various topics including:

- Data sources
- Program Generator utility
- Screen mode and Line mode display
- Table controls

Defining the Database

About the Database Repository

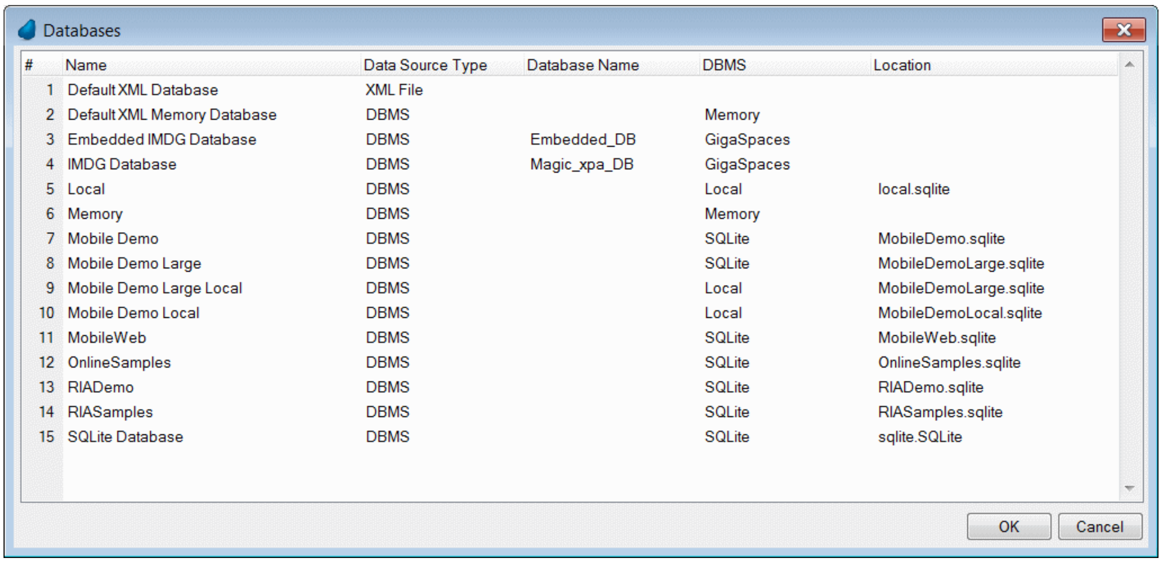
The first step in defining a data source is to define the database in which the data source is located.

Magic xpa enables you to connect to multiple database management systems (DBMSs), such as Btrieve, MSSQL, and Oracle.

The Database repository contains details about all of the physical databases that can be accessed in the current installation of Magic xpa.

In the Database repository you can define multiple entries that point to different databases or to the same database. Each entry in the Database repository contains information regarding the DBMS, including the database name, user and password, and additional details, which provide Magic xpa sufficient information to connect to this database later on.

By default, there are some predefined databases that are created automatically in the Database repository.



Databases

#	Name	Data Source Type	Database Name	DBMS	Location
1	Default XML Database	XML File			
2	Default XML Memory Database	DBMS		Memory	
3	Embedded IMDG Database	DBMS	Embedded_DB	GigaSpaces	
4	IMDG Database	DBMS	Magic_xpa_DB	GigaSpaces	
5	Local	DBMS		Local	local.sqlite
6	Memory	DBMS		Memory	
7	Mobile Demo	DBMS		SQLite	MobileDemo.sqlite
8	Mobile Demo Large	DBMS		SQLite	MobileDemoLarge.sqlite
9	Mobile Demo Large Local	DBMS		Local	MobileDemoLarge.sqlite
10	Mobile Demo Local	DBMS		Local	MobileDemoLocal.sqlite
11	MobileWeb	DBMS		SQLite	MobileWeb.sqlite
12	OnlineSamples	DBMS		SQLite	OnlineSamples.sqlite
13	RIADemo	DBMS		SQLite	RIADemo.sqlite
14	RIASamples	DBMS		SQLite	RIASamples.sqlite
15	SQLite Database	DBMS		SQLite	sqlite.SQLite

As you can see from the image above, additional databases exist for sample applications. These are installed optionally by the Magic xpa installation.

Defining the GettingStarted Database

The course is provided with a **GettingStarted** SQLite database and course-related tables and data. This database is located in the **data** folder. In the previous lesson, you were asked to copy folders, including the **data** folder, from the **CourseData** directory into the **Getting Started** directory.

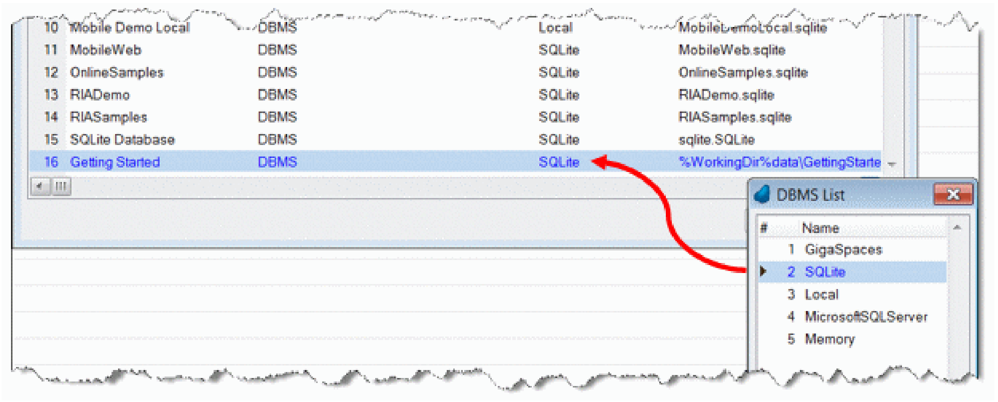
To access those tables, you need to define a connection to the **SQLite** entry in the Database repository.

To open the Database repository:

1. Verify that no projects are open.
2. If your project is open, select **Close Project** from the **File** menu.
3. From the **Options** menu, select **Settings** and then **Databases**.

In the Database repository:

4. Park on any line, such as the last line, and create **(F4)** a database entry.
5. In the **Name** column, type **Getting Started**.
6. In the **Data Source Type** column, make sure that **DBMS** is selected.
7. The default for the **Database Name** is **MAGIC**. Clear this entry. There is no Database name in SQLite.
8. Zoom from the **DBMS** column to open the **DBMS List**, park on the **SQLite** entry, and click the **Select** button (or press **Enter**).
   Note: This may already have been selected.
9. The **Location** column shows the exact path to the database data. In the previous lesson you learned about the **%WorkingDir%** logical name. Set the **Location** to:
   %WorkingDir%data\GettingStarted.dat



10	Mobile Demo Local	DBMS	Local	MobileDemoLocal.sqlite
11	MobileWeb	DBMS	SQLite	MobileWeb.sqlite
12	OnlineSamples	DBMS	SQLite	OnlineSamples.sqlite
13	RIADemo	DBMS	SQLite	RIADemo.sqlite
14	RIASamples	DBMS	SQLite	RIASamples.sqlite
15	SQLite Database	DBMS	SQLite	sqlite.SQLite
16	Getting Started	DBMS	SQLite	%WorkingDir%data\GettingStarte

DBMS List

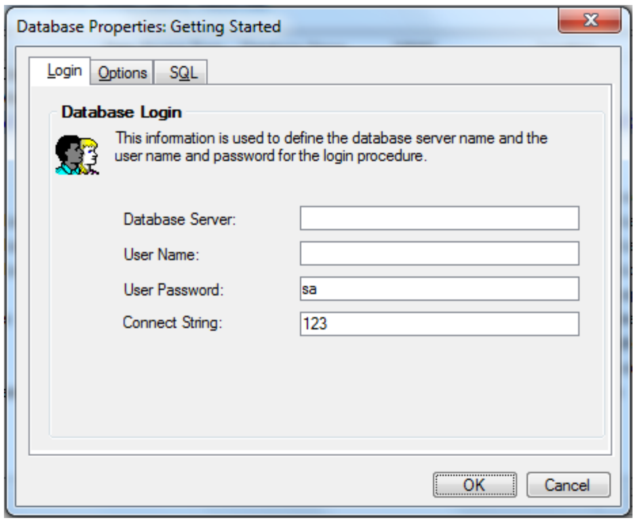
#	Name
1	GigaSpaces
2	SQLite
3	Local
4	MicrosoftSQLServer
5	Memory

Setting Database Properties

The **Database Properties** dialog box contains three tabs:

- **Login** tab – details regarding the login procedure (such as user name and password)
- **Options** tab – information regarding the connection between Magic xpa and the underlying database
- **SQL** tab – data regarding the SQL connectivity

For the SQLite database, you do not need these properties. The image below shows an example when using an MSSQL database.


Database Properties: Getting Started

Database Login

This information is used to define the database server name and the user name and password for the login procedure.

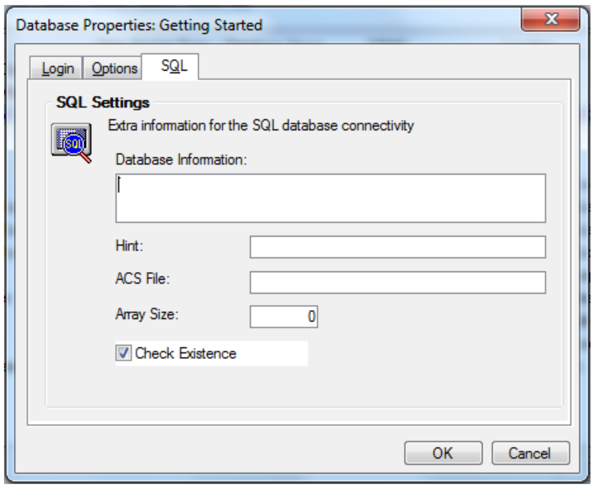
Database Server:	
User Name:	
User Password:	sa
Connect String:	123

Check Existence

When you execute a program that displays table content, Magic xpa sends a request to the DBMS (where the table is located) specifying the required information. If the table does not exist in the database (this is not referring to an empty table), an error message will be returned from the underlying DBMS. In most cases when this happens, the developer wants Magic xpa to create the required table (according to the table structure in the Data repository).

Magic xpa can check for the table's existence and create the table in the underlying database when necessary. (The checking of the table's existence and creating the table occurs once per program.) A specific property located in the **SQL** tab of the **Database Properties** dialog box, supports this behavior, the **Check Existence** property.

1. Open the **Getting Started** Database Properties dialog box (**Alt+Enter**).
2. Click the **SQL** tab.
3. Select the **Check Existence** check box.
4. Click **OK** to close the **Database Properties** dialog box.


Database Properties: Getting Started

SQL Settings
Extra information for the SQL database connectivity
Database Information:
Hint:
ACS File:
Array Size: 0
[✓] Check Existence

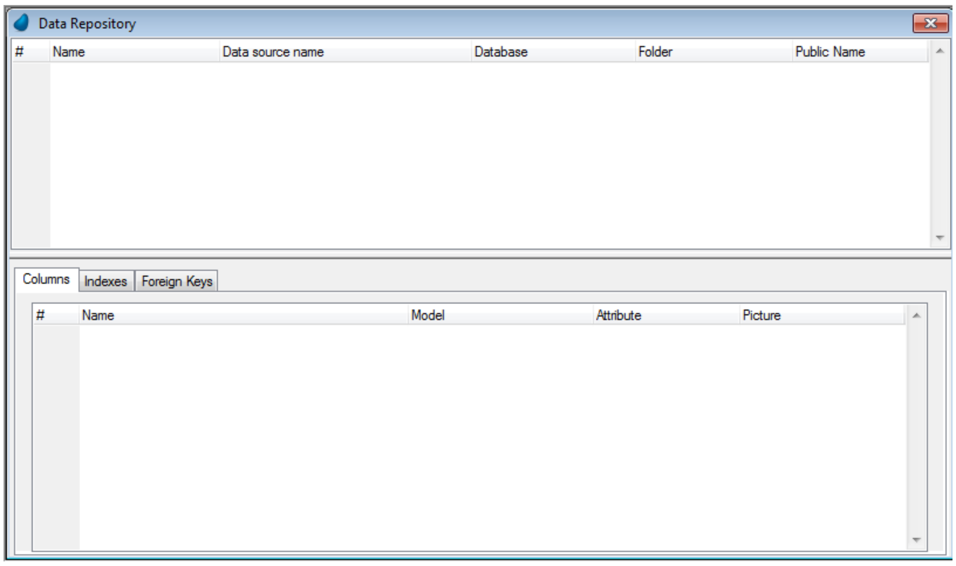
Defining a Data Source

The next step is to define the data source. Defining a data source is divided into the following main steps:

- Defining the data source header
- Defining columns
- Defining indexes
- Defining foreign keys. This is not in the scope of this course.

The Data repository is divided into two sections:

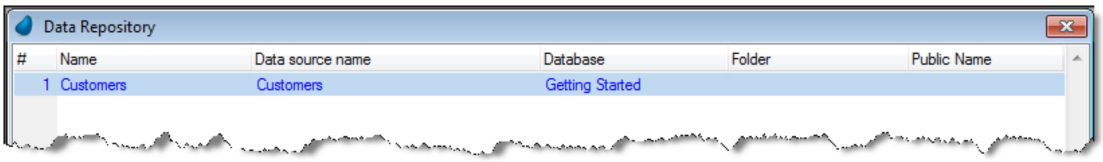
- The upper pane contains basic properties of the data source, including its name, data source name, database, folder, and public name.
- The lower pane displays three repositories: columns, indexes, and foreign keys. The displayed content in the lower pane changes according to the selected data source in the upper pane.



Defining the Customers Data Source

Now, you will define the **Customers** data source.

1. From the **Project** menu, select **Data** (**Shift+F2**) to open the Data repository.
2. Create a line.
3. In the **Name** column, type **Customers**.
4. In the **Data source name** column, type **Customers**.
5. From the **Database** column, zoom to the **Database List**, and select the **Getting Started** entry.

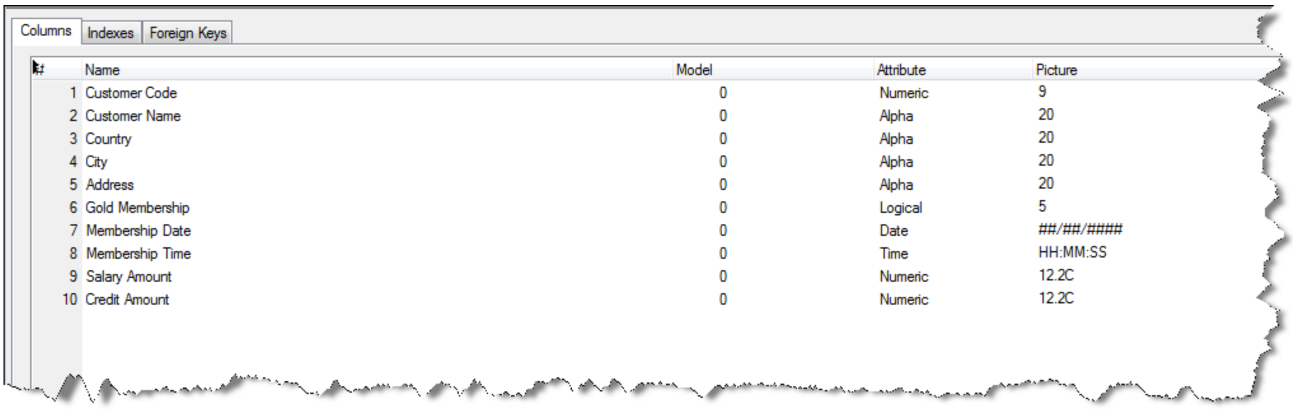


#	Name	Data source name	Database	Folder	Public Name
1	Customers	Customers	Getting Started		

Defining Columns

The Column repository contains the column definitions of the data source. Each data source has its own Column repository. You define columns for a data source in the same way you defined Virtual variables in the data view of the program. For each column you can set a name, model, attribute, and picture.

6. Park on the **Customers** data source in the upper pane.
7. Click the **Columns** tab in the lower pane.
8. Define column entries for the items **exactly** as shown in the image below:

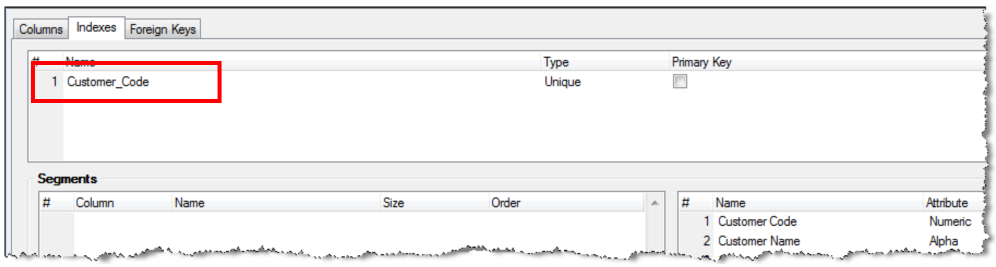


#	Name	Model	Attribute	Picture
1	Customer Code	0	Numeric	9
2	Customer Name	0	Alpha	20
3	Country	0	Alpha	20
4	City	0	Alpha	20
5	Address	0	Alpha	20
6	Gold Membership	0	Logical	5
7	Membership Date	0	Date	##/##/####
8	Membership Time	0	Time	HH:MM:SS
9	Salary Amount	0	Numeric	12.2C
10	Credit Amount	0	Numeric	12.2C

Defining an Index

Now you will define a unique index for the **Customers** data source.

9. Click the **Indexes** tab in the lower pane.
10. Create a line and define an index as shown in the image below:



Columns | Indexes | Foreign Keys

#	Name	Type	Primary Key
1	Customer_Code	Unique	☐

Segments

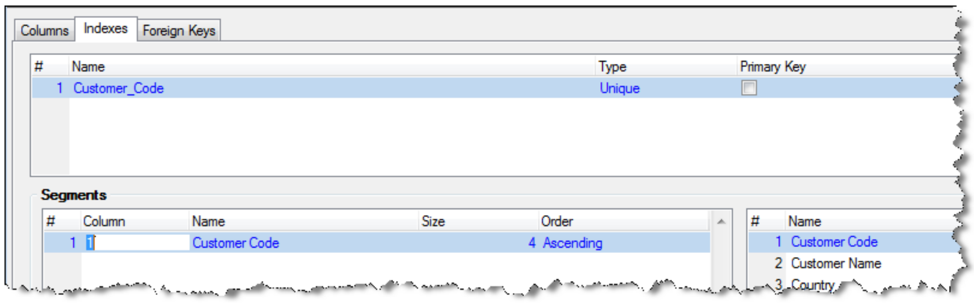
#	Column	Name	Size	Order

#	Name	Attribute
1	Customer Code	Numeric
2	Customer Name	Alpha

A Unique Index is an index where only one record with a specific value in the index can be present in the table. **Note:** Primary keys will not be discussed in this course.

Now, you will define the index segment.

11. From the **Customer_Code** Index name, zoom to the Segments repository.
12. Create a line in the Segments repository.
13. Zoom from the **Column** column. The cursor will move to the Column list (right-hand pane).
14. Park on the **Customer_Code** entry and press **Enter** to select the **Customer_Code** column. The **Customer_Code** will appear as the index segment in the Segments repository.



Columns | Indexes | Foreign Keys

#	Name	Type	Primary Key
1	Customer_Code	Unique	☐

Segments

#	Column	Name	Size	Order
1	1	Customer Code		4 Ascending

#	Name
1	Customer Code
2	Customer Name
3	Country

15. Click the upper pane to return to the Data Source definition.
16. In the Confirmation dialog box, select **Yes** to save your work.

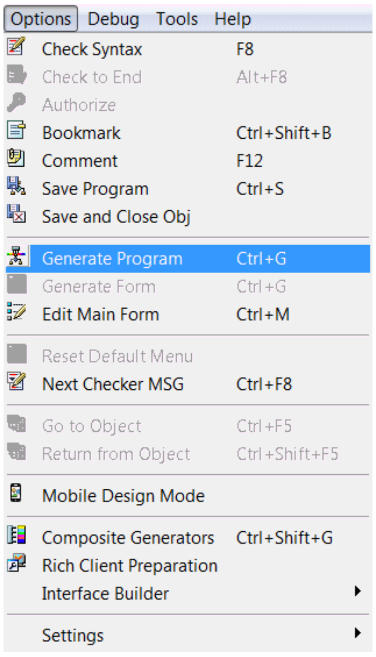
You have just finished defining the **Customers** data source.

Automatic Program Generation

Displaying the table content is performed in a program similar to the one you recently created. Up until now you created programs that display and handle Virtual variables.

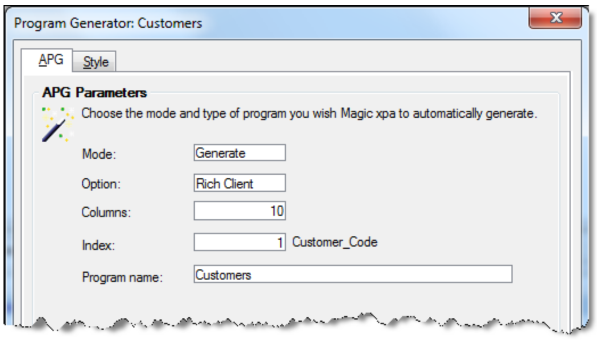
You will now use a handy tool known as the Generate Program utility to create a program that displays the **Customers** data source's content.

1. In the Data repository, park on the **Customers** data source.
2. From the **Options** menu, select **Generate Program** (**Ctrl+G**).



Check Syntax	F8
Check to End	Alt+F8
Authorize	
Bookmark	Ctrl+Shift+B
Comment	F12
Save Program	Ctrl+S
Save and Close Obj	
Generate Program	Ctrl+G
Generate Form	Ctrl+G
Edit Main Form	Ctrl+M
Reset Default Menu	
Next Checker MSG	Ctrl+F8
Go to Object	Ctrl+F5
Return from Object	Ctrl+Shift+F5
Mobile Design Mode	
Composite Generators	Ctrl+Shift+G
Rich Client Preparation	
Interface Builder	
Settings	

3. In the **Program Generator** dialog box, from the **Mode** property, select **Generate**.
   The **Mode** property defines whether the result program will be temporary or permanent. The options are:
   - **Execute** – Generates and executes a temporary program for the selected data sources.
   - **Generate** – Generates programs for selected data sources, and adds the programs to the Program repository. The programs are not automatically executed.



Program Generator: Customers

APG | Style

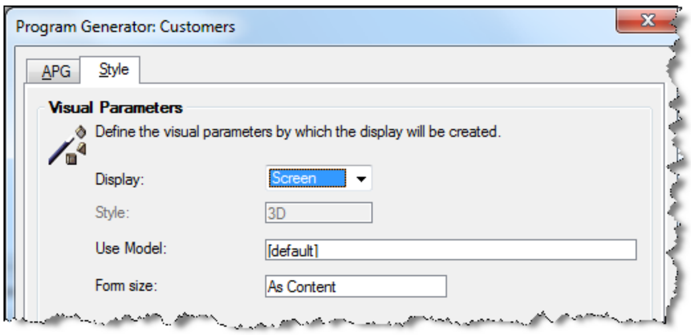
APG Parameters

Choose the mode and type of program you wish Magic xpa to automatically generate.

Mode:	Generate
Option:	Rich Client
Columns:	10
Index:	1 Customer_Code
Program name:	Customers

4. From the **Option** property, select **Rich Client**. This property sets the program's functionality.
5. Set the **Program name** property to **Customers**.

6. Click the **Style** tab.
7. From the **Display** property, select **Screen**.
8. Click **OK**.


Program Generator: Customers

APG | Style

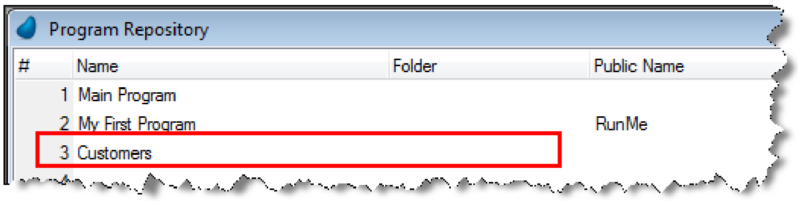
Visual Parameters

Define the visual parameters by which the display will be created.

Display:	Screen
Style:	3D
Use Model:	[default]
Form size:	As Content

The **Program Generator** added a program based on the **Customers** data source to the Program repository.

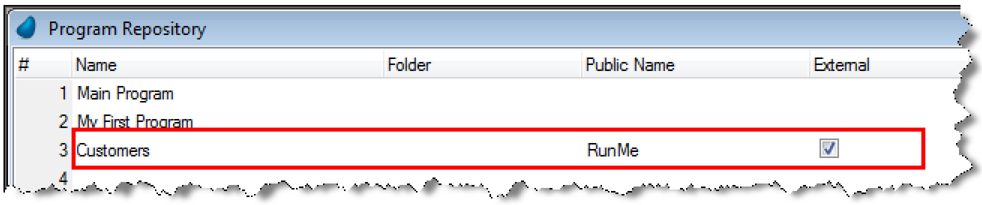
9. Open the Program repository (**Shift+F3**).
10. Verify that the new program was added.


Program Repository

#	Name	Folder	Public Name
1	Main Program		
2	My First Program		RunMe
3	Customers		

Rich Client programs need to have a Public name so that they can be executed externally. At this stage of the course, the Public name is defined as **RunMe**. In a later lesson, you will learn how to customize this:

11. Park on the **My First Program** entry and clear the Public Name definition.
12. Park on the **Customers** entry and set the **Public Name** property to **RunMe**.
13. Check the **External** box.


Program Repository

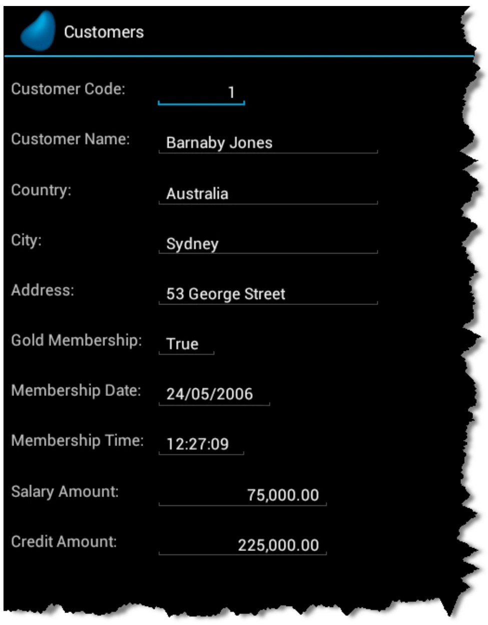
#	Name	Folder	Public Name	External
1	Main Program			
2	My First Program			
3	Customers		RunMe	✓
4				

Running the Customers Program

You now have a program that can display the data from the **Customers** data source.

14. Execute the project and display it on your mobile device.

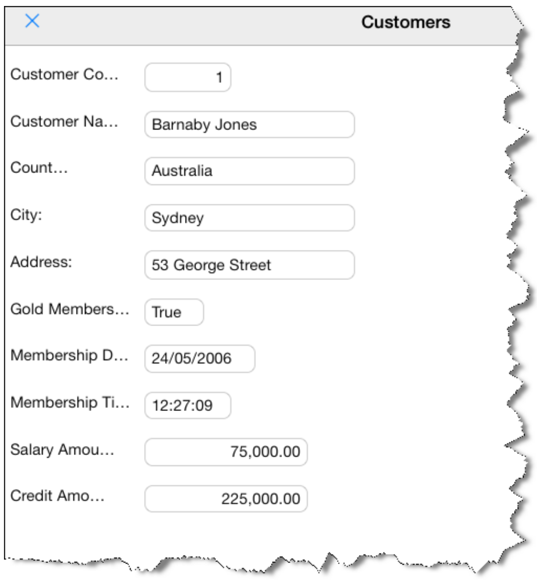
Android



Customers

Customer Code:	1
Customer Name:	Barnaby Jones
Country:	Australia
City:	Sydney
Address:	53 George Street
Gold Membership:	True
Membership Date:	24/05/2006
Membership Time:	12:27:09
Salary Amount:	75,000.00
Credit Amount:	225,000.00

iOS



Customers

Customer Co...	1
Customer Na...	Barnaby Jones
Count...	Australia
City:	Sydney
Address:	53 George Street
Gold Members...	True
Membership D...	24/05/2006
Membership Ti...	12:27:09
Salary Amou...	75,000.00
Credit Amo...	225,000.00

As you can see from the image above, you are provided with a basic program that may or may not suit your needs. You may need to make some changes to the program to make the program more readable. In iOS, the Label controls need to be increased. The Generate Program utility is useful when you need to create a simple program.

Manually Creating the Customers Program

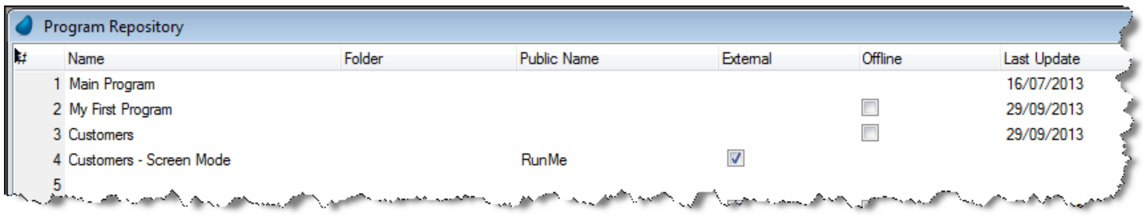
In the previous section you used the Program Generator to create a **Screen** mode program. In this section you will manually create the same program.

A Magic xpa program is created using three main steps:

- Defining the data view.
- Adding logic.
- Designing the form.

In Magic xpa you only need to define the data view and design the form. Magic xpa will provide all of the basic logic that you need to browse the content of a data source.

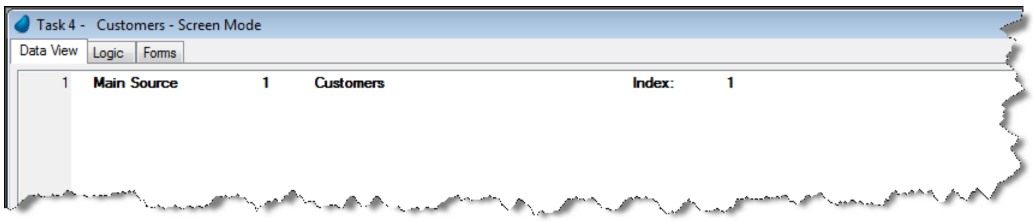
1. Open the Program repository (**Shift+F3**).
2. Create a line and name the program: **Customers - Screen Mode**.
3. As before, park on the **Customers** entry and clear the **Public Name** definition.
4. Park on the **Customers - Screen Mode** entry and set the **Public Name** property to **RunMe**.
5. Check the **External** box.


Program Repository

#	Name	Folder	Public Name	External	Offline	Last Update
1	Main Program					16/07/2013
2	My First Program				☐	29/09/2013
3	Customers				☐	29/09/2013
4	Customers - Screen Mode		RunMe	✓		
5						

Defining a Main Source

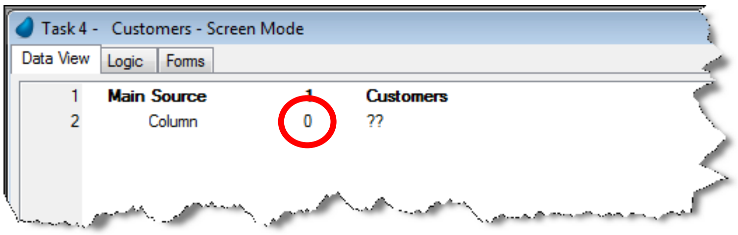
6. Zoom into the **Customers - Screen Mode** program.
7. Define the **Task Type** as **Rich Client**.
8. Click **OK**.
9. In the Data View Editor, park on the **Main Source** definition.
10. In the **Data Source Number** column (the field to the right of the words **Main Source**), zoom to the Data Source list and select the first entry, the **Customers** data source.
    Note: The first index is automatically selected in the **Index** column.


Task 4 - Customers - Screen Mode

1	**Main Source**	**1**	**Customers**	**Index:**	**1**

You will now learn how to select columns for the **Customers** data source.

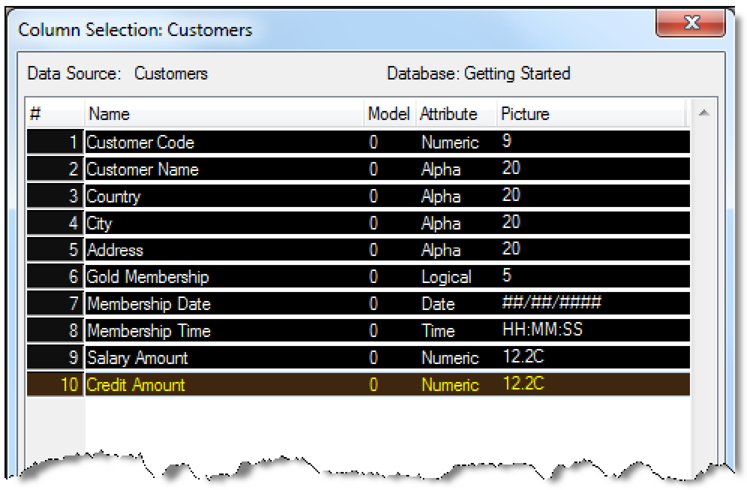
11. Create a line and from the column number (circled in red below), zoom to the **Column Selection** window.



1	Main Source	1	Customers
2	Column	0	??

12. In the **Column Selection** window, select all of the columns using one of the following methods:

- Park on the first line, hold down the **Shift** key, and click on the **#** column of the last line.
- Park on the first line, hold down the **Shift** key, and press the **Down Arrow** key until all 10 rows are marked.

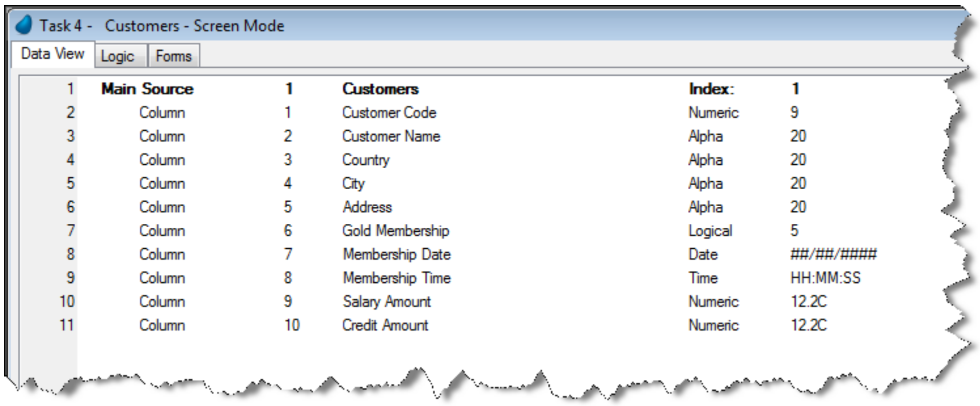


Data Source: Customers Database: Getting Started

#	Name	Model	Attribute	Picture
1	Customer Code	0	Numeric	9
2	Customer Name	0	Alpha	20
3	Country	0	Alpha	20
4	City	0	Alpha	20
5	Address	0	Alpha	20
6	Gold Membership	0	Logical	5
7	Membership Date	0	Date	##/##/####
8	Membership Time	0	Time	HH:MM:SS
9	Salary Amount	0	Numeric	12.2C
10	Credit Amount	0	Numeric	12.2C

13. Click the **Select** button or press **Enter**.

All of the **Customers** columns should now appear as shown in the image below.



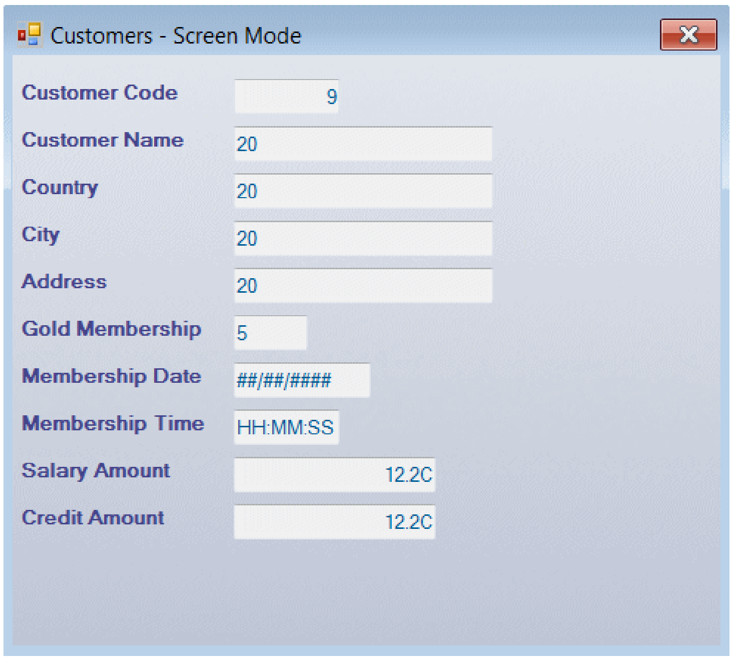
1	**Main Source**	**1**	**Customers**	**Index:**	**1**
2	Column	1	Customer Code	Numeric	9
3	Column	2	Customer Name	Alpha	20
4	Column	3	Country	Alpha	20
5	Column	4	City	Alpha	20
6	Column	5	Address	Alpha	20
7	Column	6	Gold Membership	Logical	5
8	Column	7	Membership Date	Date	##/##/####
9	Column	8	Membership Time	Time	HH:MM:SS
10	Column	9	Salary Amount	Numeric	12.2C
11	Column	10	Credit Amount	Numeric	12.2C

Designing the Form

Now that you have completed defining the data view, you now need to design the task form. First, you will set wallpaper for the form.

1. Open the Form Editor.
2. Verify that you are parked on the form for this task.
   Note: The first entry in the list is the form from the Main Program.
3. Open the **Form Properties** sheet and in the **Wallpaper** property, type:
   %WorkingDir%env\Wallpaper.jpg
4. Zoom into the Form Designer.
5. From the Task Variables pane, drag the **Customer_Code** variable and drop it on the form.
6. Repeat step 5 for the rest of the variables.
7. Change the Font of all the **Label** controls to **Text Caption**.
8. Change the Color of all the **Label** controls to **Text Caption**.
9. Fit the size of all the **Caption** controls to the displayed text.
10. Move the **Field** controls, so that the entire value of the Label controls will be displayed.

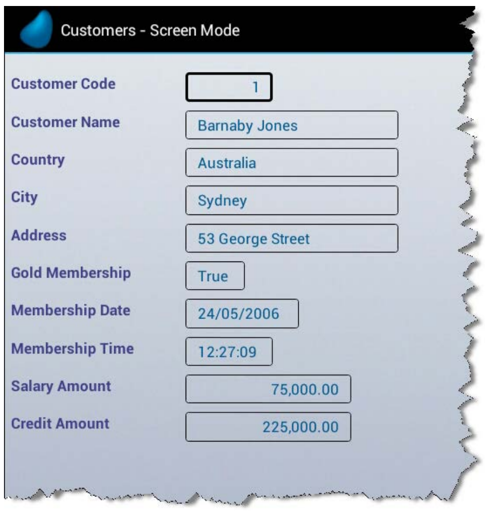
11. Change the Color of all of the **field** controls to **Edit control**.



Customer Code	9
Customer Name	20
Country	20
City	20
Address	20
Gold Membership	5
Membership Date	##/##/####
Membership Time	HH:MM:SS
Salary Amount	12.2C
Credit Amount	12.2C

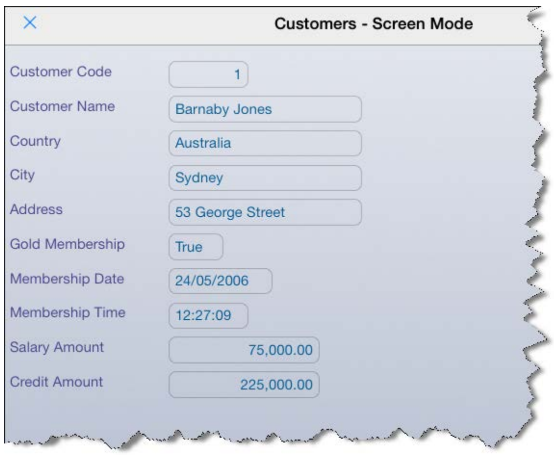
12. Close the program and save the changes.
13. Execute the project and display it on your mobile device.

Android



Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

iOS



Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

As you can see, only one record is visible.

Short Summary

Up until now you created a data source (table) in the Data repository and created two programs that browse the table content. The first program was created using the Program Generator; the second one was manually created.

During the data source creation, you were introduced to the Data repository, which is divided into two panes:

- Upper pane: where you define the data source name and its underlying database information.
- Lower pane: which is divided into three repositories:
    - Columns: where you define the Data Source columns and its properties, such as type and picture.
    - Indexes: where you define the Data Source indexes and their segments.
    - Foreign keys: where you define external (other tables') indexes that are related to specific columns in the current table. This feature is used to maintain data integrity and it will not be discussed in this course.

You learned how to create a program that browses the content of a table. You saw that development is very similar to the use of Virtual variables.

In the next section you will learn how to display more than one record on the form.

Viewing Several Records

The manner in which you display the information on the form serves a purpose.

- **Screen mode** – In this mode, only one record is displayed on the screen at a time. Screen mode is usually used to display detailed information when the end user needs to focus on a specific object and needs the maximum amount of information on the screen.
- **Line mode** – In this mode, several records are displayed on the screen at a time. Line mode is usually used to display key information when the end user needs to search within a collection of records and needs minimum information for a quick view.

Task Mode

During this course you used a screen mode form, which enabled you to enter data directly into a field. When you tapped the field, you were immediately able to edit the field. This is known as a modify mode. However, when you display multiple records such as a customer list you may not want your user to edit the data but simply browse it. In Magic xpa tasks, there is a mode of operation known as **Task Mode**. The task's mode of operation determines which data manipulations are allowed on the data. Magic xpa allows four modes of operation:

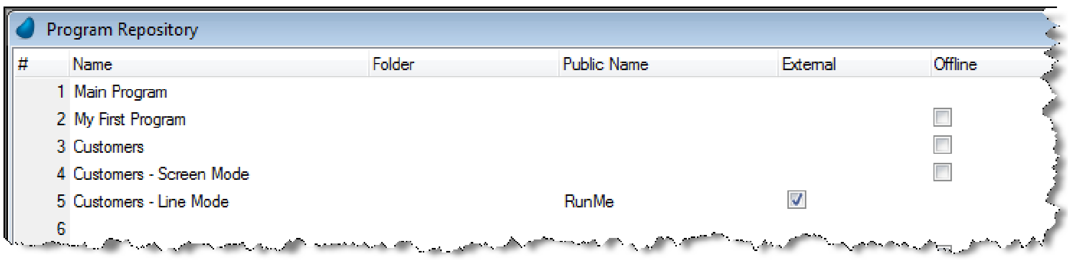
Mode of Operation	Allowed Operations
Create	This mode allows the end user to create new records. When you are in Create mode you can only see the records that you have added in the current session. To view all of the records, you need to switch to Query or Modify mode.
Modify	This mode allows the end user to edit existing records and delete records. When you are in Modify mode, you can also create records. This is a specific mode in Magic xpa called **Create in Modify**.
Query	This mode allows the end user to scan through records, without updating the records or deleting them.
Delete	This mode is part of the Modify mode. This mode deletes the current record.

The developer can specify the task's initial mode as well as the allowed modes.

Creating a Line Mode Program

You will now create a Line Mode program.

1. Open the Program repository and create a program called: **Customers - Line Mode**.
2. As before, park on the **Customers - Screen Mode** entry and clear the **Public Name** definition.
3. Park on the **Customers - Line Mode** entry and set the **Public Name** property to **RunMe**.
4. Check the **External** box.



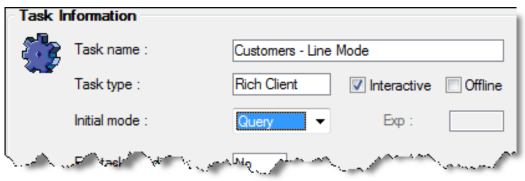
Program Repository

#	Name	Folder	Public Name	External	Offline
1	Main Program				
2	My First Program				☐
3	Customers				☐
4	Customers - Screen Mode				☐
5	Customers - Line Mode		RunMe	✓	
6					

Defining the Task Mode

Set the Task Mode as follows:

5. Zoom into the **Customers - Line Mode** program.
6. In the **Task type** property, select **Rich Client**.
7. Set the **Initial mode** property to **Query**. Users will only be allowed to scroll through the data without being able to edit it.



Task Information

Task name : Customers - Line Mode

Task type : Rich Client ✓ Interactive ☐ Offline

Initial mode : Query Exp :

Defining a Main Source

Create the data view as you did in the previous section. In the Data View Editor:

8. Park on the **Main Source** definition and select the **Customers** data source and the first index.
9. Select all of the **Customers** columns.

Designing the Form

Now that you have completed defining the data view, you now need to design the task form.

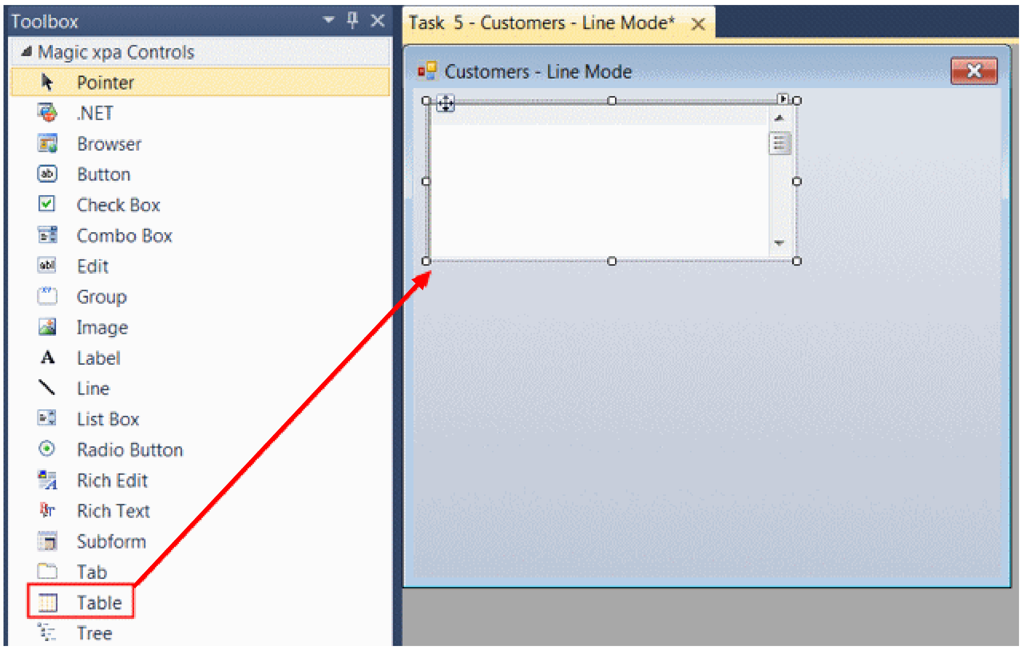
Set the wallpaper for the form as you did before:

10. Open the Form Properties sheet and in the **Wallpaper** property, type: **%WorkingDir%env\Wallpaper.jpg**
11. Zoom into the Form Designer.

Placing a Table control on the form

In the Form Designer:

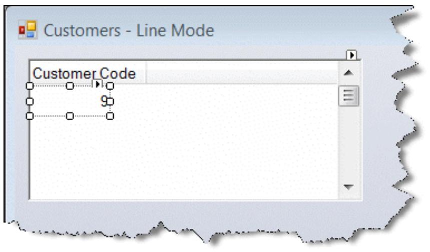
12. Zoom to the **Customers - Line Mode** form.
13. From the Toolbox, click on the Table control.
14. Drop the Table control on the form as shown in the image below.



Only one Table control is allowed on a form. Therefore, you will not be able to drag more than one onto the form.

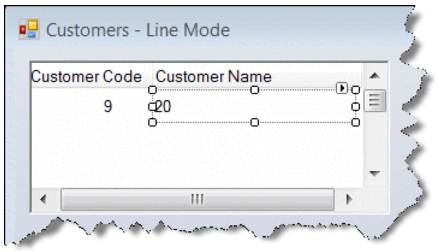
Attaching variables to the Table control

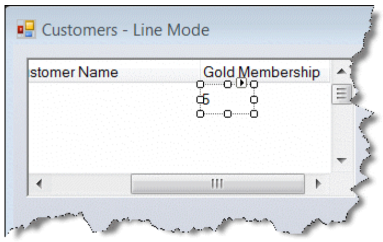
15. From the **Task Variables** pane, drag and drop the **Customer_Code** variable onto the Table control.
16. In the control properties, set the color to the **Edit control** color. This will provide a similar color to the screen mode form.



Now you will add two more variables to the table:

17. Place the following variables on the table:
    - Customer_Name
    - Gold_Membership





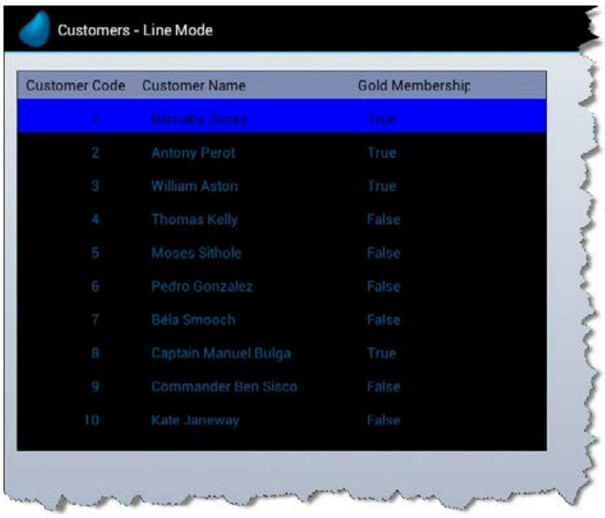
18. In the control properties of all three Edit controls, set the color to the **Edit control** color.

Expanding the table size

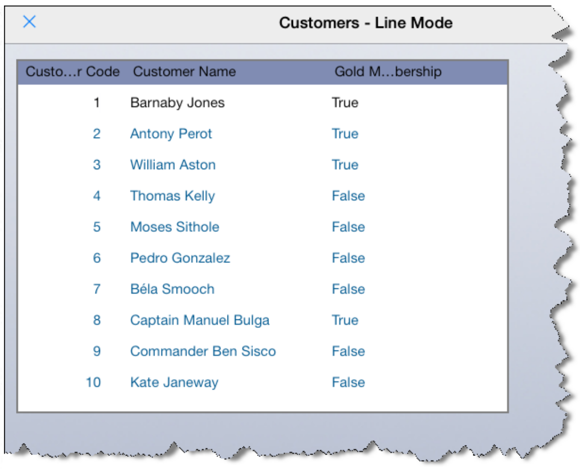
When attaching controls to the table, the table does not automatically resize. Moreover, the table height determines the number of the displayed rows.

19. Select the Table control and increase the table width and the table height.
20. Close the program and save the changes.
21. Execute the project and display it on your mobile device.

Android iOS


Customers - Line Mode

Customer Code	Customer Name	Gold Membership
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False


Customers - Line Mode

Custo…r Code	Customer Name	Gold M…bership
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False

As you can see, the customers are displayed in table format and more than one record is visible. You can move to a different record by tapping the relevant record. You can see that in the different mobile devices, there are some differences. In the Android device, the system default for the table color is Black and on the iOS device, column titles are truncated so that you are unable to read the full text.

The use of colors for your program depends on your own form design. You may decide that the defaults provided by the operating system suit your needs or you may decide that you want to provide a consistent look and feel for your application across all platforms. It is your choice. During this course you will be developing a similar look and feel for all the devices.

Changing the Table's Look and Feel

To change the table to provide a more unified look among all operating systems you will do the following:

- Change the color of the table.
- Change the color of the row highlighting line. You will add a color to support this.
- Change the text of the column header.

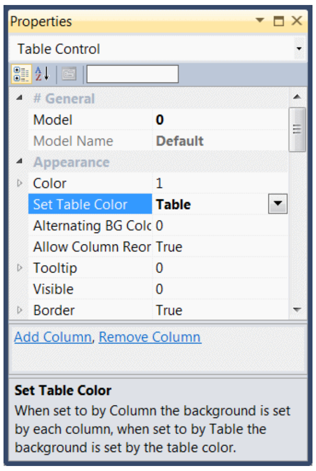
The first step will be to add a new color. All of the other actions are defined in the program's Form Editor.

1. From the **Options** menu, select **Settings** and then **Colors**.
2. Create a line.
3. In the **Name** column, type: **Row Highlight**.

4. Zoom from the **FG** column. Select the first (empty) entry from the **System** drop-down list. Define the color as Blue (Blue=255).
5. Zoom from the **BG** column. Select the first (empty) entry from the **System** drop-down list. Define the color as Blue (Red=160, Green=190, Blue=230).
6. In the **Text: BG** window, select the first (empty) entry from the **System** drop-down list.
7. Click **OK** to close the Color repository. A **Save As** dialog box will appear. From the **Effective immediately** property, select **Yes**.

Now you will change the table properties on the form:

8. Zoom into the **Customers - Line Mode** program.
9. Zoom into the Form Designer and park on the Table control.
10. Open the control properties sheet.
11. In the **Appearance** section, set the **Set Table Color** property to **Table**. You will see that the **Color** property above it is now accessible.
12. Zoom from the **Color** property and select the **Text** color.
13. Zoom from the **Row Highlight Color** property and select the color you defined previously, **Row Highlight**.



Properties

Table Control

# General	
Model	0
Model Name	Default
Appearance	
Color	1
Set Table Color	Table
Alternating BG Colc	0
Allow Column Reor	True
Tooltip	0
Visible	0
Border	True

Add Column, Remove Column

Set Table Color
When set to by Column the background is set by each column, when set to by Table the background is set by the table color.

You have now defined a uniform look across the platforms. The next stage is to change the text for the column headings:

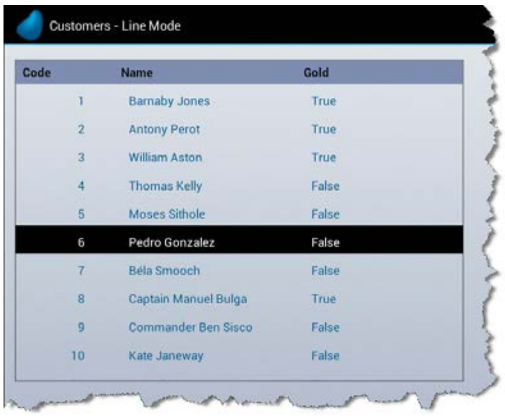
14. Return to the Table control.
15. In the **Customer_code** column, click on the column header. This will select the **Column** control.
16. Zoom into the control properties sheet for the Column control.
17. Park on the **Column title** property and change the name to **Code**.
18. Park on the **Color** property and select the **Text Caption** color.
19. Park on the **Font** property and select the **Text Caption** font.

You will now do the same for the **Customer_Name** and **Gold_Membership** columns:

20. In the **Customer_Name** column, press click on the column header.
21. Open the control properties sheet for the Column control.
22. Park on the **Column title** property and change the name to **Name**.
23. Park on the **Color** property and select the **Text Caption** color.
24. Park on the **Font** property and select the **Text Caption** font.
25. In the **Gold_Membership** column, pressclick on the column header.

26. Open the control properties sheet for the Column control.
27. Park on the **Column title** property and change the name to **Gold**.
28. Park on the **Color** property and select the **Text Caption** color.
29. Park on the **Font** property and select the **Text Caption** font.
30. Set the **Width** property to **11**. This will decrease the size of the column but not affect the size of the **Gold_Membership** Edit control, which is part of this column.
31. Close the program and save the changes.
32. Execute the project and display it on your mobile device.

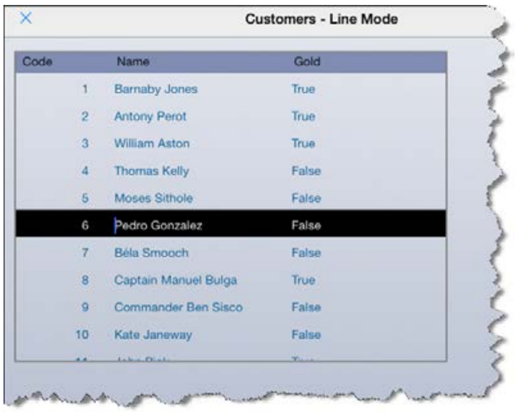
Android



Customers - Line Mode

Code	Name	Gold
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False

iOS



Customers - Line Mode

Code	Name	Gold
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False

Exercise – Suppliers Line Mode Program

Now you will practice some of what you have learned:

1. Define the table in the **Customers - Line Mode** program so that when the size of the form increases so will the table.

Now you will practice working with data sources:

2. Define a **Suppliers** data source in the Data repository. Base it on the **Getting Started** database.
3. Define the following columns:

Name	Model	Attribute	Picture
Supplier_Code	0	Numeric	9
Supplier_Name	0	Alpha	20
Phone_Number	0	Alpha	###-#######
Address	0	Alpha	20
Years_Since_Start_Working	0	Numeric	2
Bonus	0	Numeric	3.2

4. Define a unique index called **Supplier_Code** with a segment for the **Supplier_Code** column.
5. Create a program and name it: **Suppliers - Line Mode**.
6. Use the **Suppliers** data source as the program's Main Source.
7. Select all of the **Suppliers** data source columns.

Define the **Suppliers - Line Mode** form as follows:

8. Provide wallpaper for the form.
9. Display the **Supplier_Code** and the **Supplier_Name** in a table.
10. Change the heading of the **Supplier_Code** column to **Code**.
11. Use colors and fonts as you did during the lesson.

Summary

In this lesson, you learned about data sources, including defining data sources as well as viewing and manipulating the data source content.

You learned about the Database repository, where you defined a database based on the **SQLite** DBMS.

You defined the **Customers** and **Suppliers** data source in the Data repository.

You learned how to use the **Generate Program** utility to create a Screen mode programs. You can use the same tool to create a Line mode program.

You manually created Screen mode and Line mode programs.

You learned how to use a Table control within your programs.

You also learned about some other properties to improve the look and feel of your table.

Lesson 8

Models

A Magic xpa model is a set of properties that can be used to define a class of objects.

The use of models is optional, but using them will benefit you throughout the development and maintenance of the project.

The advantages of using models include saving development time, ensuring consistency throughout the project, and making project maintenance easier.

Up until now, you created the project without taking into account the issues mentioned above.

In this lesson you will learn about the Magic xpa Model repository, how to create a model, and how to attach a model to an object.

The knowledge and tools you will learn about in this lesson will help you create a more efficient and organized project.

This lesson covers various topics including:

- Model repository
- Model types
- Field models
- Rich Client Display models
- Model associations
- Inheritance mechanism

What Is a Model?

A model is a set of properties that can be inherited by an object.

When an object is associated with a model, the value of each object property that has not been defined, inherits the value of the model.

When a property value is defined for an individual object, the inheritance for that value is considered "broken". The defined value overrides the model's property value.

When properties of a model are updated, the change is automatically reflected in all of the associated object's properties.

Advantages to Defining Models

When you plan the object items required for a specific project, consider whether they can be grouped together based on common attributes.

In the Model repository, Magic xpa allows you to define these groups of objects by defining a model for each group; that is, a prototype class having all of the group's common qualifiers.

The use of model definitions is optional, but using them will benefit you throughout the development and maintenance of your projects.

Some advantages to using the Model repository are:

- **Time savings for project development**. Once you have created an object model, you no longer have to set the same property values for other objects.
- **Ease of maintenance**. Once you have defined the properties associated with a specific model, any modification to the model is automatically inherited by all of its associated objects.
- **Ensure matching of columns in different data sources for Field models**. When columns are compared for link purposes, or passed as parameters, their attributes must match exactly. A good way to ensure that they will match is to define them with the same model.

Examples

Field Model

Assume a **Customer_Code** object is used widely in the project. You can set the object properties such as Attribute property (Numeric) and Picture property (9 digits) once in the Model repository and then use this model throughout the project. In addition, if you need to modify the object's picture, for example from 9 digits to 10 digits, you only have to do it once in the Model repository. Magic xpa will update all of the corresponding objects. You could also define the color and font of this object.

Form Model

Assume that in your project all of the forms are uniform and have a wallpaper image in the background. It is time consuming to repeatedly set these properties for each new form. Using a model, you can inherit all of these properties without the need to set them, and without having to remember what you want the form to look like or how to set it. In addition, if you need to change one or more properties, the change will take effect immediately in all of the forms that use that model.

The Inheritance Mechanism

This section explains Magic xpa's inheritance mechanism.

When assigning a model to a **new** object, the object properties are inherited from the model.

Selecting or typing a new value in the properties will break the inheritance mechanism for the specific property.

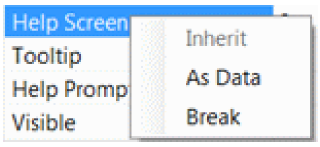
When assigning a model to an **existing** object, only the properties that have not been set individually (modified) are inherited from the model.

When changing an object's model to another model, the associated object inherits the properties of the new model.

When removing a model from an object, the object's properties are set as the object's default values.

For properties in the Form Designer, to control the inheritance mechanism, you right click on the property and select **Inherit**, **As Data** or **Break**.

A modified property (a disinherited property) is displayed in **bold**. A property set **As Data** will appear with the following marking: ✚

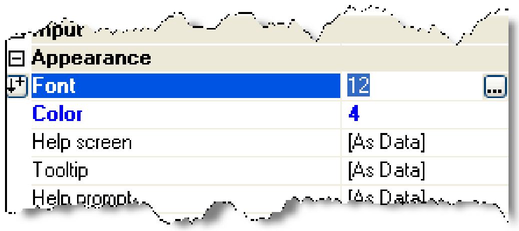


The **As Data** option only appears for some properties and only when there is a value set in the **Data** property. When this option is selected, the property inherits its value from the **Data** property and not from the model.

For properties not in the Form Designer, you can control the inheritance mechanism for each property using the following **toggle** button. This button displays the opposite of the current property's inheritance status. So, if the property is broken, the inheritance icon is displayed.

	Break Inheritance	Use this button to break the inheritance of the property.
	Inherit Property	Use this button to inherit the property from the model.

A modified property (is displayed in blue and **bold**.

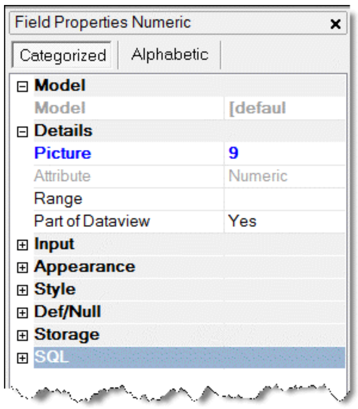


⊟ Appearance	
Font	12
Color	4
Help screen	[As Data]
Tooltip	[As Data]
Help prompt	[As Data]

Field Class Models

You will now learn how to define a **Field** class model. You will define a **Model** for **Code**, which can be used for both the **Customer_Code** and the **Supplier_Code**.

1. From the **Project** menu, select **Models** (Shift+F1).
2. Create a line.
3. In the **Name** column, type: **Code**.
4. From the **Class** column, select **Field**.
5. From the **Attribute** column, select **Numeric**.
6. Open the **Code** model properties.
7. In the **Picture** property, type: **9**.



⊟ **Model**	
Model	[defaul
⊟ **Details**	
Picture	9
Attribute	Numeric
Range	
Part of Dataview	Yes
⊞ **Input**	
⊞ **Appearance**	
⊞ **Style**	
⊞ **Def/Null**	
⊞ **Storage**	
⊞ SQL	

You have just defined your first model in the Model repository.

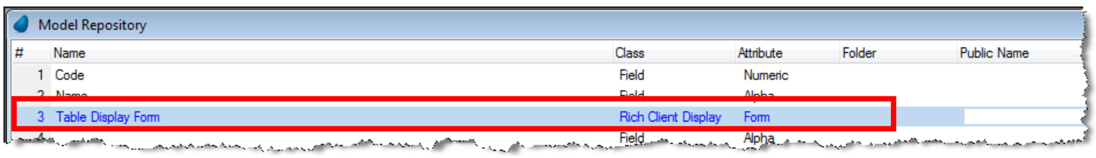
In a similar manner you can define a model for **Name**, which can be used for both the **Customer_Name** and the **Supplier_Name**.

1. Create a line.
2. In the **Name** column, type: **Name**.
3. From the **Class** column, select **Field**.
4. From the **Attribute** column, select **Alpha**.
5. Open the **Code** model properties and in the **Picture** property, type: **20**.

Display Class Model

Previously in this course, you created several programs with tables. For each program, you set the **Wallpaper** property. Now you will set a consistent form appearance for these programs using a **Rich Client Display** class model.

1. Create a line in the Model repository.
2. In the **Name** column, type: **Table Display Form**.
3. From the **Class** column, select **Rich Client Display**.
4. From the **Attribute** column, select **Form**.



Model Repository

#	Name	Class	Attribute	Folder	Public Name
1	Code	Field	Numeric		
2	Name	Field	Alpha		
3	Table Display Form	Rich Client Display	Form		
4		Field	Alpha		

5. Set the **Wallpaper** property to: **%WorkingDir%env\Wallpaper.jpg**.
6. Set the **Wallpaper Style** to **Distorted Scaling**.

When creating the programs, you changed the font and color of each of the static texts. In this section you will define a Rich Client class model for a static control. The most commonly used static control in your project is a text control that is used as a label. You will now create a Label control model and set its font, color, and style. Later, you can assign this model to all of the Text Caption controls in the forms.

7. Create a model.
8. In the **Name** column, type: **Text Caption**.
9. From the **Class** column, select **Rich Client Display**.
10. From the **Attribute** column, select **Label**.
11. Define the following properties:
    - **Font** – Text caption
    - **Color** – Text caption
    - **Style** – No Border

In this section you will define a Rich Client class model for the Edit control. Later, you can assign this model to all of the controls in the forms.

12. Create a model line.
13. In the **Name** column, type: **Edit Control**.
14. From the **Class** column, select **Rich Client Display**.
15. From the **Attribute** column, select **Edit**.
16. Set the **Color** property to the **Edit Control** color.

Assigning Models to Objects

Usually, you create the model prior to the object creation (a column in a data source, or a variable in a task). Then, when you create the object, you only need to set the model that the object should use and that's it; all of the object's properties will be inherited from the model.

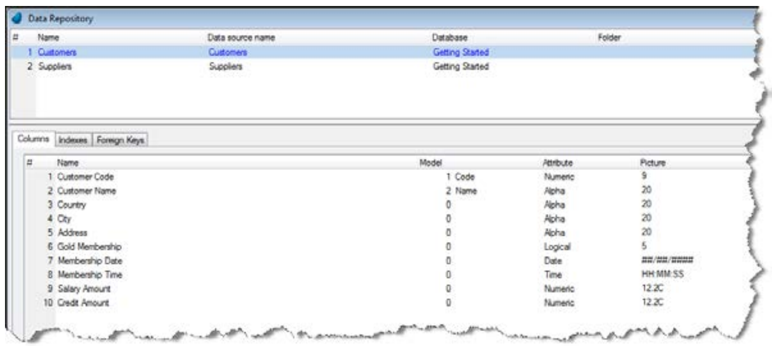
Up until this stage in the course, the data sources were already created and you simply created a definition for it in Magic xpa. Therefore, you need to assign the model to an existing object. Then, you need to change the broken properties so that they inherit the value from the model.

In the following examples you will assign different types of models to objects.

Assigning a Field Model to a Column

In this example, you will assign the **Code** model to the **Customer_Code** column in the **Customers** data source.

1. Open the Data repository and park on the **Customers** data source.
2. Zoom to the Column repository.
3. Park on the **Customer_Code** column.
4. From the **Model** column, zoom to the Model selection list.
   Note: The Model list displays only the available models for the object. In this example, only Field class models are displayed.
5. Select the **Code** model.
6. Repeat steps 3-4 for the **Customer_Name** column and select the **Name** model.



#	Name	Data source name	Database	Folder
1	Customers	Customers	Getting Started	
2	Suppliers	Suppliers	Getting Started	

#	Name	Model	Attribute	Picture
1	Customer Code	1 Code	Numeric	9
2	Customer Name	2 Name	Alpha	20
3	Country	0	Alpha	20
4	City	0	Alpha	20
5	Address	0	Alpha	20
6	Gold Membership	0	Logical	5
7	Membership Date	0	Date	##/##/####
8	Membership Time	0	Time	HH:MM:SS
9	Salary Amount	0	Numeric	12.2C
10	Credit Amount	0	Numeric	12.2C

Inheriting the Model Properties

Assigning a model to an object is only part of the process.

When assigning a model to an object, Magic xpa sets all modified properties as broken. You want the object to inherit its properties from the model; therefore, you need to change the status of the broken properties to the inherit status. This is very useful if you decide you want to change the picture of the field. You would then only make the change in the model and not everywhere that it is used.

You want the object to inherit the **Picture** properties from the model.

7. Park on the **Customer_Code** column and open the Column properties.
8. From the **Picture** property, click the **Inherit** button.
9. Verify that the button displays after you click the button.

Note: Notice that the **Code** model is set in the **Model** property.

10. Repeat the above steps for the **Customer_Name** column.

When you exit the **Customers** data source's Column repository, you will receive a confirmation window asking you if you want to save the changes.

11. Click **Yes**.

Using a Field Model in a Task

You can also use models within tasks.

1. Open the Program repository and open the first program you wrote, **My First Program**.
2. Open the Data View Editor and park on the **Customer_Code** Virtual variable's **Model** column (to the right of the word **Customer_Code**).
3. Zoom to the Model list and select the **Code** model.

Inheriting the model's properties:

4. Park on the **Customer_Code** column and open the Column properties.
5. From the **Picture** property, click the **Inherit** button.
   Verify that the button displays after you click the button.

Note: Notice that the **Code** model is set in the **Model** property.

6. Repeat for **Customer_Name**.

Assigning a Model to a Control

1. Open the **Customers - Screen Mode** program.
2. Open the task form.
3. Open the **Customer_Code** Label properties.

4. Park on the **Model** property and zoom to the Model list by pressing the ... button or by pressing **F5**.
5. Select the **Text Caption** model.
6. From the **Font** property, right-click and select **Inherit**.
   Verify that the value does not appear bolded after you select **Inherit**.
7. From the **Color** property, right-click and select **Inherit**.
8. From the **Style** property, right-click and select **Inherit**.
9. Repeat steps 4-8 for all captions. (Remember that you can mark more than one control at a time.)

Exercises

Now you will practice working with models:

1. Create the following models for the **Customers** and **Suppliers** data sources:

Name	Class	Attribute	Model Properties
Country	Field	Alpha	Picture: 20
City	Field	Alpha	Picture: 20
Address	Field	Alpha	Picture: 20
Gold_Membership	Field	Logical	Picture: 5
Amount	Field	Numeric	Picture: 12.2C
Phone_Number	Field	Alpha	Picture: ###-#######
Years_Since_Start_Working	Field	Numeric	Picture: 2
Bonus	Field	Numeric	Picture:3.2

2. Assign models to all of the **Customers** and **Suppliers** data source columns.
3. Don't forget to inherit the model's properties.
4. Assign a model to all of the Edit controls in the **Customers - Screen mode** task's form.
5. Define a model for an Edit control named **Display Only** that has:
   a. A color based on the **Text** color.
   b. No border.
6. Define a model for a Rich Client table that has:
   a. A color based on the **Text** color.
   b. Row highlighting that is based on the **Row Highlighting** color.
   c. Placement so that the length of the table increases along with the form.
7. Define a model for a table column that has the **Text Caption** font and color.

This is a bit more challenging:

8. Make it so that when the code and name are placed in a Rich Client table, they will always have a **Color** property of **Edit control** and no border.
   Hint: Look at the properties of the model.
9. Implement the models defined in 5 through 7 on the **Customers - Line Mode** program.

Summary

In this lesson you learned about Magic xpa models.

A model is a collection of object properties.

The use of models is optional, but using them will benefit you throughout the development and maintenance of the project.

Some of the advantages are:

- Saving development time
- Ensuring consistency throughout the project
- Ease of maintenance

You learned how to create models, and how to assign models to columns, variables, controls, and forms.

Lesson 9

The Application Engine Concept

The Magic xpa engine is the driving force behind Magic xpa's ability to perform data manipulations that are largely transparent to the developer and to the end user.

The Magic xpa engine automatically manages actions, such as opening files, reading records from database sources, sorting records, and additional actions that are not usually part of other development tools. The Magic xpa engine saves the developer considerable amounts of development time by supplying these built-in operations.

Magic xpa enables you to define execution steps, called logic units, and to define a set of operations within those logic units. You can select which operations are executed and when, but the Magic xpa engine dictates which steps should be taken to accomplish these operations. These steps are mostly pre-defined and carried out by the engine with no input from the developer or the end user. It is important for you to understand these steps, so that you can then develop your tasks with this information in mind.

This lesson covers various topics including:

- The Magic xpa engine
- Event-driven methodology
- Task execution logic units
- Task execution rules

Event-Driven Development Concept

The Magic xpa engine is event-driven. This means that the engine responds to events that occur during the task execution. The developer does not know which part of the code will be executed next. The event-driven methodology consists of three parts: the event, the trigger of the event, and the handler of the event.

Event

An event is a logical definition of an occurrence. The event is a flag that tells Magic xpa that something should be handled. The event can be a built-in Magic xpa event or it can be an event that was defined by the user.

Triggering an Event

An event needs a trigger that will determine when the event occurs. An event can be triggered internally by Magic xpa, by the developer, or according to an end-user action.

Handling an Event

Magic xpa takes care of an event using handlers. Each event should have a handler that handles the event when it is triggered. The handler usually consists of a set of operations. The handler can be internal in Magic xpa or defined by the developer.

You will learn more about events and handlers in the next lessons.

The Task

Magic xpa's basic object is the task. A program contains one or more tasks.

A task is a data-handling procedure with a defined beginning and end, which carries out a set of operations with or without end-user interaction. The task execution process is mainly defined by the task's main data source and the task type. Therefore, the first step in programming is defining the task's Main Source and the task type.

Main Source

Most of the time, the engine executes a task by looping through the Main Source of the task.

Task Type

There are four types of tasks: Online, Batch, Browser and Rich Client. Rich Client tasks can be marked as interactive and non-interactive. During this course you have only worked with interactive Rich Client tasks.

Online, **Browser** and **interactive Rich Client** tasks enable end-user interaction through the task execution. When a Main Source is defined, the end user browses the Main Source by navigating through the records. Only the records, which are browsed by the end user, are processed by the engine.

Batch and **non-interactive Rich Client** tasks are used to perform automatic procedures, such as producing reports and scanning a data source for calculations. In these tasks, when a Main Source is defined, Magic xpa loops automatically through the task's Main Source, starting from the first record until the last record within a given range.

In all types of tasks, you can define a range. Magic xpa loops through the records within the range.

Task Execution Stages

Once the Main Source and the task type are set, the next step is to customize the task to accomplish a specific implementation. This is done by selecting the operations to be performed, conditioning the operations' execution, and defining in which stages of the task execution the operations will be executed.

A standard task execution process follows several stages. These stages change according to the task type and according to the set logic units.

For example, in a simple Online or interactive Rich Client task, the following stages occur:

1. The database data sources that are defined in the task data view are opened (including the Main Source and linked data sources). You will learn more about this later on.
2. Magic xpa starts the task, performing actions such as initializing Virtual variables with values.
3. The first record of the Main Source is fetched for processing.
4. Manipulations are performed on the record, according to the set operations. These may include displaying the record on the screen, accepting the end-user selections and updates, and writing the record back to the database.
5. The engine checks whether another record should be fetched for manipulation or whether to end the task execution (the end-user requests to leave the task or the **End Task Condition** evaluated to **True**).
   If the next record should be fetched, Magic xpa returns to stage 3.
6. Magic xpa terminates the task, performing actions such as closing database data sources.

The Task Execution Logic Units

Most Magic xpa logic units are pre-defined for specific events that occur during the lifecycle of the task.

Within the logic unit, you define a set of operations to be performed. Consequently, these operations will be performed when the event is triggered.

Magic xpa logic units are designed to handle different task levels, as shown below.

- The Task level, which occurs when the task starts or when it ends, is handled by the task's logic units.
- The Record level, which occurs when Main Source records are manipulated, is handled by the record's logic units.
- The Control level, which occurs when a specific control is accessed by the end user, is handled by the control's logic units.
- The Variable level, which occurs when a variable value is changed, is handled by the variable's logic unit.
- The Event level, which can occur at any point where the event is triggered, is handled by the Event logic unit.

Operations within the logic units described above can be defined by the developer. Magic xpa has additional code that is internally executed according to Magic xpa's internal rules and cannot be seen or accessed by the developer. On the next pages, you will learn about the various logic units.

Task Prefix and Task Suffix

The Task logic unit is one of Magic xpa's basic logic units. In the Task logic unit you can specify operations that should be executed at the beginning and at the end of the task. It is divided into two logic units:

Task Prefix

The Task Prefix logic unit is executed **once** when the task begins. Its trigger is the task execution. Within this logic unit, you define operations that will be executed during the initialization stage. (Data source information is not available in this stage.)

Task Suffix

The Task Suffix logic unit is executed **once** when the task terminates. Its trigger is the task termination. Within this logic unit, you define operations that should be done once as a termination stage. (Data source information should not be manipulated in this stage.)

Record Prefix and Record Suffix

The Record logic unit is one of Magic xpa's basic logic units that occur during the task's lifecycle. In the Record logic unit you can specify operations that should be executed in the Record level, for every record, when it is fetched and when its manipulation ends. It is divided into two logic units:

Record Prefix

The Record Prefix logic unit is executed for every record immediately after the record is fetched and before the end user sees the record's content. Its trigger is when the record is fetched. In the Record Prefix logic unit, you define operations that should be done once per record, as the record's initialization stage.

Record Suffix

The Record Suffix logic unit is executed as follows:

- **Batch** and **non-interactive Rich Client** tasks – will be executed for each record
- **Online**, **interactive Rich Client** or **Browser** tasks – will be executed only if the user modified the current record. You can force this mode by using the **Force Record Suffix** property.

In the Record Suffix logic unit, you define operations that will be executed once per record during the record's termination stage and before the record is saved to the database.

The Record Suffix logic unit may be executed 0, 1, or 2 times per record, depending on the end user's interaction with the record. In **Delete** mode, the Record Suffix is executed twice.

The Record logic unit is cyclical and may be executed several times during the task execution.

Control Prefix, Control Verification, and Control Suffix

The Control logic unit is one of Magic xpa's basic logic units. The Control logic unit is available for Online, Rich Client, and Browser tasks.

In the Control logic unit you can specify operations that should be executed whenever a certain control on the form is accessed.

Control Prefix

The Control Prefix logic unit is executed before the insertion point is moved to the control. Its trigger is when the end user takes an action to park on the control.
In the Control Prefix logic unit, you define operations that are done before the end user can manipulate the control's content.

Control Verification

The Control Verification logic unit is executed whenever the insertion point is taken away from the control, before the Control Suffix logic unit is executed.

This logic unit is also executed when the insertion point passes through the control without parking on it. (This is a special mode, called **Fast** mode, in Magic xpa.)
In the Control Verification logic unit, you define operations that validate the control content, regardless of whether or not the end user edits the control.

Control Suffix

The Control Suffix logic unit is executed whenever the insertion point is taken away from the control. Its trigger is exiting the control.
In the Control Suffix logic unit, you define operations that validate the control content, or define actions that result from editing the control.

The Control logic unit can be executed several times depending on the number of end-user interactions.

Variable Change

The Variable logic unit is one of Magic xpa's basic logic units. In the Variable logic unit you can specify operations that will be executed as a result of a change to the variable value.

The **Variable Change** logic unit is executed whenever the variable value is changed by the end user (editing the control) or internally (such as using the Update operation). Its trigger is the variable change.

The Variable Change logic unit can be executed several times for each change to the variable value.

Short Summary

The following table describes Magic xpa's basic logic units and their triggers.

Logic Unit Name	Logic Unit Trigger	Number of Executions
Task Prefix	A task execution starts	Once
Task Suffix	A task execution ends	Once
Record Prefix	A record is fetched	1 per record
Record Suffix	A record manipulation ends	0 - 2 per record
Control Prefix	An end user parks on a control	-
Control Verification	An end user leaves a control, before Control Suffix	-
Control Suffix	An end user leaves a control	-
Variable Change	A variable value is changed	-

Execution Rules

This section will provide you with the details about the processes that are executed during the task's lifecycle.

Task Initialization

The Data View Preparation

- All of the data sources defined in the Data View Editor are opened.
- Magic xpa examines the task data view operations, and the task's logical record is created based on:
    - Column definitions from the Main Source and linked data sources.
    - Virtual variables and parameters.
- Parameter values are received from a calling task.
- Virtual variables are initialized with values that match the variable attribute (numeric variables are initialized with zero, alpha with blanks, etc.).
- The task range is defined according to the range criteria in the **Range Window**.
- The engine sorts the data view according to the specified Sort Indicators (if specified in the **Sort Indicator** window).

Additional Preparations

The next steps include operations that initialize the task.

- I/O devices are opened (if I/O devices are specified in the I/O Device repository).
- The operations in the Task Prefix logic unit are executed.

At this stage, no data from the Main Source or linked sources is available.

Record Processing

After the Task Prefix is executed, the data view records are processed as follows:

Record Prefix

- The first record in the data view is located and fetched.
  In Online and interactive Rich Client tasks, if there are no records in the data view, the task can either terminate or go into Create mode. It can also display a screen without input.
  This behavior is governed by the task properties called **Allow Empty Dataview** and **Allow Create**.
- The Init expression is evaluated for all of the Virtual variables. The Init expressions for columns are evaluated only if the task is in Create mode.
- All of the links to data sources are executed for all records displayed on the screen.
- If the **Evaluate condition** option in the Task Properties is set to **Before entering record**, Magic xpa evaluates the **End Task condition**.
- The Record Loop begins here. Therefore, from here on you can manipulate data source columns.
- The Record Prefix logic unit operations are executed.

Processing Controls

For each record, in interactive tasks, the accessed controls' logic units are executed.

- Control Prefix – For each control which is accessed by the end user, the Control Prefix logic unit operations are executed.
- Control Verification – The Control Verification operations are executed whenever the end user moves from the control that the Control Verification logic unit is set for, or the control is skipped over (Fast mode). A control is skipped over in Fast mode when the end user skips over this control to get to a different control.
- Control Suffix – The Control Suffix is performed whenever the insertion point is taken away from the control. This occurs when the end user moves to a different control, exits the record, or exits the task.

Processing Variables

In Online and Rich Client tasks, the Variable Change logic units are executed whenever a variable value is changed.

Record Suffix

In Online, Rich Client and Browser tasks, the **Record Suffix** logic unit is executed for records that were modified.
In Batch tasks, the **Record Suffix** is always executed.

In Online, Rich Client and Browser tasks, the **Record Suffix** logic unit is executed when the end-user interaction with a record is terminated. This happens when:

- The end user moves to another record.
- The end user changes the Task mode (Modify, Create, Query).
- The end user deletes the record.
- The task execution is terminated, such as a result of end-user interaction or if an End Task condition evaluates to True.

When Magic xpa terminates the task, the Record Loop must be terminated first.

Task Termination

There are several ways to terminate the task execution, such as the end user leaves the task, the **End task condition** evaluates to True, or the **Exit** event is raised.

Before the task is terminated, the following happens:

- The Task Suffix logic unit operations are executed.
- I/O devices are closed.
- The task's data sources are closed.

The Task Suffix logic unit is executed after the termination of the Record Loop. Therefore, there are no data values that are reliable during this stage of execution.

Brief Overview

The following table summarizes the Magic xpa logic unit triggers, execution rules, and the order in which the logic units are executed.

Logic Unit	Logic Unit Trigger	Next Logic Unit	The Next Logic Unit Executes If:	Else, Execute
Task Prefix	A task is executed	Record Prefix	There is at least one record in the data view	Task Suffix
Record Prefix	A record is fetched	Control Prefix	This is an Online or Rich Client task	Record Suffix
Control Prefix	The cursor parks on a control	Control Verification	No restrictions	
Control Verification	Control editing is terminated or the cursor skips over the control	Control Suffix	No restrictions	
Control Suffix	Control editing is terminated	Variable Change	The variable value is changed	Record Suffix
Variable Change	A variable value is changed	Record Suffix	The record is about to be saved in an Online or Rich Client task	Task Suffix
Record Suffix	A record is about to be saved	Task Suffix	No restrictions	
Task Suffix	A task is about to be terminated			

The above table demonstrates only one execution cycle for each logic unit. If the record or the control cycle is not terminated, that cycle will be repeated (from its Prefix logic unit to its Suffix logic unit).

A Control logic unit can only be executed inside a Record logic unit cycle, and a Record logic unit can only be executed inside a Task logic unit.
A parent logic unit cannot be terminated unless its sub-logic unit is terminated.

Summary

In this lesson you learned about the Magic xpa engine concept.

You were introduced to the event-driven methodology, which consists of three basic elements: the event, trigger, and handler.

You learned about the task execution process, including the task type and the task's Main Source.

The task's main levels include: the Task level, Record level, Control level, and Variable level. Each level is handled by different logic units. You learned about the task execution logic units, as well as the logic units' triggers and roles.

As you can see, Magic xpa has a lot of internal processes that are performed according to various scenarios. For example, when you select main data source columns in a Rich Client task, Magic xpa internally opens the data source, selects the columns from the data source, handles the displayed data, and saves the data to the data source after the end user chooses to leave the record. All of these processes were done by Magic xpa automatically.

Now that you know about most of the Magic xpa engine's behavior, you will be able to create more efficient programs that take advantage of the task execution engine's behavior rules.

Lesson 10

Events and Handlers

Event-driven programming is the writing of code for a logical unit that handles a certain scenario and will be executed on demand.

Procedural programming is the writing of code so that the operations are executed one after another, according to a predefined scenario.

To implement event-driven programming, it is important to know about triggers, events, and handlers.

You need to know how to set events in Magic xpa and how to trigger them. You must also know how to create handlers that will handle the events.

This lesson will introduce you to the Events and Handlers concept in Magic xpa. This lesson covers various topics including:

- Magic xpa triggers
- Magic xpa events
- Defining an event
- Invoking an event
- Event-driven logic
- Handling events
- Checking events
- Propagate mechanism
- Overwriting internal event handling

Events and Handlers Concept

As you learned in the previous lesson, event-driven programming consists of three elements:

- An **event** is a logical definition of an occurrence.
- A **trigger** is an action that triggers the event. Triggers can be external (by the end user) or internal (Magic xpa internal code or programmer code).
- A **handler** is a logic unit of code that is activated as a response to the event triggering.

An event can have more than one trigger and more than one handler. Each trigger can raise the event in a different scenario. One or more handlers can handle the event for each scenario.

Magic xpa events can be project-related events, which are defined in the Main program and are applied to the whole project. These events can be triggered during the execution of any of the project's programs. The events can also be task-related events, which are defined in a specific task and are applied to the task in which they are defined or to the task and its subtasks.

Types of Events

There are different types of events in Magic xpa. The first two events will be covered in the scope of this course.

- **Internal events** – These are pre-defined internal Magic xpa events. These events are generally invoked as a result of a user's action. For example, when you tapped a field, you raised a **Control Prefix** event.
- **User events** – These are events that are created by you, the developer, for use within the application.
- **Timer events** – These events are raised at the end of a defined time interval.
- **Expression events** – These events are invoked when a certain expression evaluates to True.
- **Error events** – These events are invoked when a database-related error occurs.
- **.NET events** – This is only available in Rich Client tasks on desktop machines.

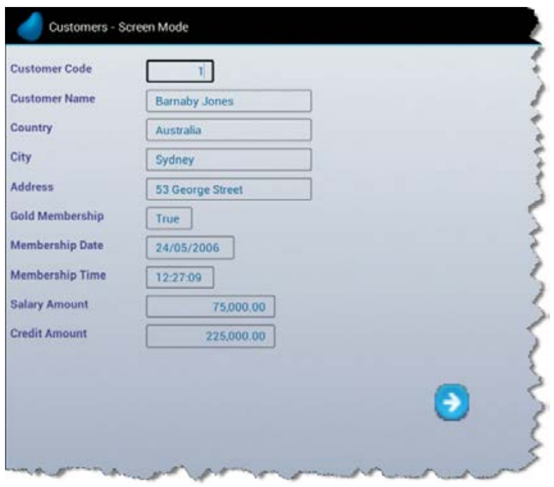
Raising Internal Events

In a previous lesson you created a program that displayed a single record in screen mode and when you displayed the same data source in table mode, you discovered that there were more records. You will now create a button that will enable you to move between the records.

1. Zoom into the **Customers - Screen Mode** program.
2. Zoom into the Form Designer.
3. Place a Button control at the bottom of the form.
4. Zoom into the control properties.
5. Park on the **Format** property and remove the text, which is initially set to **Button**.
6. Park on the **Button style** property and select **Image button** from the combo box.
7. Park on the **Image List file name** property and type **%WorkingDir%\images\Next.png**.
8. From he **Event Type** property, select **Internal**.
9. From the **Event** property, select **Next Row**.
10. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
11. From the toolbar, click the **Fit Control Size** icon.

You have just defined a button that when tapped, the next record will be displayed on the form.

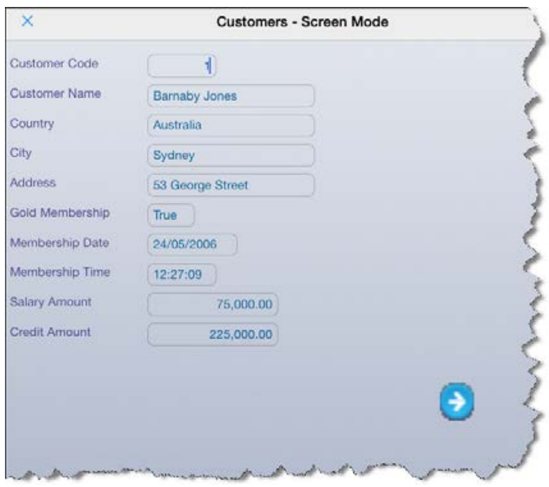
Android



Customers - Screen Mode

Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

iOS



Customers - Screen Mode

Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

How Does It Work?

You defined a button that when tapped, the next record was displayed. You did this by raising the Magic xpa internal event called **Next Row**. You learned previously that an event has three elements:

- The **event**, which in this case is the **Next Row** event.
- The **trigger** that triggers the event, which in this case was invoked by tapping the push button.
- The **handler**, which is code that is activated as a response to the event triggering. In this case it was the Magic xpa internal mechanism that handled this event for you.

Using the same method you can define a button that will move to the previous record:

1. Place a Button control at the bottom of the form, to the left of the other button.
2. Zoom into the control properties.
3. Park on the **Format** property and remove the text, which is initially set to **Button**.
4. Park on the **Button style** property and select **Image Button** from the combo box.
5. Park on the **Image List file name** property and type **%WorkingDir%\images\Previous.png**.
6. From the **Event Type** property, select **Internal**.
7. From the **Event** property, select **Previous Row**.
8. Set the **Color** to **Text Caption**.
9. From the toolbar, click the **Fit Control Size** icon.

Button Control

While working in the Studio you used push buttons often without knowing what actions they performed when you clicked on them. You can use your own push buttons on your form to perform actions that you define. A Button control is used when you want to trigger an action when the end user clicks it or in the case of smartphone applications, taps it. As an example, you can have a push button that when you tap it, the program ends.

There are different types of buttons, depending on the **Button Style**:

- **Image Button** – You used this type in this lesson. An image button is an image file containing four or six images that correspond to the four or six different states of a push button. You can read more about this in the *Magic xpa Help*. In this course you are using six-state image buttons that were designed for this course, such as the Next and Previous images.
- **Hyperlink** – This displays a hyperlink in the same way as a Browser would.

- **Push Button** – The actual look and feel of this button depends on the device that you are using. You need to provide the text that is displayed on the button and the operating system is then responsible for the look and feel.

User-Defined Events

In this section you will define and use User events. As User events are events defined for the application, it is good practice to provide them with a meaningful name.

You will now create two User events:

- **Set Gold Membership** – This event will not have a defined trigger. At a later stage you will use the **Raise Event** operation to invoke the event.
- **Set Date and Time** – This event will be triggered by pressing a button. On a desktop machine, you could raise this event using a keyboard combination, such as Ctrl+T, but with smartphones and tablets you do not have that functionality.

1. In the Program repository, zoom to the **Customers - Screen Mode** program.
2. From the **Task** menu, select **User Events** (Ctrl+U).
3. Create a line and in the **Description** column, type: **Set Gold Membership**.
4. In the **Trigger type** column, select **None**.
5. Create another line and in the **Description** column, type: **Set Data and Time**.
6. In the **Trigger type** column, select **None**.
7. In the **Force Exit** column, select **Control**. This instructs the task to exit the current control before executing a corresponding Event logic unit.

Invoking Events

In the Event repository, Magic xpa enables you to define the trigger type and value that will raise the event.

In some cases, you want to raise the event according to the application logic, regardless of whether the trigger is specified in the Event repository.

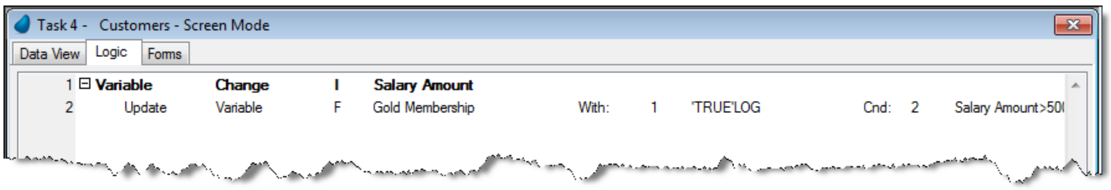
In those cases, you can use the Raise Event operation to trigger the event.

Raise Event Operation

In this example, you want the **Gold_Membership** variable to have a **True** value whenever the **Salary_Amount** is changed to a value greater than **50,000**.

The basic way to perform this is to have an **Update** operation that updates the **Gold_Membership** variable with the **True** value.

The example image below depicts what was described above. (Please do not create this operation.)



```
1 ⊟ Variable      Change     I   Salary Amount
2       Update    Variable   F   Gold Membership          With:   1   'TRUE'LOG          Cnd:   2   Salary Amount>50
```

Changing the customer status to **Gold** may involve more than just updating the **Gold_Membership** variable to **True**. (This will not be demonstrated in the current example.)

In addition, the customer status may be changed for reasons other than the customer's salary. (This will also not be demonstrated in the current example.)

When using the event-driven methodology, the upgrading of the customer status is done in one location (the handler), and the reasons (triggers) for upgrading the customer status can be done in many locations.

You will now create the logic unit that will raise the **Set Gold Membership** event when the **Salary_Amount** value is changed.

8. In the **Customers - Screen Mode** program, open the Logic Editor and create a Header line (**Ctrl+H**).
9. Set a **Variable Change** logic unit for the **Salary_Amount** variable. Answer "No" to the question about creating parameters. The reason for doing this is that you will not need the parameters within the scope of the handler.
10. Create a line and select the **Raise Event** operation from the drop-down list.
11. In the **Event** dialog box, from the **Event Type** drop-down list, select **User**.
12. From the **Event** field, select the **Set Gold Membership** event and click **OK**.
13. Set the **Wait** property to **Yes**.
14. In the **Cnd** field (or in the **Condition** property), create an expression for when the **Salary_Amount** is greater than **50000**, for example: **I>50000**.
15. Click **OK**.

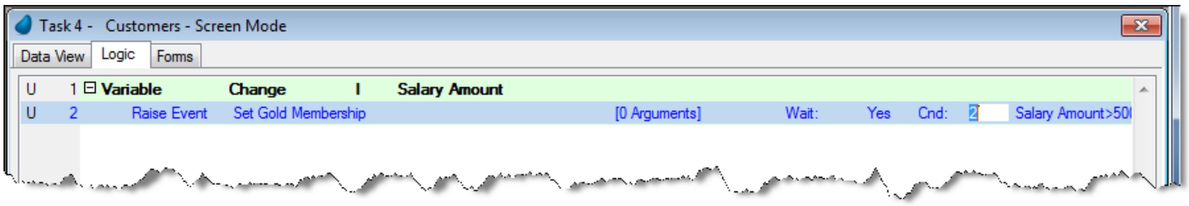
The **Gold_Membership** event will be invoked when the **Salary_Amount** value is changed and its value is greater than **50,000**.

You have just used the Raise Event operation to invoke an event according to the task logic.

The Raise Event operation's **Wait** property indicates whether the event should be invoked immediately (Yes) or placed at the end of the event queue (No).

The **Arguments** property defines the arguments that will be sent to the event handler once the event is invoked. The property displays the number of arguments passed from the raised event.

Note that by raising the event, Magic xpa does not handle the event at this stage. (If the event is raised, nothing happens since the event has not been handled.)



```
Task 4 -   Customers - Screen Mode
Data View | Logic | Forms
U   1 ⊟ Variable      Change             I   Salary Amount
U   2       Raise Event   Set Gold Membership                  [0 Arguments]     Wait:     Yes     Cnd:   2     Salary Amount>50
```

Invoking a User Event Using a Button Control

Like with the Raise Event operation, you can invoke a User event using a Push Button control on the task form.

Unlike the Raise Event operation, the Push Button control cannot force the task execution to wait until the event is done.

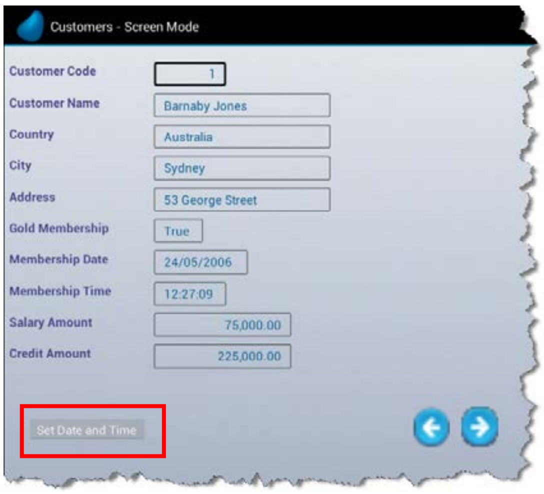
The Raise Event operation and the Push Button control can be set to invoke the following event types:

- User – Any of the events that are defined in the Event repository.
- Internal – Magic xpa internal events.
- System – Magic xpa internal events that are mapped to a specific keystroke combination. This is unavailable in smartphones and tablets.

In this lesson, you will invoke a user-defined event from a push button.

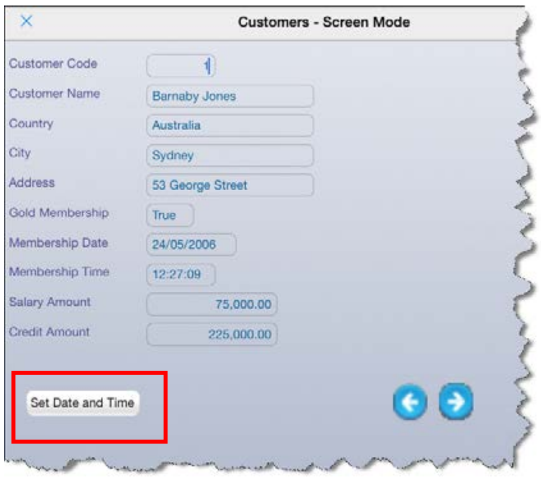
1. Zoom into the **Customers - Screen Mode** program.
2. Open the Form Designer.
3. Add a Button control. Zoom into the control properties.
4. Park on the **Format** property and type in: **Set Date and Time**.
5. From the **Event type** property, select **User**.
6. From the **Event** property, select **Set Date and Time**.
7. From the toolbar, click the **Fit Control Size** icon.

Android


Customers - Screen Mode

Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

iOS


Customers - Screen Mode

Customer Code	1
Customer Name	Barnaby Jones
Country	Australia
City	Sydney
Address	53 George Street
Gold Membership	True
Membership Date	24/05/2006
Membership Time	12:27:09
Salary Amount	75,000.00
Credit Amount	225,000.00

Handlers

In the previous section, when you tapped the **Set Date and Time** button, nothing visible happened. In essence what happened was that the button raised the **Set Date and Time** event but there was no mechanism in place to deal with the event. This is known as a handler.

You use the Event logic unit to handle user-defined events. A Magic xpa handler is a set of operations to be performed when a specified event is invoked. Magic xpa has the following handlers:

- **Handlers for internal events** – These are Magic xpa engine handlers that handle internal events. For example, to handle the **Next Row** internal event, Magic xpa will leave the current record and park on the next one.
- **Handlers for built-in events** – These handlers perform the engine execution rules for the different task levels. In previous lessons, you created a handler for one of the built-in events, the **Variable Change** logic unit.
- **Handlers for user-defined events** – You create these handlers using the Event logic unit.

Event Logic Unit

The Event logic unit enables you to handle events. Using the Event logic unit, you can assign a handler to an event. The Event logic unit is executed only when the assigned event is invoked during runtime. The Event Logic unit properties include:

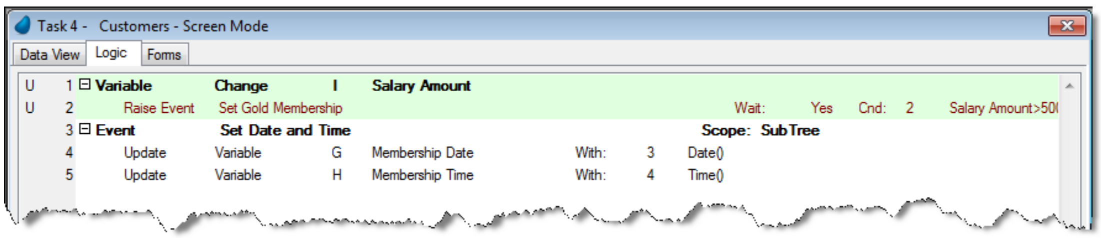
Parameter Name	Description
Event Name	Displays the handled event name. From this parameter, you can: • Zoom to the Event dialog box to select the handled event. • Select the Event type from the Event Type drop-down list. • Zoom from the Event field to select an event from the Event list.
on:	Enables you to assign an event to a control (one of the current task's controls). This column displays the assigned control name. You can type a control name or zoom to select a control name from the Control list.
Scope	Enables you to define the event scope. The options are: • Task – The handler will be executed if the event is invoked from that task only. • SubTree – The handler will be executed if the event is invoked from that task or its subtasks. • Global – The handler is executed if the event is raised from the current task, its subtasks or any other projects (host or components). This option appears in the Main program.
Cnd	You can condition the execution of the handler by setting an expression.

Handling the Set Date and Time Event

Previously you created the **Set Date and Time** event, which was triggered when the user tapped a push button. When this event occurs, the **Membership_Date** and the **Membership_Time** variables need to be updated with the **current date** and **time** values. Now, you will create the handler for the event.

1. In the Logic Editor, create a **Header line** and select **Event**. The **Event** dialog box appears.
2. In the **Event type** combo box, select **User**.
3. Zoom into the **Event** list and select **Set Date and Time**.
4. Create a Details line and update **Membership Date** with **Date()**.

5. Update **Membership Time** with **Time()**.



```
U   1 ⊟ Variable      Change               I   Salary Amount
U   2      Raise Event   Set Gold Membership                                    Wait:   Yes   Cnd:  2   Salary Amount>50(
    3 ⊟ Event          Set Date and Time                                   Scope:  SubTree
    4      Update        Variable           G   Membership Date     With:   3   Date()
    5      Update        Variable           H   Membership Time     With:   4   Time()
```

You have just created the **Set Date and Time** event handler. If you execute the program and then click the **Set Date and Time** button, you will see that the date and time are updated accordingly. Remember that this is not updated to the database until you move to a different record.

Event Checking

Magic xpa inspects the task events at different times, depending on the task type.

Online, Interactive Rich Client and Browser Tasks

In general, in this course you will only use the Rich Client task. Events are checked when:

- The end user taps something in a Rich Client task or for other tasks when they press a key.
- The task is idle. This occurs when the task is waiting for end-user input and the **Keyboard Idle Seconds** environment setting timeout occurs.

Batch and Non-Interactive Rich Client Tasks

You will learn more about this type of task in a later lesson:

- The time interval for a Batch task depends on when the **Batch Event Interval** environment setting timeout occurs.
- The number of processed records set in the **Record Event Interval** environment setting is reached.

Using the Raise Event Operation

This explanation is valid for any task type. The Magic xpa engine checks the **Raise Event** operation's execution condition. If you use the Raise Event operation with the **Wait** property set to **Yes**, Magic xpa will find and execute the related event handler immediately. This is known as synchronous. If the **Wait** property is set to **No**, the event will be added to an event queue and will be inspected according to the task type as described above. This is known as asynchronous.

Having More than One Handler for the Same Event

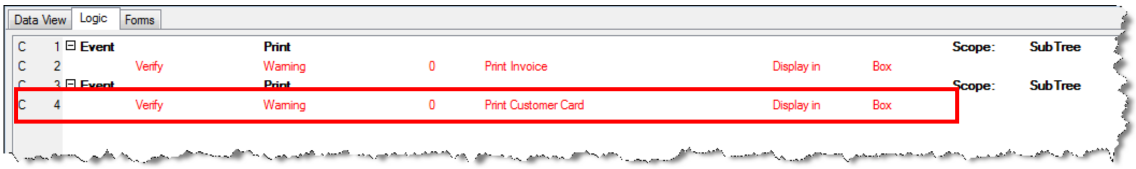
In some cases you have more than one handler for the same event. For example, take a scenario where you have two handlers for a user-defined event named **Print**, one that prints the customer's details meaning the customer card and one that prints an invoice for the order.

Magic xpa's default behavior is that each raised event is handled by the first handler (according to Magic xpa's rules) and after that the event is cleared. So, no other handler (if one exists) will handle this event. Therefore, according to Magic xpa's rules, only one handler will handle the Print event.

Which Handler Will Handle the Event?

According to Magic xpa's rules the lowest handler in the **execution tree** will handle the event.

In the example above, this means that if you create the **Customer Card** printing handler after the **Invoice** printing handler, meaning physically lower in the Logic Editor, Magic xpa will handle the **Customer card** print handler as shown in the image below.



```
Data View | Logic | Forms
C   1 ⊟ Event              Print                                                          Scope:   SubTree
C   2          Verify      Warning     0    Print Invoice          Display in    Box
C   3 ⊟ Event              Print                                                          Scope:   SubTree
C   4          Verify      Warning     0    Print Customer Card    Display in    Box
```

The Propagate Property

You can overcome the behavior described above using the **Propagate** property. The **Propagate** property has two options:

- **Yes** – Magic xpa searches for a higher level handler to handle the event. (If no handler is found, nothing will happen.)
- **No** – Magic xpa does not search for a higher level handler to handle the event.

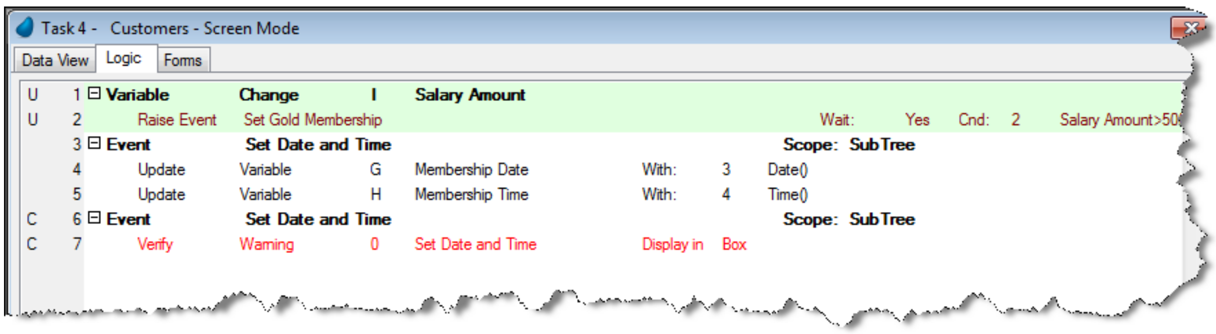
Setting this property to **Yes** is useful when you have a handler for the event in a parent task as well as in the current task. You will learn about this scenario later in this lesson.

Example

To understand the event checking mechanism, you will now add another Event handler for the **Set Date and Time** event.

1. In the **Customer - Screen Mode** program, create another handler for the **Set Date and Time** event.
2. In the details logic, set a **Verify Warning** operation with the text: "**Set Date and Time**".

You have just created an additional handler for the **Set Date and Time** event handler.


Task 4 - Customers - Screen Mode

Data View | Logic | Forms

U	1	⊟ Variable		Change	I	Salary Amount						
U	2		Raise Event	Set Gold Membership					Wait:	Yes	Cnd: 2	Salary Amount>50
	3	⊟ Event		Set Date and Time					Scope:	SubTree		
	4		Update	Variable	G	Membership Date	With:	3	Date()			
	5		Update	Variable	H	Membership Time	With:	4	Time()			
C	6	⊟ Event		Set Date and Time					Scope:	SubTree		
C	7		Verify	Warning	0	Set Date and Time	Display in	Box				

3. Execute the program on your mobile device and tap the **Set Date and Time** button.

You will receive the Warning message box with the text, "Set Date and Time". When you tap the OK button, focus returns to the form but the date and time variables are not updated.

4. Zoom into the **Customer - Screen Mode** program
5. Park on the new **Set Date and Time** handler and in the properties set the **Propagate** property to **Yes**.
6. Execute the program again.

You will notice that the date and time were changed to the current values.

Explaining the results

For each event that is invoked, Magic xpa searches for a handler that is defined for it. Whenever a handler for this event is found, Magic xpa executes the handler.

Upon completion of the handler execution, Magic xpa searches for another handler for the event, only if the **Propagate** property of the last executed event handler is set to **Yes**. (If there are several handlers for the same event in the same task, the bottom-most handler is executed first.) If the **Propagate** property is set to **No**, the search process is terminated.

In the previous example, you defined two handlers for the same event, **Set Date and Time**. When the event was invoked, the first event handler was executed. You were then notified that the date and time were set. Since the **Propagate** property of this handler was set to **Yes**, the next matching handler for this event was executed. Then, the **Membership_Date** and **Membership_Time** variables were updated with the current date and time values.

Handling Internal Events

Earlier in this lesson you added buttons to the form that raised internal Magic xpa events, such as Next Row and Previous Row. When these events were raised you allowed the Magic xpa engine to handle these events. As with other events you can add your own code to handle these events, basically overriding the Magic xpa default behavior.

As an example you will add an Edit button to the customer list so that you can edit the customer's details.

1. Zoom into the **Customers - Line Mode** program and zoom into the Form Designer.
2. Place a Button control at the top of the form.
3. Zoom into the control properties.
4. Park on the **Format** property and remove the text, which is initially set to **Button**.
5. Park on the **Button style** property and select **Image Button** from the combo box.
6. Park on the **Image List file name** property and type **%WorkingDir%\images\Edit.png**.
7. From the **Event type** property, select **Internal**.
8. From the **Event** property, select **Modify Records**.
9. Set the **Color** to **Text Caption**. This will make the button transparent for images which have a transparent color.
10. From the toolbar, click the **Fit Control Size** icon.

Now execute the application. Park on any line and tap the Edit button.

You are now able to edit the information displayed in the table.

?

When developing mobile applications, only part of the information is displayed in the list. Look at the contacts application on your smartphone and edit a contact.

How would you perform something similar in Magic xpa? Where is all of the information for a single customer displayed in a single form?

As you remember, the **Customers - Screen Mode** program displays all of the information for a customer. What you need to do is that when you press the Edit button, Magic xpa will display the **Customers - Screen** Mode program.

11. Zoom into the **Customers - Line Mode** program and zoom into the Logic Editor.
12. Add a Header line and select the **Modify Records** internal event.
13. Create a details line for the **Call Program** operation. This operation enables you to call a program and pass parameters to the new program.
14. Select the **Customers - Screen Mode** program from the list.

If you now execute the application and park on the first customer and tap the Edit button, Magic xpa will display the **Customers - Screen Mode** program and display the first customer. However, if you park on a different customer and tap the Edit button, you want that customer's details to be displayed in the called program. You can implement this by using a parameter.

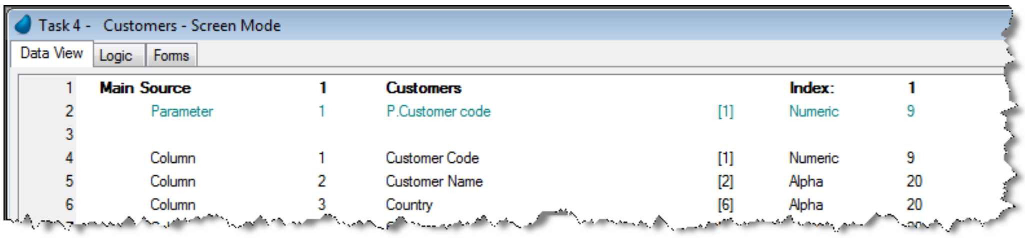
Parameters

A parameter is a local variable that holds the received values from a calling program. It is a channel between the called program and the calling program to pass information from one to the other. The parameter can be used like any other variable. However, you should remember that an update of a parameter variable will result in the parent program's variable also being updated.

1. Zoom into the **Customers - Screen Mode** program and zoom into the Data View Editor.

In the **Customers - Screen Mode** program you will set a parameter for the **Customer Code**. This will allow you to only display a specific customer according to its code.

2. Park on the first line, the **Main Source** definition, and create a line.
3. From the drop-drown list, select **Parameter**.
4. You are now parked in a field displaying **??**. Type in **P.Customer code**. You do not need to add the **P.** before the name you provided. However, when looking at a selection list of variables you will see that it is good practise to make a distinction between different types of variables.
5. To the right of that, zoom and select the **Code** model.



1	**Main Source**	**1**	**Customers**		**Index:**	**1**
2	Parameter	1	P.Customer code	[1]	Numeric	9
3						
4	Column	1	Customer Code	[1]	Numeric	9
5	Column	2	Customer Name	[2]	Alpha	20
6	Column	3	Country	[6]	Alpha	20

Parameters are passed according to the order they appear in the data view. Magic xpa does not make a distinction as to where the parameters are placed within the data view. It is possible to disperse parameters throughout the data view. However, for both readability and maintenance, it is better to group these variables together. Parameters are normally defined at the beginning of the data view.

Range Criteria

When you pass the customer code to the **Customers - Screen Mode** program, you expect the program to only display the customer that you are parked on.

Range criteria enable you to display only those records that fall between two values, from-to. You will now set the **P.Customer Code** parameter as the **Range from** criteria and the **Range to** criteria.

6. Park on the **Customer_Code** column.
7. Zoom into the **Range from** property and zoom into the Expression Editor.
8. Create an expression for the **P.Customer Code** *parameter*. As you can see, the parameter is not only visible due to a different color but also due to the name.
9. Zoom into the **Range to** property and zoom into the Expression Editor. Select the same expression as the **Range from** property.

The next stage is to pass the relevant customer code to this program. This is performed from the calling program.

10. Zoom into the **Customers - Line Mode** program and zoom into the **Modify Records** logic unit.
11. Park on the **Call Program** operation.
12. From the **Arguments** field, zoom to the Arguments repository.
13. Create a line, zoom and select the **Customer_Code** variable.

Now execute the application. Park on any line and tap the Edit button.

The **Customers - Screen Mode** program will be displayed showing the customer you wanted to edit. You can now edit the data. As an example, edit the date and time details.

When you close the **Customers - Screen Mode** program, you return to the **Customers - Line Mode** program with the focus still on the same line that you originally were parked on. The data in the table was updated.

If you edited data that was displayed in the **Customers - Line Mode** table, such as the customer name, you need to inform Magic xpa to refresh the view and display the updated data.

Magic xpa has an internal event appropriately called **View Refresh** and is responsible for refreshing the view. In the lesson scenario, there are two ways of refreshing the view:

- In the called task, meaning **Customers - Screen Mode**. If one of the controls on the form was modified, the task will pass through the **Record Suffix** logic unit. You can raise the **View Refresh** event from the **Record Suffix** logic unit.
- In the calling task, meaning **Customers - Line Mode**. You can raise the **View Refresh** event from the **Modify Records** handler.

For the purpose of this course, you will raise it from the calling task:

14. Zoom into the **Customers - Line Mode** program and zoom into the **Modify Records** logic unit.
15. Add a detail line after the **Call Program** operation.
16. Select the **Raise Event** operation from the drop-down list.
17. In the **Event** dialog box, from the **Event Type** drop-down list, select **Internal**.
18. From the **Event** field, select the **View Refresh** event and click **OK**.

Now execute the application. Park on any line and tap the Edit button.

You can now edit the data. Try editing the name and then returning to the table and see the result. The **View Refresh** event is very useful. You can read more about it in the Magic xpa Help.

You modified the **Customers - Screen Mode** program so that it displayed the record corresponding to the parameter that was passed. What happens if you now execute the **Customers - Screen Mode** program on its own to scroll through the customers?
You are faced with an empty screen because the parameter value is zero.
How can you overcome this situation and display all of the records?

When you execute the **Customers - Screen Mode** program on its own, the parameter is zero, meaning that in the **Range from** value you have zero and in the **Range to** value you have zero. What you would like to be able to do is to range from the lowest value to the highest value. The lowest value is zero; therefore you only need to handle the highest value, which is 999999999.

19. Zoom into **Customers - Screen Mode** program.
20. Zoom into the **Range to** property.
21. Add an expression in the Expression Editor:
    IF (P.Customer Code > 0, P.Customer_Code, 999999999).

Now execute the **Customers - Screen Mode** program and you will be able to scroll through the records.

Another method of handling the expression is to use the **CndRange** function. This function enables you to provide a conditional expression in the minimum and maximum properties of the range and locate expressions.

Syntax	**CndRange (*condition, value*)**
Parameters	*Condition*: A condition that will be evaluated during runtime. A condition returns either True or False. *Value*: A value that will be returned if the condition evaluates to True.

You can use this function instead of the expression you defined above.

22. Set the expression to: **CndRange (P.Customer Code > 0, P.Customer_Code)**

If a Customer Code is passed as a parameter, the condition is evaluated to True and the value will be used in the range. If a code is not passed, the expression will be ignored and behave as if no expression exists.

It is useful to use this expression in the **from** and **to** properties.

Exercise

During the lesson you raised the **Set Gold Membership** event but there was no handler for it:

1. Update **Gold_Membership** with a True value whenever the **Salary_Amount** is larger than **50,000**.
2. As before, when the **Salary_Amount** is larger than **50,000**, increase the **Credit_Amount** by **20%**.

In the **Customers - Screen Mode** program:

3. Add a button that enables the user to delete a customer. There is a ready-prepared image, **Delete.png**, in the **images** folder for you.

   Note: The user can run this program both directly and indirectly, meaning that it is called from the **Customers - Line Mode** program. If the program is called from the **Customers - Line Mode** program, then once the customer is deleted, the program must be closed and focus will be returned to the calling program.

The next exercise is slightly more challenging:

4. Add a button that enables the user to add a customer. There is a ready-prepared image, **Add.png**, in the **images** folder for you.
   Hint: What task mode should you be in? How do you change the task mode? Initial Mode can be an expression.
   If you want to further your knowledge, look at Literals in the Magic xpa Help and then look at the MODE literal.

Summary

The event-driven methodology allows you to combine non-procedural operations in a task.

The event-driven programming is based on three elements: the event, its trigger, and the event handler.

- An event can be invoked by the following types of triggers:
    - System
    - Internal
    - Timer
    - Expression
- Another way to invoke an event is by using the Raise Event operation or by using the Push Button control.

User events are located in the task's Event repository.

The event handler is defined using the **Event** logic unit in the task's Logic Editor and it contains operations to be executed when the event is invoked.

You can define several handlers for the same event and determine which of them will be executed, by using the **Propagate** property.

You can also define a handler for Magic xpa internal events.

In interactive tasks, Magic xpa checks the event's execution condition whenever the task is idle or the end user triggers the event execution.

Using the Raise Event operation (Wait = Yes/No), the developer can decide whether the event will be synchronous or non-synchronous.

Lesson 11

Conditioning a Block of Operations

In some instances, you need to condition part of the code execution. In most cases, conditioning each operation will do the job. To create a more structured and efficient code it is recommended to use the Block operation. This allows you to group several operations with one execution condition.

This lesson covers various topics including:

- Block If, Block Else, Block End and Block While operations
- Grouping the execution of several operations within one condition

What Is a Block Operation?

Assume you have three operations, such as:

- Update Gold_Membership with True
- Update Salary_Amount with Salary_Amount * 1.1
- Update Credit_Amount with Credit_Amount * 1.2

Each of these operations will be executed if the Membership_Date is before 1/1/2000. You can add the same condition to each of the operations but that means that the same expression is evaluated each time, making the code inefficient.

The Block operation encloses a group of procedural operations within a logical block. The execution of all operations in the block depends on the condition of the Block operation.

Block If – Conditioning Operations

The **Block If** operation is used to enclose several operations within a certain condition. This type of Block operation consists of three parts:

- Block If
- Block Else
- Block End

When Magic xpa reaches the Block If operation, it checks the Block If condition.

- If it is True, the operations within the Block If operation are executed one by one.
- If it is False, Magic xpa skips the operations within the Block If operation.

Block Else

The Block Else operation is an extension of the Block If operation. It enables you to check several conditions within a single block unit.

You can define several Block Else operations within the same Block If operation. The Block Else operation is defined within the logical block of the Block If operation (between the Block If and the Block End operations).

When the condition of the Block If operation is not met, Magic xpa skips to the Block Else operation. If the Block Else condition is met, the operations within the Block Else operation (until the next Block Else operation or the Block End operation) are executed. Then, Magic xpa skips to the next Block Else operation and so on, until it gets to the Block End operation.

If the Block Else condition is not met, Magic xpa skips to the next Block Else operation, if it exists, or to the Block End operation.

Take into account that Magic xpa checks each **Block Else** condition. You may find that more than one Block Else condition may result in True. In this case, the operations within each of the Block Else sections will be executed.

A Block operation can be nested.

The Structure of the Block Operation

The following examples demonstrate various ways to use the Block operation.

Example 1	**Block If** Operation Operation **Block End**	This is mainly used to condition a group of operations with one condition. In a previous lesson, you had three operations with the same condition. You could have grouped these operations in the same Block operation. In that way, the condition would have only been checked once.
Example 2	**Block If** Operation Operation **Block Else** Operation Operation **Block End**	This is mainly used to perform a scenario in a specific case. If the condition is not met, the Else section (the default) will be performed.
Example 3	**Block If** Operation Operation **Block Else** Operation Operation **Block Else** Operation **Block End**	This is the aame as Example 2, but here there is more than one special scenario. (The Else section can also be conditioned.)

Example 4	**Block If**
	    Operation
	    Operation
	    Block If
	        Operation
	    Block End
	    Operation
	Block End
		This shows an example of a nested block.

Using the Block If Operation

In this section, you will practice using the Block operation. In Lesson 4, in **My First Program**, you added a **Variable Change** logic unit and within the **Variable Change** logic unit you performed three operations, each with the same expression.

In the **Customers - Screen Mode** program you will clear the value from the **City** and **Address** variables if the **Country** variable has a value and the value was changed.

1. Zoom into the **Customers - Screen Mode** program and open the Logic Editor.
2. Create a **Variable Change** logic unit for the **Country** variable.
3. Click **Yes** to automatically create the parameters.
4. In the **Variable Change** logic unit, add a **Block If** operation with the condition: **CHG_PRV_Country <> ''**

The **Block End** operation is created automatically when you create a **Block If** operation.

5. Park on the **Block If** line and create a line under it.
6. Update **City** with **''** – remember this is the way to reset an Alpha value.
7. Update **Address** with **''**.

The two Update operations will only execute if the condition of the Block If operation evaluates to True.

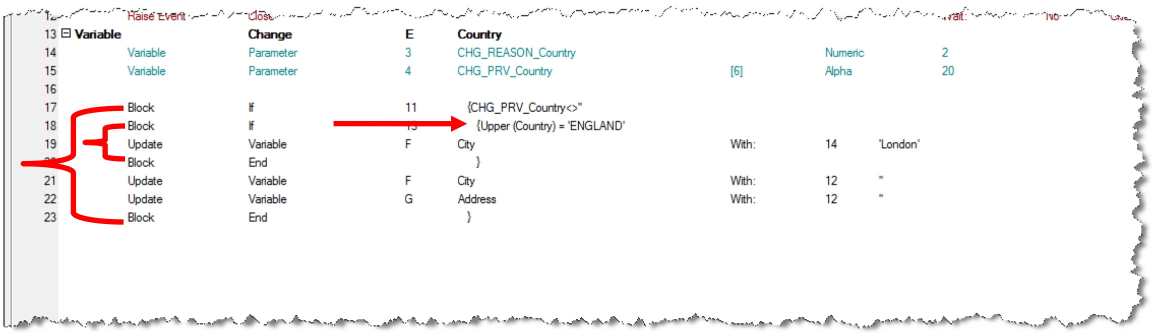
If you now execute the application and run the **Customers - Screen Mode** program and change the **Country** variable, you will see that both the **City** and **Address** values were cleared when you tapped another field.

As an example of a nested Block, if the value of the **Country** variable was changed to **England**, then update the **City** with **London**.

8. Zoom into the **Customers - Screen Mode** program, open the Logic Editor and zoom into the **Variable Change** logic unit for the **Country** variable.
9. Park on the **Block If** line and create a line under it.
10. Add a **Block If** operation with the condition: **Upper (Country) = 'ENGLAND'**.

The reason for using the expression Upper is because expressions are case sensitive. Here you are changing the value to uppercase. You will also notice that you were asked to check against the **Country** variable and not its previous value.

11. Park on the new **Block If** line and create a line under it.
12. Update **City** with '**London**' – remember this is the way to reset an Alpha value.

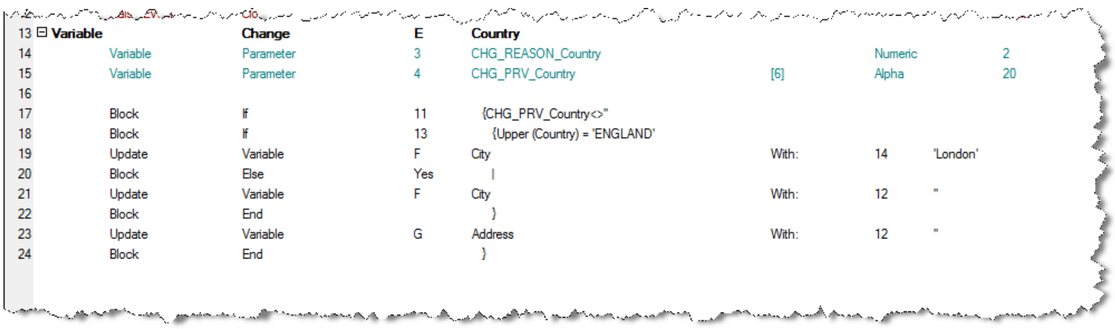


```
13 ⊟ Variable          Change       E    Country
14      Variable       Parameter    3    CHG_REASON_Country                        Numeric    2
15      Variable       Parameter    4    CHG_PRV_Country                 [6]       Alpha      20
16
17      Block          If           11    {CHG_PRV_Country<>''
18      Block          If           13     {Upper (Country) = 'ENGLAND'
19      Update         Variable     F    City                            With:     14         'London'
20      Block          End                 }
21      Update         Variable     F    City                            With:     12         ''
22      Update         Variable     G    Address                         With:     12         ''
23      Block          End                }
```

As you can see from the image above, there are two Block If operations, one directly after the other. The second Block If operation is a nested block. The second Block operation is displayed by the Logic Editor as nested. The nested block encompasses operations from the Block If to the Block End.

As you may already have understood from the task logic above, after exiting the inner block, the **City** variable will be cleared even though the nested block may have updated the value with **London**.

13. Park on the **Update Variable** line where you updated the **City** with '**London**' and create a line.
14. Add a **Block** operation and select **Else**.
15. Create a line beneath the **Block Else** operation.
16. Update **City** with ''.
17. Park on the line after the first **Block End** operation where you first updated **City** with '' and delete the line by pressing **F3**. In a later lesson you will learn how to copy lines.



```
13 ⊟ Variable          Change       E    Country
14      Variable       Parameter    3    CHG_REASON_Country                        Numeric    2
15      Variable       Parameter    4    CHG_PRV_Country                 [6]       Alpha      20
16
17      Block          If           11    {CHG_PRV_Country<>''
18      Block          If           13     {Upper (Country) = 'ENGLAND'
19      Update         Variable     F    City                            With:     14         'London'
20      Block          Else         Yes    |
21      Update         Variable     F    City                            With:     12         ''
22      Block          End                 }
23      Update         Variable     G    Address                         With:     12         ''
24      Block          End                }
```

There are many ways of doing what you just did. A nested Block If scenario was used as an example.

Advantages of the Block Operation

The Block operation encloses a group of procedural operations into a logical block, so that all the operations within the block are dependent on the same condition, which is the Block operation condition.

Using the Block operation has several advantages:

- As a developer, it saves you time since you can set a condition for several operations, instead of setting the condition for each of the operations separately. In addition, it makes the maintenance of the program easier. If you need to change the condition, you only need to change it once, in the Block operation.
- For the engine, it saves multiple checks when several operations depend on the same condition. When you use the Block operation, the engine checks the condition once, when it gets to the Block operation and not every time it gets to each of the operations.
- Using the Block operation enables you as a developer to simplify the expressions. You can divide complex expressions into several logical sections and set each of them as the condition of a Block operation.
  For example, if you want to set a Verify Warning operation when the *country is Italy and the city is Rome*, you can use:
  ```
  Block If  (Cnd: Country='ITALY')
     Block If (Cnd: City='ROME')
        Verify
     Block End
  Block End
  ```
- The Block operation enables you to nest operations. If you need to execute several operations under a certain condition and then, some of the operations need to be executed under another condition, you can use a Block operation with a condition and within it use another Block operation with another condition. In this case, all the operations within the embedded Block will be executed only if both of the Block conditions are met.
  In the above example, the **Verify** operation will be executed only if:
  Country='ITALY' and City='ROME'.

The Block While Operation

The Block While operation instructs Magic xpa to repeatedly execute the operations within the block for as long as the Block condition is evaluated to TRUE.

Magic xpa evaluates the Block While condition. If the condition is met, the operations within the block are executed one by one until the Block End operation is reached. Then, the condition is evaluated again and if it is met, the operations are executed again, and so on. This creates a loop effect.

Once the condition is not met, Magic xpa skips to the Block End operation and continues the task flow.

LoopCounter() is a special function that returns the number of times the loop has iterated. This means that you do not have to create a special counter for each loop.

As an example you are going to check the value of the address entered by the user to see if the user entered the at sign (@). If @ exists in the string, you will give a warning.

In this example, you will do this be checking each letter in the address to check whether it is an @ or not. To do this you will use two new functions: **Len**, which returns the length of an Alpha string, and **Mid**, which extracts a specified number of characters from an Alpha string.

Syntax	**Len (*string*)**
Parameters	*string*: An Alpha string.
Returns	The length of the string.
Note	If you are using an Alpha variable of size 20, then Len (A) will return 20 regardless of what is in the string. It is advisable to use this together with the Trim function to remove trailing blanks.

Syntax	**MID (*string,start,length*)**
Parameters	*string*: An Alpha string. *start*: A number representing the starting position of the substring *length*: A number representing the number of characters to fetch
Returns	A substring of the other string.

Now you will see how to use these functions:

1. Zoom into the **Customers - Screen Mode** program.
2. Open the Logic Editor and add a **Variable Change** header line for **Address**.
3. In the dialog box that pops up about adding parameters, click **No**. You do not need the parameters for this example.
4. Create a Detail line.
5. Add a **Block While** operation and set the condition to:
   LoopCounter () <= Len (Trim (Address))
   If Address contains the value 'Magic xpa', then Len (Trim (Address)) will return 9 and the block logic unit will execute 9 times. Remember that the LoopCounter function returns the current iteration of the block.
6. Create a Detail line after the **Block While** operation.
7. Add a **Verify Warning** operation and set the warning text to:
   "**The address contains the invalid @ character.**"
8. Zoom from the condition and set the following expression:
   MID (G, LoopCounter(),1) = '@'
   The MID function returns a substring of the address starting from the value returned by the LoopCounter function.

Magic xpa has a function named **Instr**, which you can use instead of the loop. Read more about this function in the *Magic xpa Help*.

Exercise

When adding a new customer, update the **Credit_Amount** value according to the following logic:

- For all customers whose **Salary_Amount** is more than 8000:
    - For customers who do not have a **Gold Membership**, the **Credit_Amount** value will be updated with the **Salary_Amount*2**.
    - For customers who have a **Gold Membership**, the **Credit_Amount** value will be updated with the **Salary_Amount*3**.
- If the **Salary_Amount** is more than a 1000, but less than 8000, update the **Credit_Amount** with the **Salary_Amount**.
- If the **Salary_Amount** is less than a 1000, update the **Credit_Amount** with 50% of the **Salary_Amount**.

Hint: Use the **Stat** function to determine whether or not the program is in Create mode.

Remember that you can only add a new customer from the **Customers - Line Mode** program.

Summary

Block operations help you simplify your expressions, save you the time of setting the same expression in several places, reduce the program maintenance, shorten the engine execution time, and provide you with a logical way to nest operations under certain conditions. The Block operation is used for two reasons:

- Conditioning a group of operations using Block If and Block Else.
- Performing a group of operations in a loop, as long as the condition is met using Block While.

You learned about some new functions, including Len, MID and LoopCounter.

Lesson 12

One-to-One Data Relationships

This lesson introduces you to one-to-one data relationships and describes how they are implemented in Magic xpa.

Data sources are designed to contain data for specific subjects, such as:

- The **Customers** data source, which contains customer information, such as Name, Address, and Phone Number.
- The **Items** data source, which contains item information, such as Name, Price, Quantity, and Supplier/Manufacturer.
- The **Orders** data source, which contains order information, such as Customer, Order Data, Total Price, and Discount Percent.

In many cases there are relations between the data sources. For example, in the **Orders** data source there is a customer field that specifies who is the customer in this order.

To validate the customer's existence in the system, or to display more information about the customer, you connect to the **Customers** data source and search for the order's customer in the **Customers** data source.

In Magic xpa, when you need to process a specific record for each record in the Main Source it is called a one-to-one data relationship and it is done using the Link operation in the task's Data View Editor. This lesson describes how to use the Link operation to create a one-to-one data relationship.

This lesson covers various topics including:

- Link types
- The Recompute mechanism
- The Link Success Indication property

One-to-One vs. One-to-Many Data Relationships

Magic xpa enables you to establish the following two relationships between data sources:

- One-to-one data relationships
- One-to-many data relationships (You will learn about the one-to-many data relationships in a later lesson.)

The following table compares the one-to-one and the one-to-many data relationships:

One-to-One Data Relationship	One-to-Many Data Relationship
Common variables are used to maintain the connection.	Common variables are used to maintain the connection.
The Main Source and the linked data source are defined in the same task.	Each data source is defined as the Main Source in a different task.
Only one record is returned from the linked data source.	Several records can be returned for each record.
The Magic xpa Recompute mechanism is responsible for maintaining the connection.	The task's range is responsible for maintaining the connection, according to a passed parameter. In addition, the Magic xpa Subform control mechanism is also responsible for maintaining the connection. You will learn about the Subform control in a later lesson.

Linking to Other Data Sources

Magic xpa enables you to connect between data sources to establish the one-to-one data relationship by defining the Link header line in the Data View Editor.

To connect between two data sources, both data sources need to contain a common variable. Usually the common variable is an index segment in the secondary data source.

During task execution, Magic xpa fetches a single record from the linked data source for every record of the Main Source. Magic xpa fetches the first record that meets the condition. Remember that there may be more than one.

Link Header Line

Linking to other data sources is part of the data view definition in Magic xpa. Therefore, the Link operation is defined in the Data View Editor.

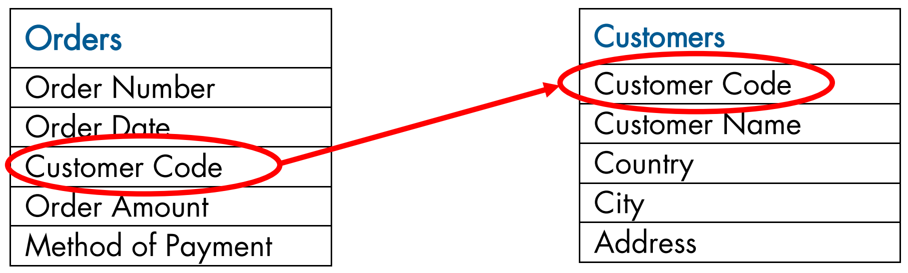
The Link definition consists of the following elements:

- The **Link header line** definition, which includes:
    - The linked data source
    - The Link type
    - The index by which the data sources are linked
- Within the Link section, detail lines are added that define the linked data source columns.
- Locate criteria, which are defined for the common variables in the linked data source.
- An **End Link** operation, which closes the Link operation.
- All of the columns between the **Link** operation and the **End Link** operation. These columns are all part of the same data source.

Link Operation Usage

The Link operation is used to:

- Extend the record's data view.
- Check the existence of a particular record in linked data sources (used to perform validity checks).



Extending the Record's Data View

In some cases, you need to extend the information you have from the Main Source by retrieving additional information from other data sources. This feature is enabled using the Link operation.

Suppose that you have the following data sources in your project:

- **Orders** – includes order number, order date, customer code, order amount, and method of payment
- **Customers** – includes customer code, name, country, city, and address

You can create a program that displays the **Orders** data source records. The program displays a customer code for each order. To display the customer name in the same program, you need to extend the program's data view. You need to link to the **Customers** data source by locating the customer according to the customer code and displaying the customer name.

Perform Validity Checks

Another usage of the Link operation is to check the existence of a value in a data source (validity checks).

Using the previous example, the **Orders** data source contains a customer code, while the **Customers** data source contains all of the customers' details.

When adding a new order, the end user will be asked to enter a customer code as part of the order details.

The information that the end user enters should be verified according to the information already existing in the system. You can use the Link operation to verify the customer code that the end user entered against the **Customers** data source.

Link Types

Magic xpa has five types of Link operations; each with a different behavior. The following are the Link operation types:

- **Link Query** – This operation is used to establish a connection to a record in the linked data source. If the link fails (because the record does not exist in the linked data source), no record is displayed.
- **Link Write** – This operation is similar to the Link Query operation. However, in the Link Write operation, if the linked record does not exist, Magic xpa will attempt to create a record in the linked data source.
- **Link Create** – Magic xpa creates a new record in the linked data source. If the indexed record exists, you will receive errors if the indexes are unique.

The following two Link types are used for SQL databases. They are not covered in this course:

- **Link Inner Join** – The program's data view includes only records that have an existing linked record in the linked data source.
- **Link Left Outer Join** – This operation behaves the same as the Link Query operation; however, the preparation of the data view is different. The program's data view includes all of the Main Source records, regardless if a linked record exists in the linked data source.

Using Link Query

In this section you will create an **Orders** program that will display the **Orders** data source with additional information from the **Customers** data source (using the Link operation).

This example involves three main steps:

- Defining the **Orders** data source
- Creating the **Orders** program
- Extending the **Orders** program data view by linking to the **Customers** data source

Defining the Orders Data Source

1. In the Data repository, create a data source and name it **Orders**.
2. In the **Data source name** column, type: **Orders**.
3. In the **Database** column, select **Getting Started**.
4. Create the following columns:

Name	Model	Attribute	Picture
Order Number	0	Numeric	6
Order Date		Date	##/##/####
Customer Code	Code	Numeric	9
Amount	0	Numeric	8.2
Method Of Payment	0	Alpha	15

Setting a Rich Client Display Control

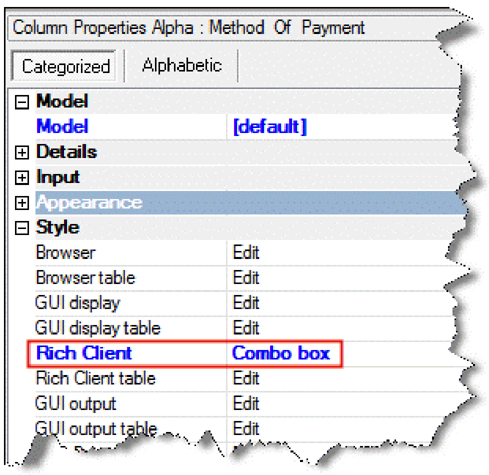
As you know, Magic xpa enables you to set the default appearance for each column. This is performed in each column's Style properties.

Multiple choice controls allow the user to see all of the possible choices, and prevent the user from accidentally entering any incorrect or invalid values.

A combo box, also known as a drop-down box, is a choice control that only displays one value, rather than all of the possibilities, until the user opens up the box. Typing the first character selects the first item matching that character.

In the **Orders** program, you want the **Method Of Payment** column to be displayed using a **Combo Box** control every time you place the column on a GUI form.

1. Park on the **Method Of Payment** column.
2. Open the Column Properties.
3. From the **Rich Client** style property, select **Combo box**.


Column Properties Alpha : Method Of Payment

Categorized | Alphabetic

⊟ **Model**	
Model	**[default]**
⊞ **Details**	
⊞ **Input**	
⊞ Appearance	
⊟ **Style**	
Browser	Edit
Browser table	Edit
GUI display	Edit
GUI display table	Edit
Rich Client	**Combo box**
Rich Client table	Edit
GUI output	Edit
GUI output table	Edit

About the Items List and Display List Properties

The **Items List** property lets you set a string that defines the options that can be selected from the choice control. The various options are delimited by commas.

The **Display List** property lets you display values that are different from the ones in the **Items List** property. The Display List string must contain an identical number of items to the Items List string. For example, if you enter "1,2,3" in the Items List and "Red,Green,Blue" in the Display List, the user will see the options Red, Green, and Blue in the combo box, but internally the variable will have the values 1, 2, or 3 according to the selected color.

When setting the **Rich Client** property to **Combo box**, the **Items List** is sometimes a set list (such as a combo box for Gender). In these cases, the list can be set during the creation of the column in the data source.

Moreover, Magic xpa enables you to differentiate between the value that the end user sees and selects (**Display List**) and the one that will be stored in the column (**Items List**). In this course you will only use the **Items List**.

In this example, you will display the following options in the **Method of Payment** column: **Cash**, **Check**, **Credit Card**, and **Coupons**.

1. From the **Rich Client** entry, click the **Zoom** button (…) to open the **Combo box** control property sheet.
2. In the **Items List** property, type: **Cash, Check, Credit Card, Coupons**.
3. Set the **Color** property to **Edit Control** color.
4. Close the **Combo box** control property sheet.

Defining the Order Number Index

You will now define two indexes for the **Orders** data source.

1. Create a **Unique** index called **Order Number**.
2. Select **Order Number** as the first segment.
3. Create a **Unique** index called **Customer** with the following segments:
    - Customer Code
    - Order Date
    - Order Number

You have now finished creating the data source.

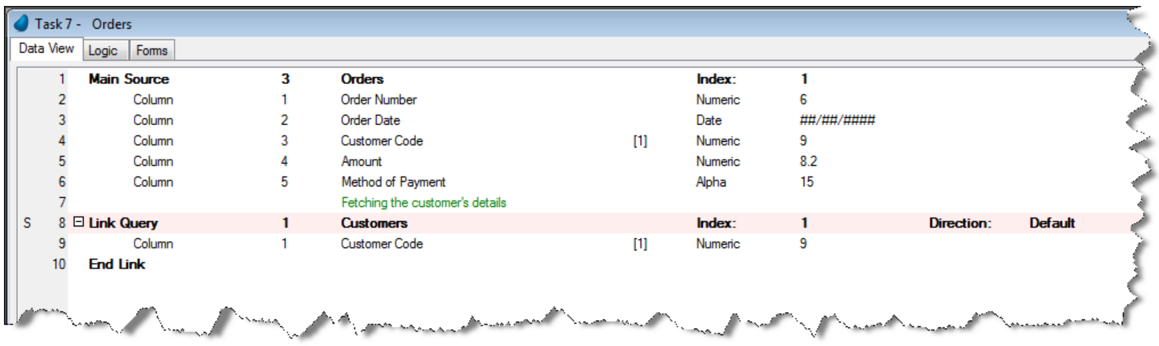
Creating the Orders Program

An order usually has two parts, the order details and the order lines. In this lesson you will display the details and in another lesson, you will complete the program. The order details can be displayed in a screen mode form.

1. Create a program and name it **Orders**. Remember to set the **Public name** to **RunMe** and to check the **External** box.
2. Zoom to the **Orders** program.
3. From the **Task Properties** dialog box, set the **Task type** to **Rich Client**.
4. Set the **Initial Mode** to **Query**.
5. Open the Data View Editor of the **Orders** program.
6. Create a Main Source definition for the **Orders** data source and use the **Order Number** index.
7. Add all of the **Orders** data source columns to this program.

Linking to the Customers Data Source

1. Park on the last line in the Data View Editor.
2. Add a Remark line.
   Note: A Remark line is not needed for the program to work; however, it makes the program more readable.
   Enter the following remark: **Fetching the customer's details**
3. Create a **Header line** and from the combo box, select **Link Query**.
4. Select the **Customers** data source.
5. Zoom from the **Index** property and select the first index. You will see that the **Customer Code** was added automatically.



	1	**Main Source**	**3**	**Orders**		**Index:**	**1**		
	2	Column	1	Order Number		Numeric	6		
	3	Column	2	Order Date		Date	##/##/####		
	4	Column	3	Customer Code	[1]	Numeric	9		
	5	Column	4	Amount		Numeric	8.2		
	6	Column	5	Method of Payment		Alpha	15		
	7			Fetching the customer's details					
S	8	⊟ **Link Query**	**1**	**Customers**		**Index:**	**1**	**Direction:**	**Default**
	9	Column	1	Customer Code	[1]	Numeric	9		
	10	**End Link**							

The **End Link** operation was created automatically when you added the **Link** operation.

When you create a **Link** operation, the **Index Segments** are automatically added as columns within the Link section.

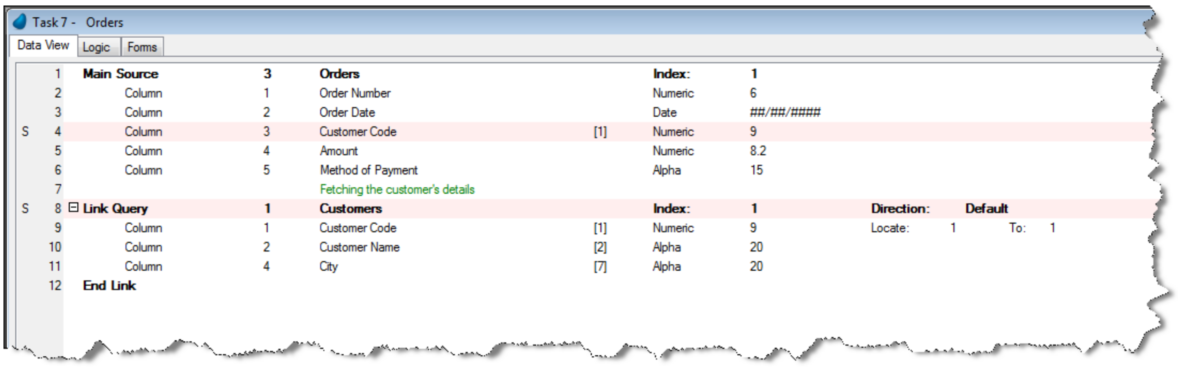
The **S** to the left of the link is an indication that this is a Server operation. This will be discussed in a later lesson.

6. Park on the **Customer Code** column definition.
7. Create a line.
8. Add the **Customer Name** and the **City** columns from the **Customers** data source.
   You can add any column from the **Customers** data source. The ones you were asked to add were selected simply to show the power of the link mechanism.

Defining Locate Parameters

In this example, the **Customer Code** column is the common variable between the two data sources. To set the **Orders** data source's **Customer Code** as the **Locate** parameter in the **Customers** data source's **Customer Code** column:

1. Park on the **Customers** data source's **Customer Code** column definition.
2. From the **Locate** entry, zoom to the Expression Editor.
3. Create an expression for the **Customer Code** variable from the **Orders** data source. You will see that in this program there are two **Customer Code** variables. You are defining a link between the two variables.
4. In the **To** entry, enter the same expression number.


Task 7 - Orders

	1	**Main Source**	**3**	**Orders**		**Index:**	**1**					
	2	Column	1	Order Number		Numeric	6					
	3	Column	2	Order Date		Date	##/##/####					
S	4	Column	3	Customer Code	[1]	Numeric	9					
	5	Column	4	Amount		Numeric	8.2					
	6	Column	5	Method of Payment		Alpha	15					
	7			Fetching the customer's details								
S	8	⊟ **Link Query**	**1**	**Customers**		**Index:**	**1**	**Direction:**	**Default**			
	9	Column	1	Customer Code	[1]	Numeric	9	Locate:	1		To:	1
	10	Column	2	Customer Name	[2]	Alpha	20					
	11	Column	4	City	[7]	Alpha	20					
	12	**End Link**										

In the **Locate** and **To** columns, it is important <u>not</u> to select the **Linked** data source's common variable. This is why you selected the **Customer Code** form the **Orders** data source and not the **Customers** data source.
The variable used in the **Locate** and **To** expressions must be declared <u>above</u> the **Link** section in the Data View Editor.

Designing the Form

In this section, you will design the **Orders** program form. The form will display the order details and some of the customer details.

1. Open the Form Editor.
2. Park on the **Orders** form and zoom to the Form Designer.
3. Open the Form Properties sheet (Alt+Enter).
4. Select **Table Display Form** as the model.

You will now place controls on the form. See the image below as an example of the required layout. Please adhere to this layout as you will need it in a later lesson.

5. From the Task Variables pane, drag the **Order_Number** variable and drop it on the top left area of the form.
6. Place all of the variables on the form except for the **Customer_Code** from the **Customers** data source.
   Note that when you place the **Method_Of_Payment** variable, only a Combo Box control will appear.
7. Place the **Customer_Name** variable to the right of the **Customer_Code**
8. Delete the **Customer_Name** caption control.
9. Place the **City** variable to the right of the **Customer_name**. Delete the **City** caption control.

There are two **Customer_Code** variables, one from the **Orders** data source and one from the **Customers** data source. You only placed the variable from the **Orders** data source. The reason for this is:

- If the **Link** operation was successful, these columns will have the same value. There is no need to display the value twice.
- The **Orders** data source is the Main Source and is used for editing. The **Customers** data source is used to expand the Main Source and in this case, is used to display extra information. In essence, it is used here only for display purposes.

For the **Method_Of_Payment** combo box:

10. From the Toolbox, select a **Label** (Text) control and add it to the left of the **Method_Of_Payment** combo box.
11. Open the **Label** control properties and in the **Text** property, type: **Payment Method**.

Handling the Edit control captions

12. Select all of the Static controls (by holding down the **Control** key and selecting all of the Static controls, one at a time).
13. Open the property sheet for the selected controls.
14. Attach the controls to the **Text Caption** model.
15. Inherit the **Font** property for all of the controls.
16. While all the **Text** controls are selected, click the **Fit Control Size** icon (▣) from the toolbar.
17. Click on the form to de-select all of the controls.

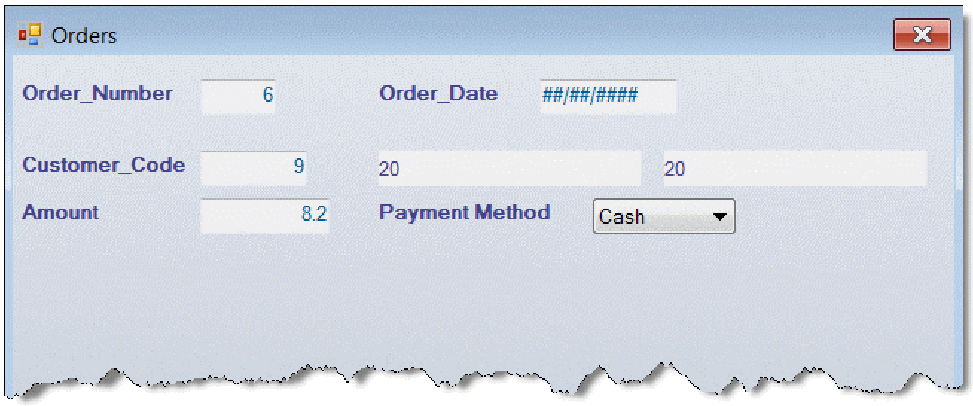
Handling the Dynamic controls

Controls belonging to the **Orders** data source are editable controls and those fetched from the **Customers** data source are for display. You will connect the Edit controls to different models.
In a previous lesson, you created two models: **Display only** and **Editable field**.

18. Attach the controls to the models as follows:

Model	Controls to Attach
Editable field	Order_Number, Order_Date, Customer_Code and Amount edit controls
Display only	Customer_Name and City edit controls

19. Move the **Edit** controls so that the entire value of the Label controls are displayed. Your form will look similar to the image below.


Orders

Order_Number 6 Order_Date ##/##/####

Customer_Code 9 20 20

Amount 8.2 Payment Method Cash

When you execute the application, you will find that all of the fields are empty and you cannot tap any control. This is because your program is in Query mode and there are no orders in the data source.

As you know from this course, the best practice for designing an application for a mobile device is to have a list of orders where you have buttons enabling you to add, edit and view an order.

You will practice this during the exercise.

Often the option to modify or delete a specific item will not be in the table but in the details screen. This depends on your own application design.

For the purpose of this example:

1. Zoom to the **Orders** program.
2. From the **Task Properties** dialog box, set the **Initial Mode** to **Create**.
   When the program runs, it will immediately go into Create mode.

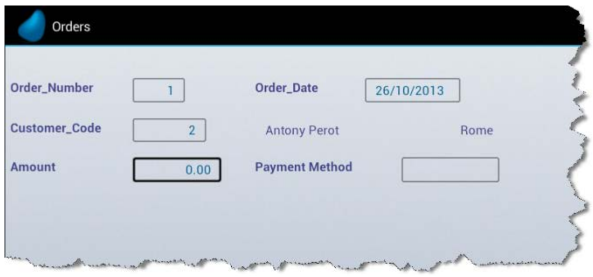
Execute the application.

3. In the **Order_Number** field, type: **1**.
4. In the **Order_Date** field, type the current date (such as 26/10/2013).
5. In the **Customer_Code** field, type: **2**.
6. Tap the **Amount** field.

The customer details are displayed for the customer code that you entered. You will probably see the details for Antony Perot who lives in Rome.

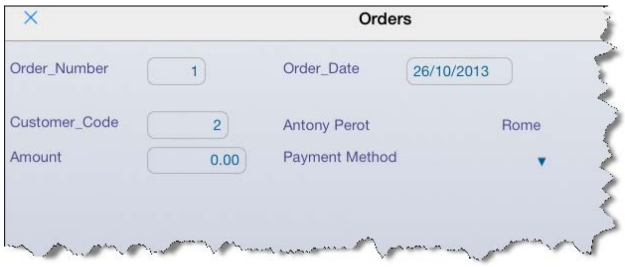
7. Change the customer code to **4** and see how the customer details are changed accordingly.

Android



Order_Number	1	Order_Date	26/10/2013	
Customer_Code	2	Antony Perot		Rome
Amount	0.00	Payment Method		

iOS



Order_Number	1	Order_Date	26/10/2013	
Customer_Code	2	Antony Perot		Rome
Amount	0.00	Payment Method		▼

8. Change the customer code to **25**. You will see that when the customer does not exist, the customer details are cleared.

9. Set the customer code to: **2** (Antony Perot).

If you tap the **Payment Method**, the device will open a drop-down box enabling you to select the type. The display depends on each operating system.

Short Summary

The data view of the **Orders** program that you created consists of:

- **Orders** data source columns
- **Customers** data source columns

Both data sources share a common column, the **Customer_Code** column. The data relationship between both data sources is one-to-one, since each order has a **Customer_Code** value that appears only once in the **Customers** data source.

You used the **Link Query** operation to retrieve the customers' details (if the linked customer exists).

The **Orders** form displays both the order details and some of the customer details.

Link Recompute Mechanism

In the previous section, where you executed the **Orders** program, you were asked to change the **Customer Code** value and view the result.

Each time the **Customer Code** was changed, the customer's details were also changed. Magic xpa re-evaluates the Link condition and attempts to locate the appropriate customer record in the **Customers** data source for the new **Customer Code** value.

This behavior is referred to in Magic xpa as **Link Recompute**.

The Link operation is only recomputed when all of the following occur:

- A variable is part of the Locate **From** and **To** expressions.
- The variable used in the Locate **From** and **To** expressions is declared <u>above</u> the **Link** section in the Data View Editor.
- The variable data has changed.

Link Success Indication

When you set the **Customer Code** value to **25**, you saw that the Customer details were cleared (nothing was displayed). This happened because the **Customer Code** number **25** does not exist.

The end user does not know if the customer details are empty because:

- The customer does not exist, or
- The customer does exist, but the information for the customer is cleared.

Since the end user does not have a clear indication of the customer's existence, you must provide one.

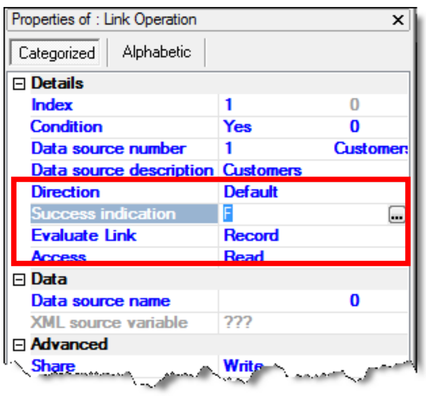
One of the Link operation properties is **Success indication**. This property returns a logical value.

To retrieve the Link **Success indication** value, you need to set a Logical variable in the property.

You can use the **Success indication** value to alert the end user when the link fails.

This is another method of validation; validating that the input is correct. Without using this method, you could add a non-existing customer code to the order, but this would create a data-integrity problem.

In this example, you will improve the **Orders** program by adding a failure notification when a customer cannot be found in the **Customers** data source.



Details		
Index	1	0
Condition	Yes	0
Data source number	1	Customer:
Data source description	Customers	
Direction	Default	
Success indication	F	
Evaluate Link	Record	
Access	Read	
Data		
Data source name		0
XML source variable	???	
Advanced		
Share	Write	

1. Open the Data View Editor of the **Orders** program.
2. Add a **Customer Exists** Logical Virtual variable *before* the **Link Query** operation.
3. Park on the **Link Query** line and open its properties.
4. From the **Success indication** property, zoom to the Variable list and select the **Customer Exists** variable.
5. Open the Logic Editor and create a **Control Suffix** logic unit for the **Customer_Code** control.
6. Create a Verify Error operation with the text: "Customer does not exist.".
7. Set the condition to **NOT Customer Exists**.

Now execute the application again.

8. In the **Order_Number** field, enter **2**.
9. In the **Customer_Code** field, type the number **25**.
10. Tap another field. An error message appears.

To leave the screen you must enter a valid customer code.

11. In the **Customer_Code** field, type the number **5**.

You have now actually added two orders, order #1 and order #2.

Short Summary

In the last section, you used the Link operation to validate data entry. For this purpose you used the **Success indication** property.

First, you defined a Virtual variable to store the **Success indication** property value.

Then, you defined a Control Suffix for the **Customer_Code** variable, which raised an error when the Link operation failed.

During the program execution, when an end user enters a non-existing customer code, Magic xpa alerts the end user that the customer code does not exist in the **Customers** data source and forces the end user to enter an existing customer code value.

Exercise

Complete the order scenario so that its behavior reflects the correct method of creating applications on a mobile device. This means:

1. Add parameters to the **Orders** program so that according to the parameter the user will be able to view a specific order, update a specific order or add a new order.
    a. When a new order is created, set the current date as the initial date.
    b. As an extra example, add a push button that deletes the current order. This is only visible when the program is in Modify mode.
    Hint: The visibility condition will be **Stat (0,'M'MODE)**.
2. Add a program named **List of Orders**, which displays a list of all of the orders.
    a. Display only the order number, order date and the customer's name.
    b. Add a push button that when pressed will add a new order.
    c. Add a push button that when pressed will enable the user to modify the current order.
    d. When the user taps a specific order in the table, the details of the order will be displayed.
    Hint: The mobile **tap** action raises the Magic xpa internal event named **Click**. What happens when you handle the **Click** event and call the **Orders** program?

You will now define the products for the course. You will:

3. Define the **Products** data source in the same way that you previously created data sources.
4. Create the **Products List** program to display all of the products.
5. Create the **Products** program to display a single product.
    a. Also display the supplier's details.
    b. If a user enters an invalid supplier code, then display an error.
    c. Add a button enabling the user to modify the current product.

6. The **Products** data source has the following fields:

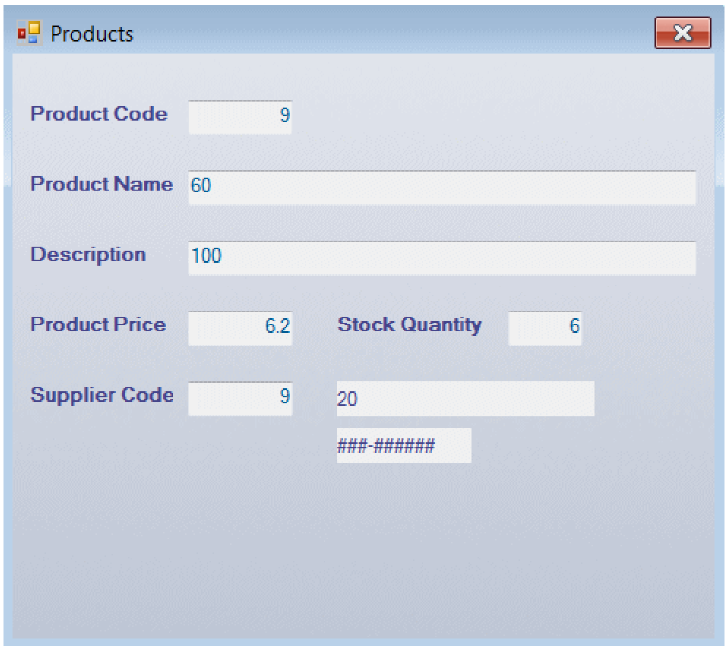
Name	Model	Attribute	Picture
Product Code	Code	Numeric	9
Product Name	0	Alpha	60
Description	0	Alpha	100
Supplier Code	Code	Numeric	9
Product Price	0	Numeric	6.2
Stock Quantity	0	Numeric	6

7. Define two unique indexes:
    a. **ProductCode** with the **Product_Code** index segment
    b. **SupplierCode** with the **Supplier_Code** and **Product_Code** index segments

You need to add data to the **Products** data source. You can do this by using Magic xpa's internal **Import** mechanism to import data. This imports a text file into the table.

8. Park on the **Products** data source.
9. From the **Options** menu, select **Generate Program**.
10. In the **Option** parameter, select **Import**.
11. In the **Text file** parameter, select the **Products.TXT** text file. This text file is located under the **Text** folder of the **Getting Started** project directory. In Lesson 6 you were asked to copy this directory. You can zoom to browse for the file.
12. The program will be added to the Program repository. Execute the program to import the data.

13. Add a new program that browses the **Products** data source.
14. Set up the form as shown in the image below.



Products

Product Code	9		
Product Name	60		
Description	100		
Product Price	6.2	Stock Quantity	6
Supplier Code	9	20	
		###-######	

15. Define placement on the **Product Name** and **Description** so that:

a. When the form increases, the width and height of the product name and the description increases.
b. All of the controls below the description must move so that they always appear below the description.

As with all programs, the product only displays a single product. Therefore:

16. Define a program named **All Products**, which displays a table with only the product name. When the user taps a product, the **Products** program will be called displaying that product. Both **All Products** and **Products** are display only programs.

Summary

This lesson introduced you to the Link operation, which is used to implement one-to-one data relationships.

The main use of this type of relationship is to connect a record of the program's **Main Source** to a specific record of another data source.

The Link operation is used to:

- Extend the record data view by adding variables of linked data sources.
- Perform validity checks for the entered data and check the existence of a particular record in linked data sources.
- Modify or create records in the linked data source, according to the link criteria.

You were introduced to the various Link types:

- Link Query
- Link Write
- Link Create
- Link Inner Join
- Link Left Outer Join

In this lesson, you used the Link Query operation to extend the **Orders** program's data view and perform validity checks for the **Customer_Code** value using the **Success indication** property.

In the exercise, you created the **Products** program, which displays the product details and some of the supplier details, using the Link Query operation. You also added a validity check for the **Supplier Code** value using the **Success indication** Link property.

Lesson 13

Selecting Data from a List

In this lesson you will learn how to create and call a program that allows the end user to select a value from a list. You will also learn about the Data control, which allows you to select data from a data source using a Combo Box control, and how to determine when to use a Data control instead of a selection list.

This lesson covers various topics including:

- Selection tables
- Defining a Data control

Selection List

In the **Orders** program from the previous lesson, the end user had to type in the Customer Code and only then the customer details appeared.

Having the end user type in the Customer Code without knowing the Customer Name is not a good idea. This means that the end user needs to memorize the customer codes. In addition, in most cases the Customer Code has no real meaning to the end user, but the Customer Name usually does have meaning.

The proper solution for the above is to let the end user select a value from a list.

By showing the end user all of the available information such as the customer's name, they can then select an entry without the frustration of having to guess the appropriate value.

A selection list is an interactive program that displays its data view as a list and enables the end user to select a value from the list.

Using a selection list involves two parts:

- Creating a Selection List program.
- Calling the selection list from the host program.

In this example you will create a program that displays the **Customers** data source content (Customer Code and Customer Name) as a selection list.

1. Create a program named **Select Customers**.
2. From the **Task type** property, select **Rich Client**.
3. From the **Initial mode** property, select **Query**. A selection list is used to select values from a list. Other operations, such as modifying or creating new records, should not be used here. Therefore, it is better to set the task's **Initial mode** to **Query**.
4. From the **Selection table** property, select **Yes**.

By setting the **Selection table** property to **Yes**, Magic xpa executes the **Record Suffix** logic unit and exits the program as soon as the **Select** event is raised. This special task engine behavior includes the following stages:

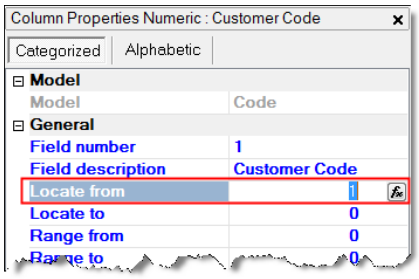
- Updating the selected value as a return parameter
- Closing the current program
- Returning to the calling program

1. Open the Data View Editor.
2. Set the Main Source to **Customers** and use the **Customer_Code** index.
3. Add the **Customer_Code** and **Customer_Name** columns to the program.
4. Park on the first line, the Main Source definition, and create a line.
5. From the drop-drown list, select **Parameter**.
6. Set the name to **P.Customer code** and select the **Code** model.

Setting Locate Criteria

The Locate criteria enable you to locate a specific record and have the cursor park on the matching record when the program starts. You will now set the **Customer Code** parameter as the **Locate from** criteria, so that if the calling program sends a parameter, Magic xpa will search for this customer and the cursor will park on that record when the selection list opens.

1. Park on the **Customer_Code** column.
2. Open the **Column Properties** and zoom from the **Locate from** property to the Expression Editor.
3. Create the following expression: **P.Customer Code**.


Column Properties Numeric : Customer Code

Model	
Model	Code
General	
Field number	1
Field description	Customer Code
Locate from	1
Locate to	0
Range from	0
Range to	0

Returning the Selected Value

As was mentioned before, when you set a program's **Selection table** property to **Yes**, Magic xpa executes the **Record Suffix** logic unit as soon as the end user selects a value by raising the **Select** event.

Now you will use this unique behavior to update the parameter with the selected Customer Code.

1. In the Logic Editor, create a **Record Suffix** logic unit.
2. Create an **Update Variable** operation.
3. Update the **Customer code** parameter with the **Customer_Code** column.

The Selection List Form

Most of the Selection List program's forms are similar: a Table control with two columns (the Code column and the Description or Name column), and two push buttons: Select and Exit. Now you will create the **Customers** selection list's form according to the above description.

1. Open the Form Editor.
2. Park on the **Select Customers** form.
3. Attach the form to the **Table Display Form** model.
4. Zoom to the Form Designer.
5. Drag the Table control onto the form. Set the **Model** property to **Table**.
6. Select the **Customer_Code** variable and place it on the table. This is already linked to the **Edit control** model.
7. Select the **Customer_Name** variable and place it on the table. This is also already linked to the **Edit control** model.
8. From the **Customer_Code** column's **Column Title** property, change the name to **Code**.
9. Select the **Column Heading** model.
10. Set the **Customer_Name** column's model to the **Column Heading** model.
11. Increase the width and the height of the table to display some records.

Adding a Select Button

As was mentioned before, the unique behavior of a Selection List program is achieved when the **Select** internal event is raised.

12. Place a Button control at the top of the form.
13. Zoom into the control properties.
14. Park on the **Format** style and remove the text, which is initially **Button**.
15. Park on the **Button style** property and select **Image Button** from the combo box.
16. Park on the Image List file name property and type **%WorkingDir%\images\OK.png**.
17. From the **Event Type** property, select **Internal**.
18. From the **Event** property, select **Select**.
19. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
20. From the toolbar, click the **Fit Control Size** icon.

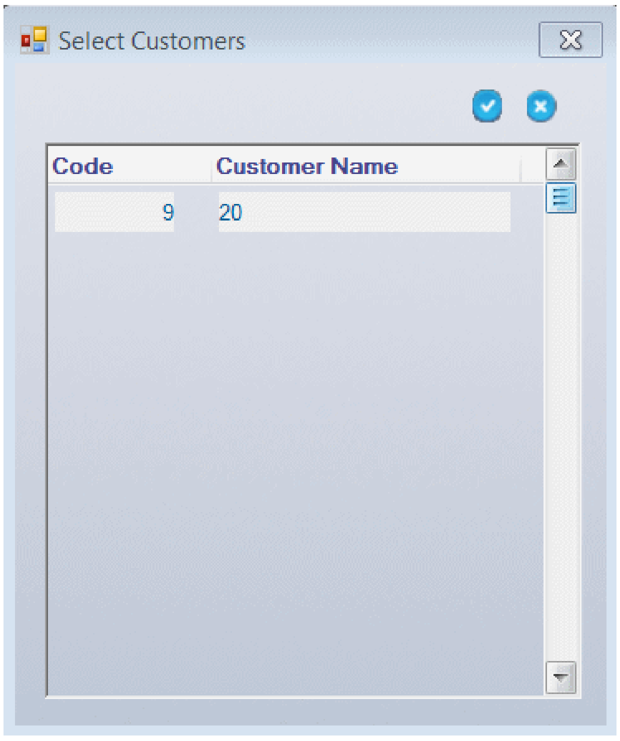
Adding the Exit Button

You can also add an exit button.

21. Place a Button control at the top of the form.
22. Zoom into the control properties.

23. Park on the **Format** style and remove the text, which is initially **Button**.
24. Park on the **Button style** property and select **Image Button** from the combo box.
25. Park on the **Image List file name** property and type: **%WorkingDir%\images\Exit.png**.
26. From the **Event Type** property, select **Internal**.
27. From the **Event** property, select **Exit**.
28. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
29. From the toolbar, click the **Fit Control Size** icon.

The form will look similar to the image below.



Calling the Selection List

Creating the selection list is only part of the process of using a selection list. You need to call this list from a different program.

You will now set an event that will be raised in the **Orders** program. The event handler will call the **Select Customers** program and pass the order's **Customer Code** value as a parameter.

1. Zoom to the **Orders** program and select **User Events**
2. Create an event named **Select Customers** and from the **Force Exit** column, select **Editing**.

The **Force Exit** column is set to **Editing** so that the new selected value will be displayed in the **Customer Code** control after the selection list is closed.

Now you will create an **Event** logic unit that will call the **Select Customers** program and send the Customer Code value as an argument.

3. In the Logic Editor, create a Header line for the **User** event, **Select Customers**.
4. Create a line for the **Call Program** operation. This operation enables you to call a program and pass parameters to the new program.
5. Call the **Select Customers** program.
6. From the **Arguments** field, zoom to the Arguments repository.
7. Create a line, zoom and select the **Customer_Code** variable from the **Orders** data source.
8. Zoom into the Form Designer.
9. Add a push button to the right of **Customer_Code**.
10. Zoom into the control properties.
11. Park on the **Format** style and type … (three dots).
12. From the **Event Type** property, select **User**
13. From the **Event** property, select the **Select Customer** event.
14. Set the **Visible** property to **NOT (Stat ('Q'Mode))**. This means that it will not be visible when the program is in Query mode.
15. Set the **Width** property to **3.5**.
16. Set the **Height** property to **1.5**.

You will now run the **Orders** program and learn how to use the selection list that you just created.

1. Remember to give the **List of Orders** program the public name **RunMe**.
2. Execute the application.
3. Park on any order and tap the **Edit** button.
4. The **Orders** program will appear.
5. Tap the browse button that you defined to the right of **Customer_Code**.

The **Select Customers** window opens, enabling you to select a customer from the list.

Note that the customer number that appeared in the order is highlighted. This is because you have located the customer according to the argument from the **Orders** program.

5. Tap on a different customer and tap the **Select** button.

You can see that the **Customer_Code** value was updated with the selected Customer Code value.

Data Control

A selection list is an all inclusive solution for selecting data from a list. However, in some cases, when the list to select from is short, it is more efficient to use a **Data control**. Like a Selection List program, a Data control enables the end user to select a value from a list.

A Data control is a Combo Box control that displays one column from a data source.

Magic xpa builds the control's list of options during runtime, according to the properties specified in the Control Properties.

Unlike a Selection List program, a Data control is a control within the central program and not a separate program.

By using the Data control:

- You do not need to create a separate Selection program and you do not need to call the Selection program.
- You do not need to validate the end-user selection.
- The user can select a value from a list, instead of typing a value.
- You do not expose internal codes in your programs; the user will only see the description or name.
- Like a selection list, you can type the value in the Data control to perform an Incremental Locate.

Using a Data control has some restrictions, such as:

- The list should be short.
- You can only display one column from the data source.

As an example, you will display the suppliers' list in the **Products** program as a Data control. In most cases, selecting a supplier using a Data control is not reasonable, since you have many suppliers. However, in this course, the **Suppliers** data source only has a few records; so for practice purposes only, you will use this data source with a Data control.

1. Zoom into the **Products** program.
2. Zoom into the Form Editor.
3. Zoom from the **Orders** form into the From Designer.
4. Select the **Supplier_Code** Label control.
5. Open the Control Properties.
6. Park on the **Text** property and change it to: **Supplier**.
7. Delete the **Supplier_Code** Edit control.
8. Delete the **Supplier_Name** Edit control.

Adding the Data Control

9. From the Toolbox, drag a Combo Box control ( ) and drop it to the right of the **Supplier** Label control (where the **Supplier Code** Edit control was).
10. Open the Combo Box control properties.
11. From the **Data** property, select the **Supplier_Code** variable (from the **Products** data source). This was previously used as an Edit control.
12. From the **Data Source number** property, zoom and select the **Suppliers** data source.
13. From the **Display field** property, zoom and select the **Supplier_Name**.
14. From the **Linked field** property, zoom and select the **Supplier_Code**.
15. From the **Index** property, zoom and select the **Supplier_Code** index.
16. Increase the **Width** of the control to **30** so that all of the data will be visible. You may need to increase this even further according to your device.

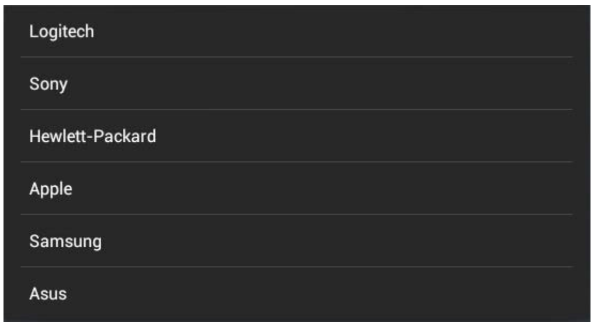
Remember that you set placement for other controls, so you need to set placement for this control as well. You only need to move the control if the height of the form increases.

17. Zoom into the **Placement** property and enter **100** in the **Y** property. The placement will look like this: **{0,0,100,0}**.
18. Remember to give the **All Products** program the public name **RunMe**.
19. Select a product from the list.
20. In the **Products** program, tap the **Edit** button.

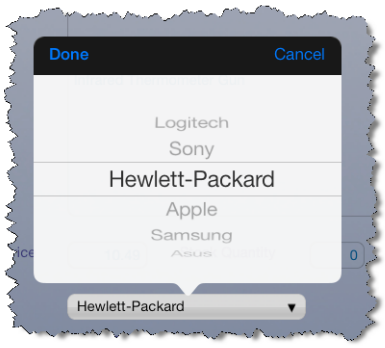
You can see that the Data control displays the supplier name value since you set the **Supplier_Name** as the displayed field.

21. Park on the **Supplier** combo box and select a different supplier. The look and feel of the Data control is governed by the operating system.

Android



iOS



Remember that Magic xpa will internally save the **Supplier_Code** variable and not the supplier's name that is displayed.

Exercises

As you will be using push buttons more and more, it is good practice to define models for them.

1. Create a model named **Browse button**, for the Browse button with the same properties as the button you defined during this lesson. You can use the Internal event, **Zoom,** as the initial Raise event.
2. Create a model named **Image button**, for the image button with the same properties as the button you defined during this lesson. You can use the **Edit.png** image as the default image and the Internal event, **Zoom,** as the initial Raise event.

You will now practice creating a selection list. In a previous lesson, you copied the **products** folder to your project directory. This folder contains image files for the products. Each image file name has the same name as the product name but with a **jpg** extension, such as **Apple 2 GB iPod Nano.jpg**.

3. In the **Products** program, add the display of the product's image.
4. For use in the later lessons, create a selection list for the **Products** data source.
5. This selection list will be similar to the other selection lists that you created, but it will display the image as a thumbnail and the product name in the table.
6. As an extra example, if the user taps a row in the table, it will display the **Products** program so that they can see more details.

Summary

In this lesson you learned about the selection list and the Data control.

Selection Lists

A selection list is a Magic xpa interactive program that is used to enable the end user to select a value from a list. The main steps in creating a Selection List program are:

- Setting the program as a **Query** only program.
- Setting the **Selection table** property to **Yes**.
- Defining a Main Source.
- Defining a parameter to return the selected value.
- Setting the Locate criteria.
- Creating the program form, including **Select** and **Exit** buttons.

When the end user selects a value from the Selection List program, Magic xpa's behavior is:

- Magic xpa executes the Record Suffix logic unit and the Task Suffix logic unit.
- The program execution is terminated.
- The parameter is returned to the calling program.

Data Control

A Data control is a Choice control that is used to enable the end user to select a value from a list. The Data control displays a list of values, which are records of a data source.

A Data control saves development time as well as making the end user's interaction easier.

A Data control, as opposed to a selection list, has some limitations, such as being limited to one column in a data source and only being useful when there are only a few options to select from.

Lesson 14

One-to-Many Data Relationships

In a previous lesson, you learned about the one-to-one data relationship. Another common type of data relationship is the one-to-many data relationship. In this lesson you will learn about one-to-many data relationships, and how to implement them in Magic xpa.

A one-to-many data relationship exists when one record from one data source is related to several records from another data source. In Magic xpa, a one-to-many relationship is implemented by using two tasks or programs.

In this way, the first task (usually called the parent task) displays the main record and the second task (usually called a subtask) displays the "many" or multiple records. This relationship is often displayed to the end user using two different display formats, the parent program displaying the single record in screen mode and the child program displaying the multiple records in a table. This is not always the case, since you may decide that the parent program will also be a table. During this course you will only use the method of displaying the parent in screen mode.

Magic xpa uses a Subform control to display the child task's form within the parent task's form and refreshes the subtask each time the common variable is changed in the parent ask.

This lesson covers various topics including:

- Writing a program that handles two data sources with a one-to-many data relationship
- Creating a subtask
- The Subform control

One-to-Many Data Relationship Preface

A one-to-many data relationship is established between a primary data source and a secondary data source in such a way that each record in the primary data source has several related records in the secondary data source.

The one-to-many data relationship will be explained using the **Orders** and **Order Lines** data sources.

Primary Data Source

In the following example, the **Orders** data source is the primary data source.

An order record usually contains basic details regarding the order, for example:

- Order Number
- Order Date
- Customer Code
- Total Amount
- Method of Payment

The **Order Number** will be the *Unique* index of the data source, since each record has a unique order number.

Secondary Data Source

The **Order Lines** data source is the secondary data source. It contains several data items:

- Order Number
- Order Line
- Product
- Quantity
- Price

The **Order Number** and the **Order Line** will be the *Unique* index of the data source. The same order number can repeat several times with different order line numbers. The linkage between the **Orders** data source and the **Order Lines** data source is the **Order Number**.

The Relationship Between the Data Sources

For each record in the **Orders** data source there are several records in the **Order Lines** data source. This is the exact definition of a **one-to-many data relationship.**

Here's an example of an order:

Orders data source with sample data

Order Number	Order Date	Customer Code	Total Amount	Method of Payment
1	20/11/2012	1111	100	Cash

Order Lines data source with sample data

Order Number	Order Line	Product	Quantity	Price
1	1	Logitech G5 Laser Mouse	3	15
1	2	Key Chain with LED Light	4	10
1	3	Infrared Thermometer Gun	1	15

The **Order Number** variable appears in both data sources; this is the common element of the two data sources that establishes the connection.

In the above example, there is one record with **Order Number = 1** in the **Orders** data source and three records in the **Order Lines** data source with **Order Number = 1**.

Advantages of the One-to-Many Data Relationship

If someone is not familiar with one-to-many data relationships, they would just create one data source for orders. Using the same example from the last page, the data source would look as follows:

Order Number	Order Date	Customer Code	Total Amount	Method of Payment	Line	Product	Qty	Price
1	20/11 /2012	1111	100	Cash	1	Logitech G5 Laser Mouse	3	15
1	20/11 /2012	1111	100	Cash	2	Key Chain with LED Light	4	10
1	20/11 /2012	1111	100	Cash	3	Infrared Thermo meter Gun	1	15

You can see that the first five columns are repeated for each line in the order.

This will cause problems, such as the end user having to repeatedly type the same information, and large records because of the repeated data. The result would be a large data source, which takes up more disk space and slows down the engine's performance.

The solution to this problem is to define two data sources according to the following rules:

- All of the repeating data columns should be defined in the **primary** data source.
- All of the unique data columns should be defined in the **secondary** data source.
- Both data sources should be connected using a **common** column.
- The **common** column is the only column that will be repeated in the **secondary** data source.

Separating data into two data sources saves the end user from having to type the same data several times and it also reduces the size of the records.

Defining the Many Data Source

In previous lessons, you added a data source by creating the table in the **Data** repository. When working with most SQL databases, the DBA creates the table in SQL. Magic xpa enables you to import the definition of that table thereby saving you from redefining the table. This is known as **Get Definition**.

You will use this to retrieve the definition of the **Order_Lines** data source.

1. Open the Data repository.
2. Create a data source named **Order_Lines**.
3. Use the same name for the **Data source name** column.
4. Select the **Getting Started** database.
5. From the **Options** menu, select **Get Definition** or press **F9**.

The table definition is now a part of the Data repository. All that is left to do is to attach the entries to the models you have already created:

6. Attach **Product_Code** to the **Code** model.
7. Make sure to inherit the **Picture**.

The **Order_Number** of the new table must be the same as the **Order_Number** in the **Order_Lines** data source. In a previous lesson, you defined this as Numeric with a size of 6. If you had defined a model, you could simply attach the model here and it would inherit all of the attributes.

Make the following changes to the column pictures:

8. Set the **Order_Number** to **6**.
9. Set the **Line_Number** to **3**.
10. Set the **Product_Price** to **6.2**.
11. Set the **Product_Quantity** to **3**.

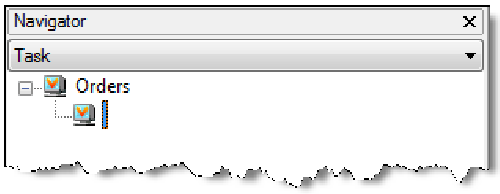
The Get Definition functionality is only valid for SQL databases.

Establishing the One-to-Many Data Relationship

When viewing an order that has the order heading and the order lines, you often want to display both the heading and the lines on the same form. When you use a tablet, there is a difference in what can be displayed when the table is in portrait or landscape mode. There is also a difference between what is displayed on a tablet and what is displayed on a smartphone. You need to know where your program will be executed and set up the program accordingly. In a later lesson you will learn how to find out if the program is in landscape or portrait mode. If your program is running on a smartphone, you may decide that to show a one:many scenario, you would add a push button in the Orders program to call another program that displays the lines for that program. This method is suitable for all of the mobile environments. In this example, you will be displaying the order and its lines on a single form. You can decide which method suits your purpose for your own application.

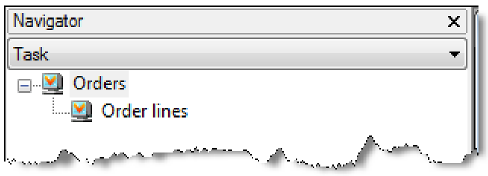
To establish the one-to-many data relationship, you will use two tasks in the same program. The first task will be a parent task and the second task will be a child task. This is known as a subtask.

1. Zoom to the **Orders** program.
2. In the Navigator pane, click the **Orders** entry.
   (If required: Select **Navigator** from the **View** menu to open the Navigator pane.) If the **Properties** pane is displayed, click the **Navigator** tab which is visible at the bottom of the pane.
3. Create a line. A new entry (task) will be created in the Task tree, as you can see in the image below.



The **Task Properties** dialog box (of the new task) opens.

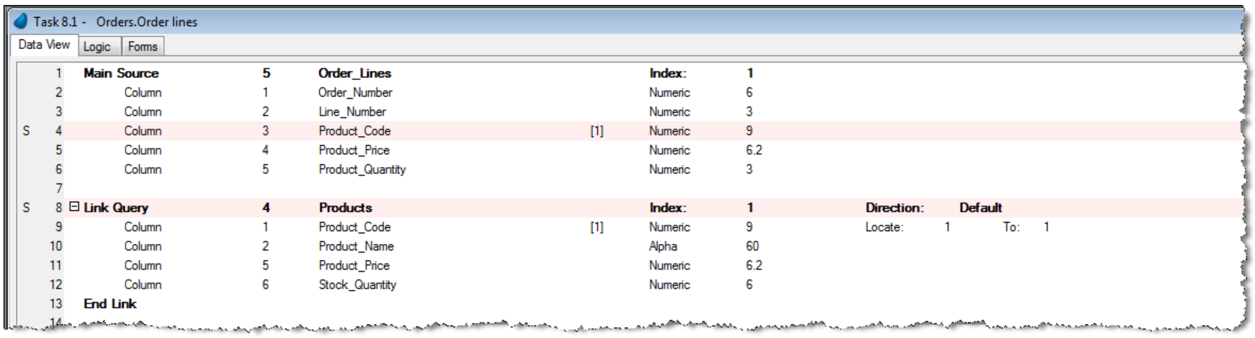
4. In the **Task name** property, type: **Order Lines**. This is automatically defined as a Rich Client task because the parent task is a Rich Client task.
5. Set the Initial mode to Query.



6. Open the Data View Editor of the **Order Lines** program.
7. Set the Main Source to **Order_Lines** and the index to **OrderLine**.
8. Add all of the **Order_Lines** columns.

The **Order_Lines** data source has a column that has a **one-to-one** data relationship with the **Products** data source. The connection is the **Product_Code** column. To increase the information about the selected product, you will add a link to the **Products** data source.

9. Create a Header line and select **Link Query**.
10. Select the **Products** data source with the **ProductCode** index.
11. Add the following columns from the **Products** data source: **Product_Name**, **Product_Price**, **Stock_Quantity**.
12. In the **Product_Code** (of the **Products** data source), define an expression for the **Locate** properties, which uses the **Product_Code** from the **Order_Lines** data source.


Task 8.1 - Orders.Order lines

Data View | Logic | Forms

	1	**Main Source**	**5**	**Order_Lines**		**Index:**	**1**			
	2	Column	1	Order_Number		Numeric	6			
	3	Column	2	Line_Number		Numeric	3			
S	4	Column	3	Product_Code	[1]	Numeric	9			
	5	Column	4	Product_Price		Numeric	6.2			
	6	Column	5	Product_Quantity		Numeric	3			
	7									
S	8	⊟ **Link Query**	**4**	**Products**		**Index:**	**1**	**Direction:**	**Default**	
	9	Column	1	Product_Code	[1]	Numeric	9	Locate: 1	To:	1
	10	Column	2	Product_Name		Alpha	60			
	11	Column	5	Product_Price		Numeric	6.2			
	12	Column	6	Stock_Quantity		Numeric	6			
	13	**End Link**								
	14									

Maintaining Data Integrity

The **Orders** data source has a one-to-many data relationship with the **Order_Lines** data source.

To ensure that the **Order_Lines** subtask will only manipulate records that are related to the parent task's order record, you need to do the following:

- Set a Range criteria according to the common variable (Order_Number). This ensures that only records relating to a certain order will be displayed.
- Initiate the subtask's common variable (Order_Number) with the parent task's common variable (Order_Number). This ensures that when a new record is added, the **Order_Number** will be added automatically.
- Prevent the end user from seeing or updating the subtask's common variable (Order_Number).

In addition, the subtask content should be displayed within the parent task's form, using a Subform control.

The Subform control enables automatic displaying and refreshing of the subtask content according to a parameter that was passed from the parent task.

To use the Subform control's automatic behavior, a parameter should be defined in the subtask.

Defining the Task Range

In the **Order_Lines** subtask's Data View Editor:

13. Create a line below the Main Source row.
14. Add a parameter: **P.Order_Number** parameter with **Attribute = Numeric** and **Picture = 6**.

Now you will add the task range:

15. Park on the **Order_Number** column from the Main Source.
16. From the **Range from** property, zoom to the Expression Editor and create an expression for the **Order_Number** parameter.
17. From the **Range to** property, select the same expression number that you set in the **Range from** property.

Initializing the Order Number Value

The **Order_Number** is the connection between the two tables. This should be added automatically to each new record in the child task. This is performed using the **Init** property.

18. From the **Order_Number** column, tab to the **Init** property.
19. Zoom to the Expression Editor.
20. Select the expression that you created for the **P.Order_Number** parameter.

Each time a new record is created in the subtask, Magic xpa will initialize the **Order_Number** in the subtask with the **Order_Number** value passed from the parent task.

Designing the Order Lines Form

The form is a similar form to the forms you have created during this course.

1. Open the Form Editor.
2. Open the **Order Lines** Form Properties.
3. Attach the form to the **Table Display Form** model.
4. Set the **Height** property to **12.750**.
5. Zoom to the **Order Lines** form.
6. Place a Table control on the left corner of the form. Use the model named **Table**.

Remember that in runtime, adding an order line to the table will be performed by calling a program. Therefore you need to leave space for the push button.

7. Drop the **Line_Number** variable on the table. Select the **Edit control** model.
8. Click on the **Line_Number** column's header area.
9. Zoom into the Column Properties and select the **Column Header** model.
10. In the **Column Title** property, type: **Line**.
11. Set the **Width** property to **7**.

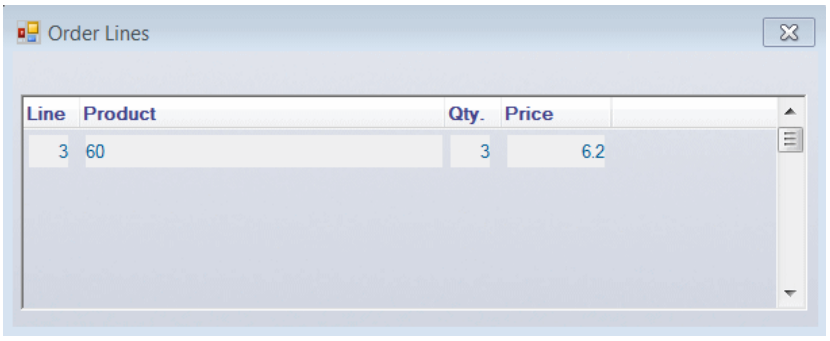
You will now add the **Product_Name** Edit control. This field is large and therefore you need to reduce the size of the field initially. The Placement properties will increase the width wherever possible.

12. Increase the size of the table so that it fills the form.
13. Drop the **Product_Name** variable on the table. Select the **Edit control** model.
14. Open the **Product_Name** Edit Control Properties.
15. Set the **Width** property to **40**. Note that the **Allow Parking** property is set to **False**.
16. Click on the **Product_Name** column.
17. Open the Column Properties and select the **Column Header** model.
18. In the **Column Title** property, type: **Product**.
19. Set the **Width** property to **40**.

In a similar manner, you will add the quantity and the price, both taken from the **Order_Lines** data source:

1. Drop the **Product_Quantity** variable on the table. Select the **Edit control** model.
2. Open the **Product_Quantity** Control Properties.
3. Click on the **Product_Quantity** column.
4. Open the Column Properties and select the **Column Header** model.
5. In the **Column Title** property, type: **Qty.**
6. Set the **Width** property to **7**.
7. Drop the **Product_Price** variable on the table. Select the **Edit control** model.
8. Open the **Product_Price** control Properties.
9. Click on the **Product_Price** column.
10. Open the Column Properties and select the **Column Header** model.
11. In the **Column Title** property, type: **Price**.
12. Set the **Width** property to **13.5.**

Your form should look similar to the image below:


Order Lines

Line	Product	Qty.	Price
3	60	3	6.2

Adding a Line Total Column

In each line, the product price and the product quantity are displayed. It makes sense then to add a **Line Total** for each line. You will now add an Edit control that will display the **Line Total**.

1. Drag an Edit control onto the table, on the header of the **Price** column.
2. Open the **Edit** Control Properties and set the model to **Edit control**.
3. Expand the **Data** property and zoom from the **Expression** line
4. Set the expression: **Product_Price*Product_Quantity**, both from the **Order_Lines** data source.
5. Set the **Format** property to **6.2**.
6. Set the **Width** property to **12**.
7. Set the Horizontal alignment property to Right.

You will now change the Table Column Properties:

8. Select the new **Edit Control** column.

9. Open the **Edit Control** Column Properties and select the **Column Header** model.
10. In the **Column Title** property, type: **Total**.
11. Set the **Width** property to **13.5**.

You have almost finished creating the program. What is left is to be able to add a row and to change a row. You will do this soon. By this stage you should know how to do this.

Subform Control

Now that the subtask creation is completed, it should be called and displayed within its parent task, the **Orders** task.

Magic xpa uses a Subform control to display, call, and refresh a subtask's content.

The Magic xpa Subform control provides an easy way to integrate another task's form within a given form.

A parent-child relationship between two tasks, which you are learning about in this lesson, is an example for the usage of the Subform control.

The task that is called from the Subform control is independent of the parent task, in terms of record cycle and logic, and is a dependent unit in terms of the parent task's values.

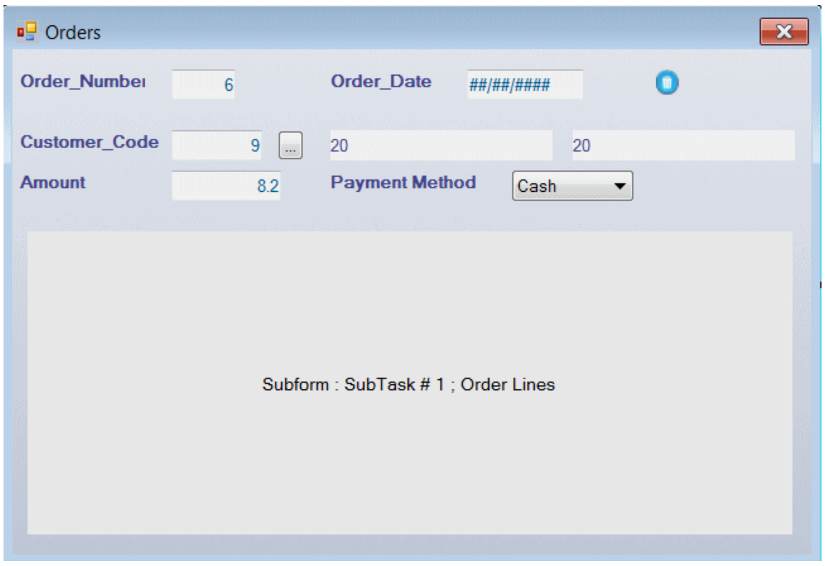
Using the Subform control has the following advantages:

- The navigation from the parent task to the called task is handled automatically by Magic xpa.
- The data view of the subform can be refreshed automatically (by Magic xpa) according to the parent task's passed arguments.
- When the end user re-enters the subform's task, Magic xpa retains the last position in the subform's data view.
- Magic xpa provides a tab cycle for the Subform control, and therefore for the calling task.
- The Subform control permits event activation from a parent task.

You will now add a Subform control to the **Orders** form.

1. From the Navigator pane, click the **Orders** task.
2. Zoom to the **Orders** form.
3. Place a Subform control ( )on the form (under the controls that are already there).
4. Open the Subform Control Properties.
5. From the **Connect to** property, select **SubTask**.
6. From the **PRG/TSK num** property, zoom to the **Subtask list**.
7. Select the **Order Lines** subtask.
8. From the **Arguments** property, zoom to the Argument repository.
9. Create a line and zoom from the **Var** column to select the **Order_Number** column.
10. You want the subform to increase when the size of the form increases so that you can see more information. Therefore, enter **100** in the **Width** and the **Height** properties: **{0,100,0,100}**.
11. Increase the width and height of the subform to fill the bottom of the **Orders** form.

You have finished creating the **Order Lines** subtask and displaying its form within the **Orders** form. This means that you have created your first one-to-many data relationship program.


Orders

Order_Numbei	6	Order_Date	##/##/####	
Customer_Code	9	20	20	
Amount	8.2	Payment Method	Cash	

Subform : SubTask # 1 ; Order Lines

In this lesson, you used a Subform control that called a subtask and a different task was executed within that subtask.

The Subform control is navigated just like any other Magic xpa control. To move to any entry in the subform, you just tap it.

If you run the application by cutting and pasting the **RunMe** public name to the **List of Orders** program and then viewing an order, you will not see any order lines, since you have not added any. You need to create the program that adds a specific line.

You can add the **Add Lines** program as a subtask of the **Order_Lines** program:

1. Zoom to the **Orders** program.
2. In the Navigator pane, click the **Order Lines** entry and press **F4** to create a new subtask.
3. The **Task Properties** dialog box (of the new task) opens. In the **Task name** property, type: **Add / Edit Line**. This is automatically defined as a Rich Client task because the parent task is a Rich Client task. This task will be called when you edit a line and when you add a line
4. In the Data View Editor, add a **Parameter** and name it **P.Initial Mode**. It will be **Alpha** with a size of **1**.
5. Set the Main Source to **Order_Lines** and the index to **Lines**.
6. Add all of the **Order_Lines** columns.
7. Zoom into the **Range from** property of the **Order_Number** column and set the range to the **Order_Number** column of the **Order_Lines** subtask that appears in the **Order Lines** task, not the current task.
   Since this is a subtask, the variables from the parent task are visible and you can use them directly. There is no need to use a parameter.
8. Set the same expression in the **Range To** property and in the **Init** expression. Remember that the **Init** expression is evaluated when the task is in Create mode.

If the subtask is called in modify mode, you also need to range according to the **Line_Number** column, but in create mode, you set the value of the **Line_Number** column:

9. Zoom into the **Range from** property of the **Line_Number** column and set the expression to:
   CndRange (U='M', N)
   where **U** is **P.Initial Mode** and **N** is **Line_Number** from the **Order Lines** subtask
10. Set the same expression in the **Range to** expression.

As you might remember, the **Order_Lines** data source has a column that has a **one-to-one** data relationship with the **Products** data source. The connection is the **Product_Code** column. You will be adding a link to the **Products** data source. As the user needs to select the product, you need to also add a Link Success variable.

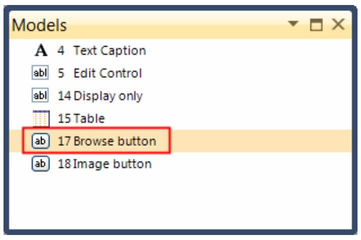
11. Add a Virtual variable with a **Logical** attribute named **Product exists**.
12. Create a Header line and select **Link Query**.
13. Select the **Products** data source with the **ProductCode** index.

14. Add the following columns from the **Products** data source: **Product_Name**, **Product_Price**, **Stock_Quantity**.
15. In the **Product_Code** (of the **Products** data source), define an expression for the **Locate** properties, which uses the **Product_Code** from the **Order_Lines** data source, of the current task. Remember that the variables from the parent task are visible.
16. Zoom into the Task Properties and set the **Initial mode** property to **By Exp**. Zoom into the **Exp.** property and set an expression:
    If (P.Initial Mode = 'C', 'C'Mode,'M'Mode)
    This means that if **C** is passed, the subtask will be in create mode and if not it will be in modify mode.
17. From the **Task** menu, select **User Events**.
18. Create an event named **Select Product** and from the **Force Exit** column, select **Editing**.
19. In the Logic Editor, create a Header line for the **User** event, **Select Product**.
20. Create a line for the **Call Program** operation.
21. Call the **Select Products** program.
22. From the **Arguments** field, zoom to the Arguments repository.
23. Create a line, zoom and select the **Product_Code** variable from the **Order_Lines** data source in the current task.

Now you need to define the form:

1. Zoom into the Form Editor and set the model to **Table Display Form**.
2. Drop the **Line_Number** and **Product_Code** variables from the **Order_Lines** data source on the form. Do not forget to use the models.

To the right of the **Product_Code** you are going to add the select button:



3. Drag the **Browse button** model from the Models pane onto the form. Magic xpa will automatically create the required control for you and attach the model to the control.
4. Zoom into the control properties and from the **Event Type** property, select **User**.
5. From the **Event** property, change the event so that it raises the **Select Products** User event.
6. Add the **Product_Name** variable to the form.
7. Add the Product_Price and the Product_Quantity.
8. You can add the product's picture as well.

Each product has a default price, which is fetched from the **Product_Price** column of the **Products** data source. When creating a new order line, the default product price should be presented to the end user. This is what is known as the list price or catalog price. In your scenario, the end user should be able to change the price of the product for the specific order. Therefore, there is a **Product_Price** column in the **Order_Lines** data source.

In the following section you will implement the above logic, using the **Variable Change** logic unit and the **Update** operation.

1. Open the Logic Editor and create a Header line.
2. Set a **Variable Change** logic unit for the **Product_Code** variable (from the **Order_Lines** data source).
3. Create the parameters by clicking **Yes** in the Confirmation dialog box. The two parameters will be created in the **Variable Change** Logic Unit section.
4. Create an **Update Variable** operation.
5. Update the **Product_Price** (from the **Order_Lines** data source) with the **Product_Price** (from the **Products** data source).

If the user changes the **Product_Code** number in the order, a new price will be fetched from the database.

The parent task that displays the order's lines needs to know that there are changes and therefore the view needs to be refreshed.

6. Create a **Record Suffix** logic unit.
7. Raise the **View Refresh** event.

Now you need to call this subtask from the parent subtask.

1. Zoom into the **Order Lines** subtask.
2. Select **User Events**.
3. Create an event named **Add order line**.
4. Create another event named **Change order line**.
5. In the Logic Editor, create a Header line for the **User** event, **Add order line**.
6. Create a line for the **Call Program** operation. Park on the **Program** combo box and select **Subtask**. Select the **Add / Edit line** subtask.
7. From the **Arguments** field, zoom to the Argument repository.
8. Create a line and create an expression: **'C'**.
9. Create a Header line for the **User** event, **Change order line**.
10. Create a line for the **Call Program** operation. Park on the **Program** combo box and select **Subtask**. Select the **Add / Edit line** subtask.
11. From the **Arguments** field, zoom to the Argument repository.
12. Create a line and create an expression: **'M'**.

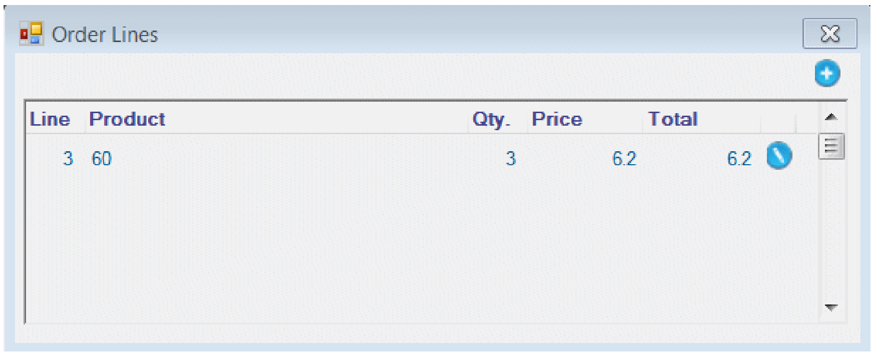
Now you need to add the push buttons to the form that will raise the events.

13. Zoom into the Form Editor and zoom into the **Order Lines** form.
14. From the Models pane, drag and drop the **Image button** model at the top of the form.
15. Zoom into the Control Properties.
16. Park on the **Image List file name** property and modify the image name to **Add.png**.
17. From the **Event Type** property, select **User**.
18. From the **Event** property, select **Add order line**.
19. Click the Fit Control Size icon.

The edit line icon can be displayed on the table:

20. Drag and drop the **Image button** model on the table, on the **Line Total** column.
21. Zoom into the Control Properties. Since the default image is the **Edit.png**, there is no need to update the image.
22. From the **Event Type** property, select **User**.
23. From the **Event** property, select **Change order line**.
24. Click the **Fit Control Size** icon.

Your form will look similar to the image below:



Remember that you run the **Orders** program from the **List of Orders** program.

1. Execute the application. The **List of Orders** program is displayed.
2. If there is an order, such as order #1, click the Edit icon. If no order exists, click the Add icon.
3. The order will open but no lines will be displayed.
4. Click the Add icon and add the following line:

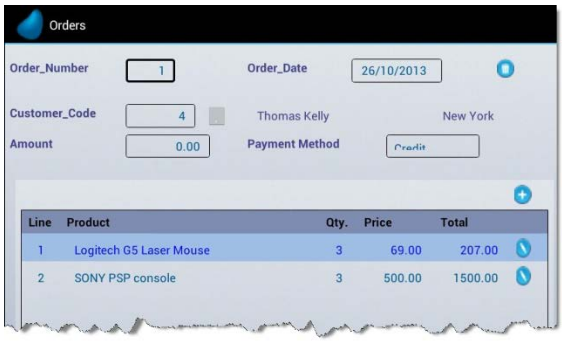
Order	Line	Product_Name	Price	Quantity	Line Total
1	1	Logitech G5 Laser Mouse	69	3	207

5. Add another line:

Order	Line	Product_Name	Price	Quantity	Line Total
1	2	Sony PSP console	500	3	1500

The Sony PSP console price is: 527.14. You have to manually change the price to 500.

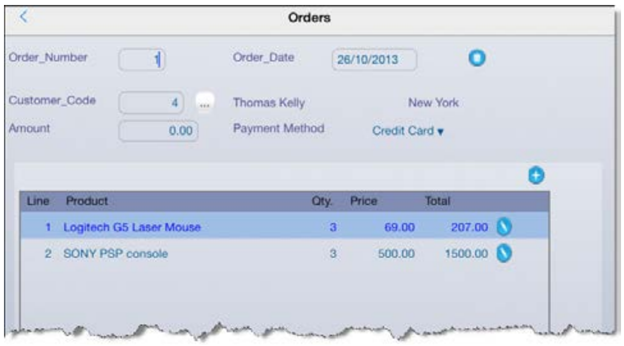
Android


Orders

Order_Number	1	Order_Date	26/10/2013
Customer_Code	4	Thomas Kelly	New York
Amount	0.00	Payment Method	Credit

Line	Product	Qty.	Price	Total
1	Logitech G5 Laser Mouse	3	69.00	207.00
2	SONY PSP console	3	500.00	1500.00

iOS


Orders

Order_Number	1	Order_Date	26/10/2013
Customer_Code	4	Thomas Kelly	New York
Amount	0.00	Payment Method	Credit Card ▾

Line	Product	Qty.	Price	Total
1	Logitech G5 Laser Mouse	3	69.00	207.00
2	SONY PSP console	3	500.00	1500.00

6. On the **Orders** form, tap on the **Amount** control.
7. Type the value: **1707.00**.
8. Select **Credit Card** as the **Method Of Payment**.

More About the Subform Control

Subform View Refresh

The subform view is refreshed whenever the end user executes its task, such as clicking the Subform area or tabbing into it.

The subform view is also refreshed automatically by Magic xpa. This occurs when the value of one of the passed parameters is changed and the **Automatic Refresh** property is set to **Yes**.

Note: This is why you sent the **Order_Number** as a parameter to the subtask. In this way, whenever the end user navigates between the orders (even without entering the Subform area), the Subform view is automatically refreshed.

Magic xpa also supplies an internal **Subform Refresh** event.

Subform Task Execution

The Subform task is executed:

- For the first time, after the first Record Prefix of the parent task. (This mode of execution is referred to as: executing the first Record Prefix.)
- When the end user tabs into the Subform control.
- When the end user clicks the Subform area (either when clicking on a specific control or any location in the Subform area).

You can control when a subform is loaded with the **Refresh When Hidden** property. This property is useful when the subform is not initially visible, which may be when it is a part of a Tab control or when there is an expression on its Visible property.

Subform Task Termination

The Subform task is terminated when:

- The outside of the Subform area is clicked.
- The **Close** (internal) event is raised.
- The **End Task** condition of the Subform task evaluates to **True**.
- The parent task is closed.

When the Subform task is terminated, the following occurs:

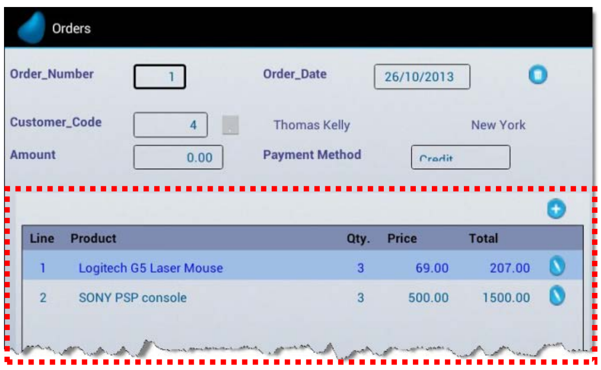
- The **Control Suffix** logic unit (of the parked control in the subform) is executed.
- The **Record Suffix** logic unit (of the connected task) is executed.
- The **Task Suffix** logic unit (of the connected task) is executed.
- The focus returns to the next control in the parent task as determined by the parent task's **tabbing order** and **tabbing direction**.

Subtask Form Transparency

Look at the **Orders** program while it is executing.

Can you see that there are actually two wallpapers: one for the **Orders** task and one for the **Order_Lines** subtask?

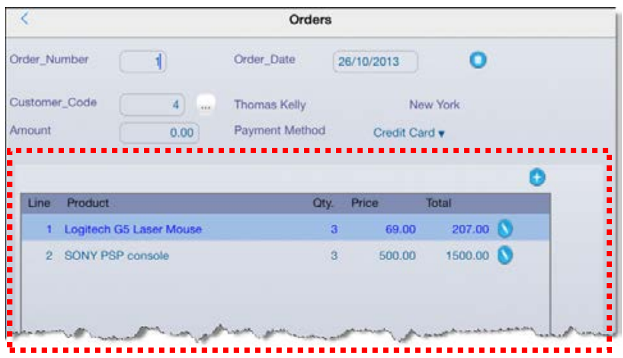
Android


Orders

Order_Number	1	Order_Date	26/10/2013
Customer_Code	4	Thomas Kelly	New York
Amount	0.00	Payment Method	Credit

Line	Product	Qty.	Price	Total
1	Logitech G5 Laser Mouse	3	69.00	207.00
2	SONY PSP console	3	500.00	1500.00

iOS


Orders

Order_Number	1	Order_Date	26/10/2013
Customer_Code	4	Thomas Kelly	New York
Amount	0.00	Payment Method	Credit Card

Line	Product	Qty.	Price	Total
1	Logitech G5 Laser Mouse	3	69.00	207.00
2	SONY PSP console	3	500.00	1500.00

If you would have used plain wallpaper you would not be able to tell the difference; but since you are using wallpaper, the difference is observable. To overcome this, you need to set the Subtask form with a transparent color and remove the Subtask form's wallpaper.

1. Zoom to the **Orders** program and select the **Order_Lines** task.
2. In the Form Editor, park on the **Order_Lines** form.
3. Clear the **Wallpaper** property.
4. In the **Color** property, select the color: **Text Caption**.

Incremental Update

While creating the first order you were asked to manually fill in the **Amount** value.

As you may have already figured out, this is not the right way to calculate the order's amount.

Magic xpa has a specific mechanism that performs such calculations; this mechanism is called **Incremental Update**.

This section will introduce you to Magic xpa's incremental updating.

Magic xpa enables you to incrementally update a variable, meaning that the update expression value is evaluated and added or subtracted from the updated value according to the following rules:

- If a new record is added, the value is added to the updated value.
- If a record is deleted, the value is subtracted from the updated value.
- If a record is modified, the old value is subtracted and the new value is added to the updated value.

You will now automatically calculate the order's **Amount** and disable the ability to park on the **Amount** control, so that the end user will not be able to manually change its value. For each order line, you have a total for the line that is based on a calculation of the product price and the quantity. You used an Edit control containing an expression. In this section, you will update the **Amount** variable with the same expression. You are going to update the **Amount** variable when the **Product_Price * Product_Quantity** value changes. To implement this, you will add a **Line Total** variable and use that on the Form Editor instead of the Edit control. You will update the **Amount** variable from the **Record Suffix** logic unit.

To add the **Line Total** variable:

1. Zoom to the **Orders** program and zoom to the **Order_Lines** task.
2. Open the Data View Editor.
3. Add a line, preferably as the last line in the editor.
4. Create a Virtual variable and name it: **Line Total**.
5. Set the **Attribute** to **Numeric** and the **Picture** to **6.2**.

The **Line_Total** Edit control that you placed on the form was a calculation of the product price and the quantity. You will use Magic xpa's internal recompute mechanism by using this expression in the **Init** property of the Virtual variable that you added.

6. Park on the **Line Total** variable and tab to the **Init** property.
7. Zoom to the Expression Editor.
8. Select the expression **Product_Price * Product_Quantity** that you previously created.

The **Line Total** variable is automatically calculated every time a change is made to either the **Product_Price** variable or the **Product_Quantity** variable. Since the value is automatically calculated, there is no need to park on the variable. You need to make the variable **non-parkable** on the form. This will be the same behavior as using an Edit control. There are two methods of doing this:

- Placing the variable on the table and making the control non-parkable. You have used this method throughout this course.
- Defining the non-parkable property at the level of the variable.

For this example, you will use the second method:

1. Park on the **Line Total** variable.
2. Open the variable property sheet.
3. In the **Style** node, park on the **Rich Client table** property. This is the property that governs the look and feel of this variable when it is dropped on a table.
4. Click the Zoom button. This opens a control property sheet.
5. In the **Parking** section, set the **Allow Parking** property to **No**.
6. Set the **Model** to **Edit Control**.

Adding the Line Total variable to the form:

1. Zoom to the **Order_Lines** form.
2. Park on the **Total** Edit control and delete it.
3. From the Task Variables pane, drag the **Line Total** variable and drop it on the table on the **Price** column.
4. Open the control property sheet. Check that the **Allow Parking** property is set to **No**.
5. Open the **Column Control Properties**.
6. Set the model to **Column Header**.
7. Set the **Column title** property to **Total**.

To update the **Amount** variable you need to perform this where you update the lines and this is in the **Add / Edit Line** subtask.

1. Zoom into the **Add / Edit Line** subtask.
2. Open the Logic Editor.
3. Zoom into the **Record Suffix** logic unit.
4. Before the **Raise Event** of the **View Refresh** event, create a line for the **Update** operation.
5. Update the **Amount** variable (from the **Orders** data source) with the **Product_Price * Product_Quantity** variables, both from **Order_Lines** data source.
6. From the **Update** operation's properties (**Alt+Enter**), set the **Incremental** property to **Yes**.

Now when the **Amount** variable is automatically calculated by Magic xpa, the end user should not be able to manually change its value. Therefore, you will now set the **Amount** control's **Allow Parking** property to **False**.

1. In the Navigator pane, click the **Orders** task.
2. Open the Form Editor and zoom to the **Orders** form.
3. Select the **Amount** Edit control.
4. From the **Amount** Control Properties, set the **Allow Parking** property to **False**.

You can now execute the application.

1. Edit the first order, order number 1.
2. Tap the **Add** button to add an order line.
3. In the **Line** column, type: **3**.
4. Zoom from the **Product Code** column and select: **HealthCare Foldable Full Body Massage Lounger (product number 30)**.
5. In the **Quantity** column, type: **1**

Close the task and you will be returned to the list of order lines. The new line is there. Check the **Amount** value; it should display: **1906.95**. This is the sum of the existing **Amount** value plus the value of line number 3 (**199.95**).

1. Click the edit button on order line number 3.
2. Change the **Product Price** from **199.95** to **340**.
3. Close the task and check the **Amount** value; it should display: **2047.00**. This is the sum of the existing **Amount** value plus the difference in line 3 (340 - 199.95 = **140.05**).

Exercise

Countries and **cities** have a one-to-many data relationship. Each **country** can have many cities.

Now you will practice what you learned so far, by creating a program that displays a list of countries and their related cities.

1. Create a model named **Country&City Code** with a **Numeric** attribute and a **Picture** of **6**. Set the **Color** to **Edit control**.
2. Create **Getting Started** data sources for **Countries** and **Cities** and import the definition using the **Get Definition** utility.
3. Attach the **Country&City Code** model to the **Country_Code** and **City_Code** columns and re-inherit the **Picture** property.
4. Define a **Countries** program that displays the country and its cities.

Summary

This lesson introduced you to the **one-to-many data relationship**.

You learned how a one-to-many data relationship can help save space and reduce maintenance, when compared to having all of the data in one data source.

You practiced creating a parent task and child task in order to implement a one-to-many data relationship using Magic xpa.

The example included:

- Defining a parent task and a subtask.
- Using the Range and Init properties to prevent data integrity violation.
- Using a Subform control to display the subtask's form within the parent task's form.
- Determining the refresh trigger by passing an argument to the child task in the Subform control.

Lesson 15

Non-Interactive Processing

Throughout this course you learned about Rich Client programs. Until now, the assumption when developing a program was that the programs ran on the same machine, but this is not a real life scenario. There is often a client machine and a server machine. The server holds the main part of the application installation, including images, files and the database. The client is a thin client and provides the front-end interface.

In every interactive program, there is a part of the program that executes on the server and a part that executes on the client. There is constant traffic of information along the network between the client and the server. The communication between the two is done by sending a compressed XML file. It is this traffic that may prevent a smooth user experience by creating a bottleneck. It is good practice to try and minimize this traffic.

In the next few pages you will learn about client interaction with the server. Then, you will learn about the Magic xpa Batch task.

This lesson covers various topics including:

- How the Rich Client interacts with the server
- How the Magic xpa engine works in Batch tasks
- The differences between Rich Client tasks and Batch tasks
- When to use Batch tasks
- How the Magic xpa engine processes Batch tasks
- Maintaining data integrity using a Batch task

Data View Editor and Rich Client

The interaction between the client and the server is via XML files. The data view is assembled on the server. The actual records and fields that are used are included in the XML that is sent to the client.

Variables

Variables are usually neutral. However, for some variables, every time the variable changes on the client, server access is immediately required. Such variables are called server-side variables. These variables have an "S" next to them, and are shown in a contrasting color. You have seen this while developing programs during this course.

This can happen if the variables are used in the Range/Locate of a link. Each change of the variable value will execute the link again; hence there will be access to the server.

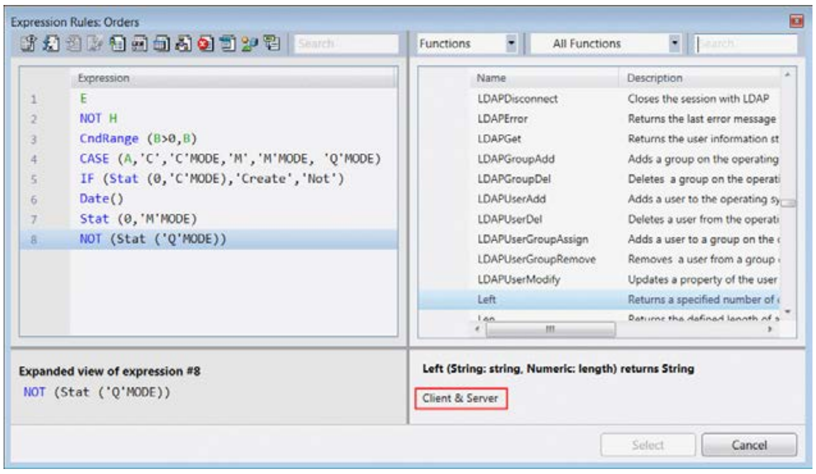
BLOBs

In a Rich Client program, if the data view contains BLOB variables with lots of content, the BLOBs will be passed between the server and clients (like other variables), and this might slow down the program. It is advised not to define the BLOBs in the data view if they are not used, or you should create a new Batch task that handles them if they are to be used on the server side.

Functions

Functions are the smallest building-block of an expression, and the side of the expression will depend on the functions (and variables) in it. Each Magic xpa function is client-side, server-side, or neutral. You can find a list of these in the Magic xpa Help.

When you select a function, you can see its execution side in the Function List. It will say Server, Client, or Client & Server (meaning neutral).



	Expression
1	E
2	NOT H
3	CndRange (B>0,B)
4	CASE (A,'C','C'MODE,'M','M'MODE, 'Q'MODE)
5	IF (Stat (0,'C'MODE),'Create','Not')
6	Date()
7	Stat (0,'M'MODE)
8	NOT (Stat ('Q'MODE))

Name	Description
LDAPDisconnect	Closes the session with LDAP
LDAPError	Returns the last error message
LDAPGet	Returns the user information st
LDAPGroupAdd	Adds a group on the operating
LDAPGroupDel	Deletes a group on the operati
LDAPUserAdd	Adds a user to the operating sy
LDAPUserDel	Deletes a user from the operati
LDAPUserGroupAssign	Adds a user to a group on the
LDAPUserGroupRemove	Removes a user from a group
LDAPUserModify	Updates a property of the user
Left	Returns a specified number of
Len	Returns the defined length of a

Expanded view of expression #8

```
NOT (Stat ('Q'MODE))
```

Left (String: string, Numeric: length) returns String

Client & Server

Expressions

Expressions are built from one or more functions and variables. The side of the expression is determined by the side of the functions and variables.

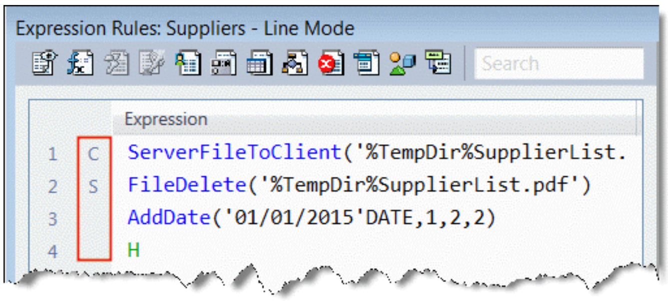
Every expression can be categorized as being Client-side, Server-side, Neutral, or Mixed.

- **Client-side** expressions can only execute on the client.
- **Server-side** expressions can only execute on the server.
- **Neutral** expressions, however, can execute in either place. They will execute on the server if the operation is being executed on the server or on the client if the operation is being executed on the client.
- **Mixed** expressions contain elements from the server and from the client.

It is important to know what kind of expression is being used. Expressions are used in many different places within a task. If the expression is a **server-side** expression, but is being used in a place that would require it to be evaluated on the **client**, the client has to stop and send a request back to the server before processing can continue. In some cases, the syntax checker will disallow the process; in other cases, it is allowed, but it will be slow.

Every expression is marked as to where it will be executed:

- S = Server-side
- C = Client-side
- M = Mixed
- blank = Neutral


Expression Rules: Suppliers - Line Mode

```
  Expression
C ServerFileToClient('%TempDir%SupplierList.
S FileDelete('%TempDir%SupplierList.pdf')
  AddDate('01/01/2015'DATE,1,2,2)
  H
```

This is important to know, because the expression will be used in an operation, a property, or an Init value. You want to make sure that the side of the expression matches where it is used.

Range and Locate Expressions

Range and Locate expressions are sometimes evaluated on the client, and at other times on the server. Therefore, the expressions used in Range and Locate need to be neutral.

Init Property

Client-side expressions cannot be used in the **Init** property. If you need to use client-side expressions in an **Init** property, use an **Update Variable** operation in the **Record Prefix** logic unit.

Client-Side vs. Server-Side Operations

As was discussed before, there is a part of the program that executes on the server and a part that executes on the client. Some of the internal Magic xpa operations are client-side operations and some are server-side operations.

Operation	Where executed
Verify	Client
Call	• Mixed: A Call to a Rich Client task using the Destination property. There is immediate access to the client to close the task that currently runs in that subform. • Server – Other Calls (to Rich Client tasks or Batch tasks) are considered as Server, since there is access to the server to initiate the task that has been called. After the focus returns to the client, the task is drawn there. You will learn more about Batch programming later in this lesson.
Update	This depends on the variable being updated and the expression used for the update.
Invoke	• Server – UDP, Web S, Web S Lite • Client and Server – OS Cmd (depends on the Execute on property) • COM – Not supported for Rich Client tasks
Raise Event	The Raise event execution side is determined by the handler's execution side of that event.
Evaluate	This can be executed either on the client or the server depending on the operation.
Block	This can be executed either on the client or the server depending on the Block expression.
Form	Not supported for Rich Client tasks. You can run reports on the server and display the results on the client.
Link Query	Server. It is recommended that you limit the use of Link operations because they are costly in terms of server interaction. Many times, you can use a Data control or Inner Join instead.

Operations are also affected by the expressions used in their properties, such as the Condition property.

The Task Life Cycle

You learned about the application engine in a previous lesson. You were also introduced to the program life cycle and the various internal logic units.

When developing a **Rich Client** program, some of these logic units are executed on the client and some on the server:

Operation	Where executed
Task Prefix	Server. No client-side operations or functions can be used here.
Task Suffix	This can be client or server, depending on the operations within.
Record	This can be client or server. It is recommended not to use Server-side functions and operations here in order to diminish accessing the server when browsing through the record.
Control	This can be client or server. It is recommended not to use Server-side functions and operations here in order to diminish accessing the server when tabbing through the controls.
Variable Change	This can be client or server. It is recommended not to use Server-side functions and operations here in order to diminish accessing the server during the recompute mechanism or when changing a control's value.
Error	Server.

These are recommendations; however, it is not always possible to refrain from accessing the server to perform the task logic.
It is good practice to group client operations and server operations together to decrease network activity.

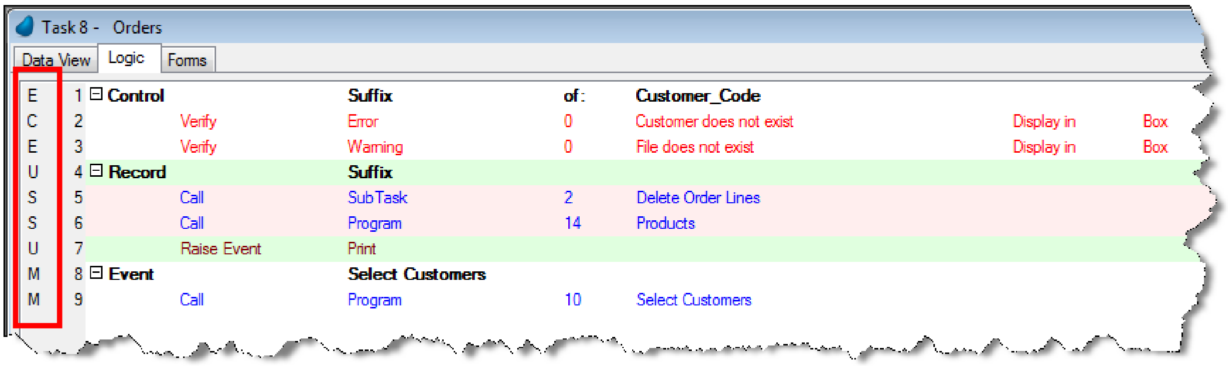
Identifying Client and Server Activity

As some of the operations are client and some are server, when you develop a program, it is important to know where the program may have a performance problem due to running back and forth between the client and the server.

Magic xpa provides a visual aide and displays the handling mode of each handler, operation, and function. A character appears on the left side of the line number column representing the handling modes below:

- C – Client-side handling: The operation will be performed on the client.
- S – Server-side handling: The operation will be performed on the server.
- M – Mixed mode handling: The operation will be performed on both the client and the server.
- U – Unknown: This is for operations that the development Studio does not know where they will be evaluated. For example, when you use the Raise Event operation but the handler is not found.
- E – Error: The operation is not supported, such as the Form Output operation.
- If a character does not appear, the entry can be executed on either the client or the server.

This is shown in the following image:



Task 8 - Orders

Data View | Logic | Forms

E	1	⊟ **Control**		**Suffix**	**of:**	**Customer_Code**		
C	2		Verify	Error	0	Customer does not exist	Display in	Box
E	3		Verify	Warning	0	File does not exist	Display in	Box
U	4	⊟ **Record**		**Suffix**				
S	5		Call	SubTask	2	Delete Order Lines		
S	6		Call	Program	14	Products		
U	7		Raise Event	Print				
M	8	⊟ **Event**		**Select Customers**				
M	9		Call	Program	10	Select Customers		

Rich Client Operation Colors

The background color of the operations in the Task Editor indicates whether the operation is server-side, client-side, neutral, or unknown. The colors are provided by Magic xpa but you can change them in the Studio Color repository.

The table below describes the various colors:

Color Name	Description
White	• Client-side operations that must be executed on the client side • Handlers' headers that contain client-side operations • Neutral operations that appear after a client-side operation • Unsupported operations, such as mixed operations
Server 🟪	• Server-side operations that must be executed on the server side • Handlers' headers that contain server-side operations • Neutral operations that appear after a server-side operation
Unknown 🟩	• Unknown-side operations. These are operations which the Studio cannot know if they will be executed on the server side or client side.

Neutral lines will have the same colors as the line above them, since they will be executed on whichever side is currently doing the processing. The flow itself can vary, since some of the operations may have a condition that will evaluate to FALSE, so they will not be processed.

For more information about working with Rich Client applications, see the **Developing Rich Client Applications** concept paper in the *Magic xpa Help*.

Batch Programming

Up until now you learned about Rich Client programs. Sometimes it is necessary to have a program that performs a process without user interaction. In Magic xpa, these types of programs are called Batch programs.

In this and the next lessons you will be introduced to the concept and behavior of Batch programs. You will then be shown how to create a program that performs a process without user interaction.

Batch tasks are used to perform a process on a set of records from a data source. They can also perform processes with no data source. Batch tasks are used for processes, such as reports, record updates or deletion, and calculations.

Although a Batch task has a form, most Batch tasks are executed without displaying anything to the end user.

Rich Client Task vs. Batch Task

Rich Client Task	Batch Task
Enables end-user interaction.	Does not allow end-user interaction.
Enables the end user to navigate through the Main Source records. Only the records that are scrolled are scanned.	The task scans all of the Main Source records within the range criteria. If no Main Source is defined, or the task is in Create mode, the task loops until the End Task condition is met.
The Control logic unit is available. The engine handles logic related to end-user navigation.	The Control logic unit is not available, since there is no end-user interaction and the engine does not park on controls.
The Group logic unit is not available.	The Group logic unit is available and enables you to handle groups of data. You will learn about this logic unit in a later lesson.
The Task Form is an essential part of the task creation and execution.	The Task Form is optional and most of the time is not in use.
If the data is changed, the Record Suffix logic unit is executed and the data is saved.	The Record Suffix logic unit is always executed. A record is always saved to the database, even if it was not changed.
Events are handled when the task is idle.	Events are handled every set period of time or every number of processed records.

Engine Flow for a Batch Task

The engine execution rules work differently in Batch tasks than in Interactive tasks.

Interactive task engine cycle	Batch task engine cycle

Task Prefix
    Record Prefix
        Control Prefix
        Control Verification
        Control Suffix
        Variable Change
    Record Suffix
Task Suffix
	Task Prefix
	    Group Prefix
	        Record Prefix
	        Record Suffix
	    Group Suffix
	Task Suffix

As you can see, in the Batch task there are no Control-related logic units, but there are logic units that allow you to handle groups of records.

Note: You will learn about the Group Prefix and Group Suffix logic units in a later lesson.

Batch Task's Life Cycle

This section explains the processes that are executed during a Batch task's life cycle.

Task Initialization

The Task Initialization operations are similar to the operations in the Online task.

Group Processing

- The Group logic unit creates breaks during the data view processing. You can use the Group logic unit to calculate sums at certain levels, to display data by groups, or to print breaks in a report. You will learn more about this logic unit in a later lesson.
- The Group Prefix logic unit operations are executed for the first task cycle and whenever the Group logic unit's variable value is different from the previous record value.

Record Processing

- The Record Prefix logic unit operations are executed.
- The Record Suffix logic unit operations are executed.
- The record is saved with the new values. If the Task mode is set to **Delete**, the record is deleted.

End Group Processing

When a new record is fetched and the Group logic unit's variable value is different from the previous record value, the Group Suffix logic unit operations are executed for the previous record and the Group Prefix logic unit operations are executed for the current record.

Batch Task Termination

The Task Termination operations are similar to the operations in a Rich Client task.

Unlike in Rich Client tasks, where the end user terminates the task, in Batch tasks the termination occurs according to the following properties or actions.

The End Task Condition Property

This property enables the developer to determine the task termination. The options are Yes, No, and setting an expression.

The Evaluate Condition Property

This property determines when Magic xpa evaluates the End Task condition. The options are:

- **Before Entering Record** – The End Task condition is evaluated before entering the record.
- **After Updating Record** – The End Task condition is evaluated after the record is updated.
- **Immediately when condition is changed** – The End Task condition is evaluated immediately when the condition is changed (without completing the current logic unit execution).

Batch Task Behavior

In this section you will learn about different characteristics that determine the Batch task behavior.

Batch Task with a Main Source

A Batch task loops through the Main Source records (within a Range criteria). The task repeats the Record Prefix and Suffix logic unit operations for each scanned record.

The task execution is terminated if one of the following conditions is met:

- The End Task condition (in the Task Properties) evaluates to True (as explained on the previous page).
- The engine reaches the last record in the range of records.

Batch Task with No Main Source

If no Main Source is defined, the Batch task is executed as long as the **End Task** condition is not met.

This type of task is called a **Zero Main Source Batch task**.

For example, if you create a program that cycles through Virtual variables, the record loop will be endless, until the End Task condition is met.

Batch Task with a Main Source and in Create Mode

When defining a Batch task with a **Main Source** and the **Initial Mode** property is set to **Create** mode, the task behaves like a **Zero Main Source Batch task**.

For example, if you create a program that only adds records to the data source, the record loop will be endless (no range can apply here), until the End Task condition is met.

Batch Task Usages

The Batch task has many usages. In many cases there is a clear connection between the Batch task mode and its usage.

The following will give you a general description about the Batch task usages depending on the Task mode.

Task Mode	Usages
Query	Used to scan Main Sources and produce outputs, such as reports, or calculation outcomes, such as number of records, maximum value and minimum value.
Create	Used to import data from an I/O source or to copy data from one or more data sources.
Modify	Used to perform sequential updates in a data source or make collective calculations for records in the data source.
Delete	Used to perform sequential deletions in a data source according to a certain scenario, such as maintaining data integrity.

Batch Delete

Batch tasks with an initial mode set to **Delete** can be used to maintain data referential integrity.

In the previous lesson you learned about one-to-many data relationships. You practiced this concept by creating the **Orders** program.

The end user creates an order by creating one record in the primary data source (**Orders**) and several records in the secondary data source (**Order_Lines**).

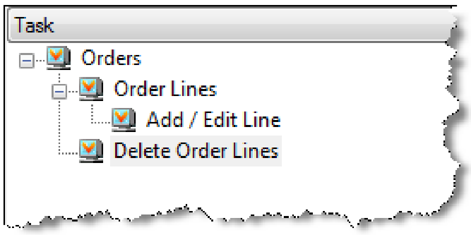
When deleting the order record from the primary data source, the related order lines' records in the secondary data source are left orphaned.

This will happen unless you handle this by using a Batch task that deletes all of the related order lines' records from the secondary data source whenever the related record is deleted from the primary data source.

On the following pages you will improve the **Orders** program by adding a Batch task to handle orphaned records.

In a previous lesson you added a push button to the **Orders** program that will enable the end user to delete an order. But what if that order also had lines? They also need to be deleted. In this example you will add a subtask to the **Orders** task that will be called every time the end user deletes a record from the **Orders** task. The called subtask will be a Batch task that deletes all of the **Order_Lines** records that have the same **Order_Number** as the **Orders** task's **Order_Number**.

1. Zoom into the **Orders** program.
2. In the Navigator pane's Task tree, park on the **Orders** node and create a line.



In the **Task Properties** dialog box:

3. In the **Task name** entry, type: **Delete Order Lines**.
4. From the **Task type** entry, select **Batch**.
5. From the **Initial mode** entry, select **Delete**.

In the Data View Editor:

6. Set the **Main Source** with the **Order_Lines** data source.
7. Set the Main Source index with the **OrderLines** index.
8. Create a line and select the first **Order_Lines** column: **Order_Number**.

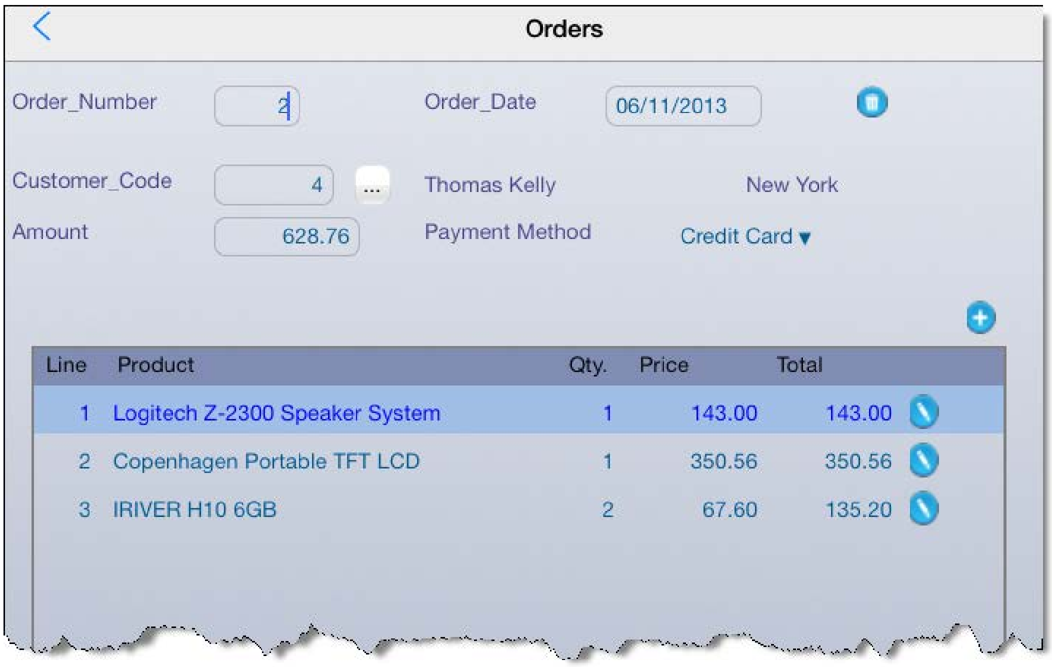
The subtask should delete only the lines that have the same **Order_Number** as in the parent task. Therefore, you need to set a range on the **Order_Number** column.

9. Park on the **Range** property.
10. Create an expression for the **Order_Number** from the parent task's **Orders** data source.
11. Set the same expression number in the **Range To** property.

The **Delete Order Lines subtask** should be called from the **Orders** program whenever the end user deletes an order. This is from the **Delete Line** logic unit.

12. Click on the Navigator pane and in the Task tree, park on the **Orders** node.
13. Open the Logic Editor and zoom into the **Delete Line** logic unit.
14. Add a detail line immediately after the Header line.
15. Call the **Delete Order Lines** subtask.

16. Execute the application and display the **List of Orders** program.
17. Tap the **Add order** button and create a new order like the image below (iOS display).
18. Create a new order with order lines as shown in the image below.


Orders

Order_Number	2	Order_Date	06/11/2013
Customer_Code	4	Thomas Kelly	New York
Amount	628.76	Payment Method	Credit Card ▾

Line	Product	Qty.	Price	Total
1	Logitech Z-2300 Speaker System	1	143.00	143.00
2	Copenhagen Portable TFT LCD	1	350.56	350.56
3	IRIVER H10 6GB	2	67.60	135.20

19. Return to the list of orders and you will see the new order.
20. Tap the **Edit order** button.
21. When the order loads, tap the **Delete** button.
22. In the **Confirmation** dialog box, click **Yes**.

Although you do not see the Batch task execution, Magic xpa executed the Batch task that deletes the related order lines.

Summary

This lesson introduced you to the Rich Client engine's client/server activity. You learned which parts of the task were executed on the client and which were executed on the server. You also learned how to identify problematic operations during development.

You then went on to learn about Batch tasks, which are server-side tasks that are executed without any user interaction. You learned about the main differences between Rich Client tasks and Batch tasks. You also learned about the Batch task's engine cycle compared to the Rich Client task's engine cycle.

You then learned about the Batch task's engine flow. You learned that Batch tasks behave differently when there is a Main Source and when there is no Main Source.

In addition, you learned that Batch tasks are used mainly for:

- Creating reports
- Processing a group of records
- Processing sequential updates
- Calculations
- Copying records from one data source to another
- Importing data from an external file

You created a Batch subtask that deletes order lines; this maintains data integrity in the **Orders** program.

Lesson 16

Reports

Creating reports is an essential part of the application's functionality. A report is a display of the application data.

You will use Magic xpa's **Program Generator** utility to generate a simple report of a data source.

You will then learn how to manually create a simple report.

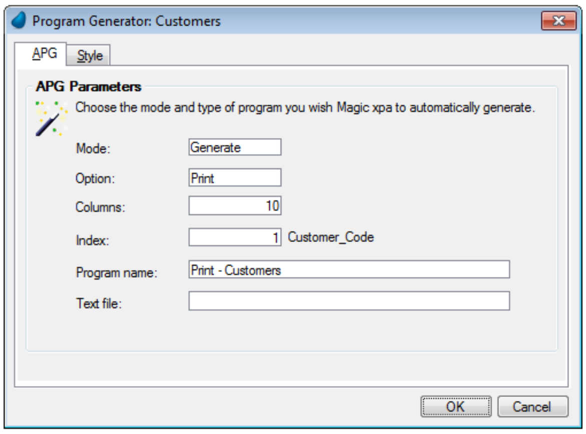
This lesson covers various topics including:

- Using the Program Generator to create a simple report.
- Manually creating a simple report.
- Designing a report that includes headers and footers.
- Creating the report as a PDF.
- Displaying the PDF using the Browser control.
- Displaying the PDF using third party applications.
- Printing the PDF from a mobile device.

Using the Program Generator Utility

In this section you will create a program that produces a report of the **Customers** data source.

1. Open the Data repository.
2. Park on the **Customers** data source.
3. From the **Options** menu, select **Generate Program**.
4. The **Program Generator** dialog box opens.
5. In the **Mode** field, select **Generate**.
6. In the **Option** field, select **Print**.
7. Park on the **Columns** field.
8. Zoom to the **Column Selection** window.

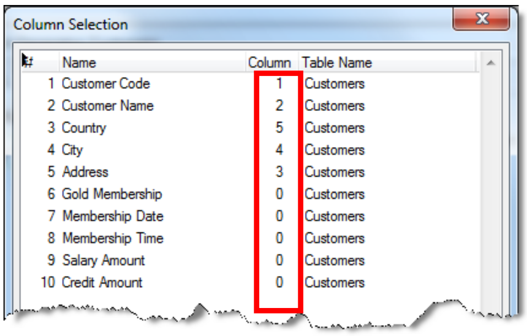


APG Parameters
Choose the mode and type of program you wish Magic xpa to automatically generate.

Mode:	Generate
Option:	Print
Columns:	10
Index:	1 Customer_Code
Program name:	Print - Customers
Text file:	

The value in the **Column** column shows the position of the column on the form.

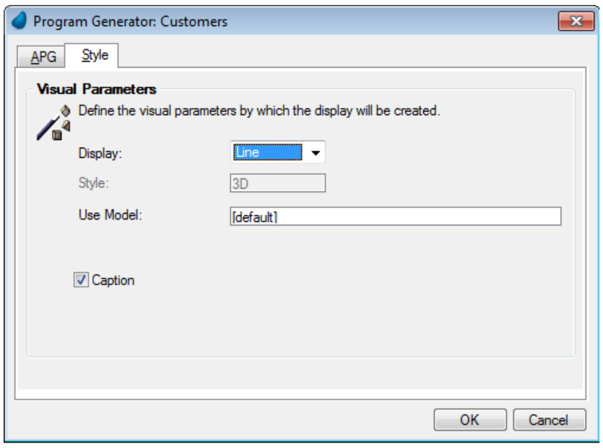
9. For the following columns, set the **Column** column with the value zero (**0**): **Gold_Membership**, **Membership_Date**, **Membership_Time**, **Salary_Amount**, and **Credit_Amount**.
10. Define the order so that the report will display the address first, then the city and then the country.
11. Click **OK**.



	Name	Column	Table Name
1	Customer Code	1	Customers
2	Customer Name	2	Customers
3	Country	5	Customers
4	City	4	Customers
5	Address	3	Customers
6	Gold Membership	0	Customers
7	Membership Date	0	Customers
8	Membership Time	0	Customers
9	Salary Amount	0	Customers
10	Credit Amount	0	Customers

As was mentioned in an earlier lesson, the **Column Selection** window allows you to determine which variables will be displayed in the report and the order of the displayed variables. A value of zero in the **Column** column of a variable indicates that it will not be displayed.

12. Click the **Style** tab.
13. Select the **Caption** check box.


Program Generator: Customers

APG | Style

Visual Parameters

Define the visual parameters by which the display will be created.

Display: Line

Style: 3D

Use Model: [default]

☑ Caption

The **Print - Customers** program was added to the end of the Program repository.

You can zoom into the program and view it. Next you will learn how to create the same program manually.

Manually Creating a Report

You have just learned about automatically generating a program that prints the **Customers** data source content. You will now create the same program manually.

1. Create a program and in the **Name** column, type: **Print - Customers 2**.
2. Zoom into the program.

The **Task Properties** dialog box opens.

3. Set the **Task type** to **Batch**.
4. Set the **Initial mode** to **Query**.

Now you're going to set the task's data view.

5. In the Data View Editor, set the **Main Source** to **Customers**.
6. Set the **Index** to **Customer_Code**.
7. Add the first 5 **Customers** columns from the data source to the Data View Editor.

Defining I/O Devices

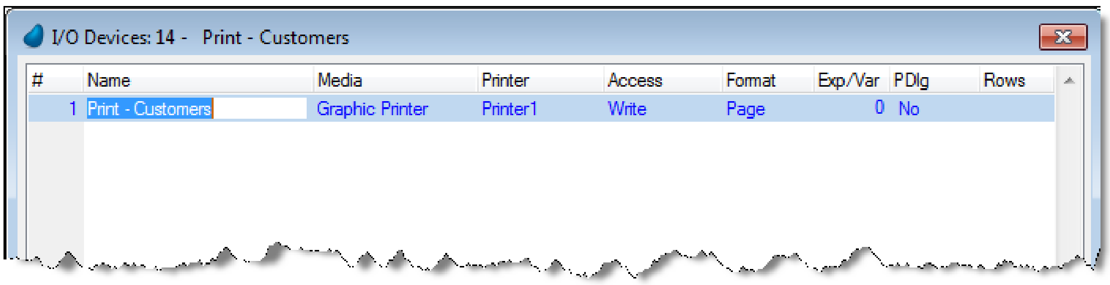
When running a report on a desktop computer, the output is printed on a printer connected to the computer. When you are using a mobile device, this is not feasible. The report needs to be displayed on the client machine. Although there are many cases in which you might want to print reports on the server machine, in this course you will be printing on the client. The suggested method for printing to a client machine is to create a PDF and to display it.

To setup an output for a program, you need to define an I/O device. This enables you to define an output that:

- Imports data from external files.
- Outputs data to a graphic printer, files, PDF, and so on.
- Sends HTML files to the internet requester.
- Accesses XML files.

You will define an entry in the I/O Device repository that will enable you to print the report to a PDF through the Graphic Printer properties.

1. From the **Task** menu, select **I/O Devices** (Ctrl+I).
2. In the I/O Device repository, create a line.
3. Set the name to **Print - Customers**.


I/O Devices: 14 - Print - Customers

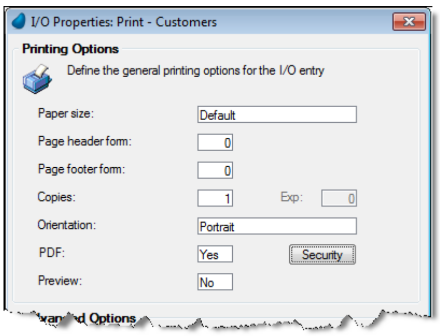
#	Name	Media	Printer	Access	Format	Exp/Var	PDlg	Rows
1	Print - Customers	Graphic Printer	Printer1	Write	Page	0	No	

By default, the generated program is designed to print the output to a printer, the default printer of the operating system. To direct the program output to a PDF:

4. Park on the **Exp/Var** property and zoom to the Expression Editor.

You need to provide a name for the file that will be created. You can provide a simple name such as **c:\Temp\CustomerList.pdf** . The PDF is created on the server. You can use the temporary folder to save the files or any other folder. Magic xpa has a logical name, **%TempDir%**, which returns the temporary folder.

5. Create an expression for the file name: **'%TempDir%CustomerList.pdf'**
6. Open the **Print - Customers** I/O Properties (**Alt+Enter**).
7. Set the **PDF** property to **Yes**.


I/O Properties: Print - Customers

Printing Options

Define the general printing options for the I/O entry

Paper size: Default

Page header form: 0

Page footer form: 0

Copies: 1 Exp: 0

Orientation: Portrait

PDF: Yes

Preview: No

The **PDlg** column (in the I/O Device repository) and the **Preview** property (in the I/O properties) are both set to **No**. In Rich Client tasks, the printing is done on the server and if one of these properties would be set to **Yes**, the **Print** dialog box would be unnecessarily opened.

You can also define specific security options for the PDF file by clicking the **Security** button.

You will learn about some of the other properties in this dialog box later on.

8. Close the dialog box.

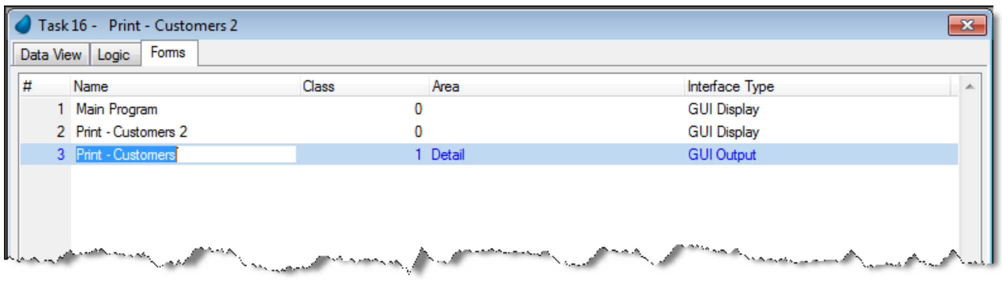
Print - Customers Form

Up until now you have used only one type of form in the **Form Editor**, the Class = 0 form. In Magic xpa the forms are divided into two major classes:

- **Class = 0** is used for GUI display and user interaction.
- **Class > 0** is used for I/O operations.

Now you will use a Class > 0 form. This form will be printed to the PDF that you set in the I/O Device repository.

1. Open the Form Editor.
2. Create a form.
3. In the **Name** column, type: **Print - Customers**.
4. In the **Class** column, type: **1**.


Task 16 - Print - Customers 2

Data View | Logic | Forms

#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	Print - Customers 2	0		GUI Display
3	Print - Customers	1	Detail	GUI Output

Note that the **Class** column value is larger than zero.

5. From the **Interface Type** column, select **GUI Output**.

The dimensions of an A4 page are 210x297mm or approximately 21x30 cm (or 8.3 x 11.7 inches). Setting the **Form units** property to **Centimeters** and the form's **Width** property to **20** cm will ensure that the form you are designing will be presented properly in the print-out device.

You will now set the form properties according to the explanation above.

6. Open the **Form Properties** sheet.
7. In the **Form units** property, select **Centimeters**.
8. In the **Width** property, type: **20**. This will give the form a 1-cm margin.
9. In the **Height** property, type: **3**.

In Magic xpa, when using a Table control in a Class > 0 form, the detail area is duplicated for each printed line. This is unlike Class = 0 forms (GUI Display), where the number of lines is fixed and the end user can scroll between the records.

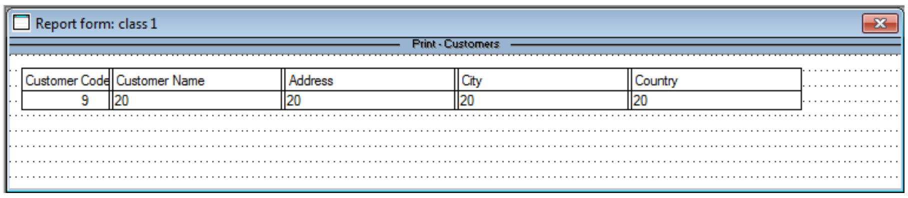
Now, you will add a Table control to the form to display the **Customers** details.

10. Zoom to the **Print - Customers** form.

Notice that the form area looks different. Output forms have a different form designer than Display forms.

11. From the Control palette, select a Table control and place it on the form.
12. Change the **Height** property to **1**.
13. Select the following variables, one by one, from the Variable palette and place them on the Table control:
    Customer_Code, **Customer_Name**, **Address**, **City**, and **Country**.

The form should look like the image below.


Report form: class 1

Print - Customers

Customer Code	Customer Name	Address	City	Country
9	20	20	20	20

14. Exit the Form Editor (**ESC**).

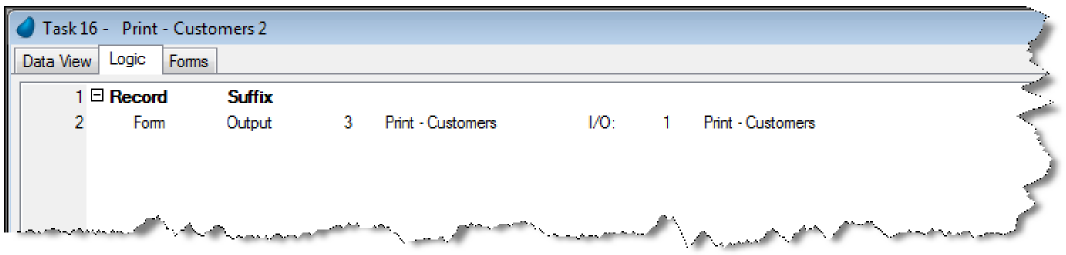
Using the Form Output Operation

A task can have many Class > 0 forms. Magic xpa does not know the order of the **Form Output** operations' output or how many times the forms should be used.

The Form Output operation is used to instruct Magic xpa which form to use and when to send it to the output device. You will now use the Form Output operation in the **Record Suffix** logic unit to print all of the customers in the **Customers** data source.

As previously mentioned, a Table control in a Class > 0 form behaves differently than a Class=0 form; the table detail area is duplicated every time the form is printed out. You decreased the table size on the previous page. This behavior enables you to print data in a table format without having to deal with the table design.

1. Open the Logic Editor and create a **Header** line.
2. Define a **Record Suffix** logic unit.
3. Create a detail line.
4. Select **Form** from the combo box and then select **Output**.
5. Zoom and select the **Print - Customers** Output form.



```
1 ⊟ Record   Suffix
2     Form   Output   3   Print - Customers   I/O:   1   Print - Customers
```

6. Close the program and save the changes.

You have just created a simple report that displays the **Customers** data source. You can run the program from the Program repository and then go to your temporary directory and open the PDF file you just created.

Designed Report

The last section showed you how to create a very basic report; when only the most necessary elements are printed. Now you will learn how to improve the display of your report. You will start by improving the Table control that you created in the last section.

Then, you will add the following forms that will make your report look more professional and comprehensive:

- A page header
- A header
- A footer
- A page footer

To improve the look of the report, you will now add the following:

- A new color
- A new font

Adding a New Color

This example includes adding a Header form to the report. The report header will be text with a specific color and font. Therefore, you will now add a new color to the Color repository.

1. From the File menu, select Application Properties.
2. In the **Application Properties** dialog box, select the **External Files** tab.
3. Park on the Application Color Definition file property.
4. Zoom to the Application Color repository.
5. Park on the last line and create an entry.
6. In the **Name** column, type: **Report Header**.
7. Zoom to the **FG** column.
8. In the **System** entry, select the blank (first) option.
9. In the **Basic Colors** area, define the following shade of blue:
   RGB = 0,64,128.
10. Return to the Application Color repository.
11. Zoom to the **Report Header: BG** dialog box.
12. Check the **Transparent** box.
13. Return to the Application Color repository.
14. Click **OK** (to return to the **Application Properties** dialog box).

In the **Save As** dialog box:

15. In the **Effective immediately** entry, select **Yes**.

Adding a New Font

1. In the **Application Properties** dialog box, park on the **Application Font Definition file** property.
2. Zoom to the Application Font repository.
3. Park on the last line and create a new entry.
4. In the **Name** column, type: **Report Header**
5. Zoom from the **Font** column.
6. In the **Font** section, select **Book Antiqua**. Magic xpa supports only True or Open Type fonts for display. For printing purposes you can use any font that your printer driver supports.
7. In the **Font Style** section, select **Bold Italic**.
8. In the **Size** section, select **18**.
9. Return to the Application Font repository.
10. Return to the **Application Properties** dialog box.

In the **Save As** dialog box:

11. In the **Effective immediately** entry, select **Yes**.

Modifying the output

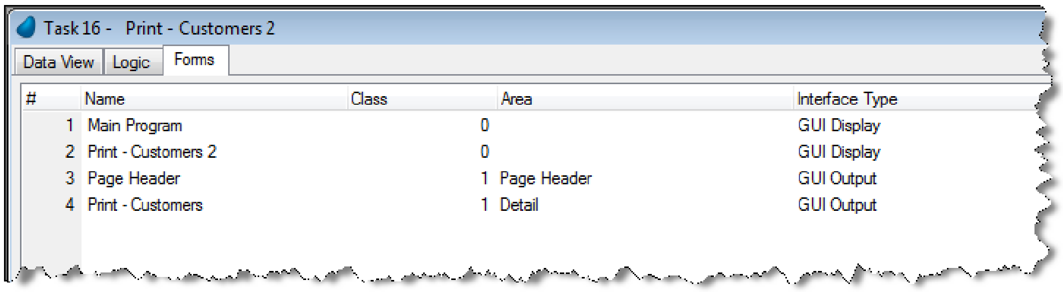
You will now update the color and font for the table's column titles.

1. Open the Program repository and zoom into the **Print - Customers 2** program.
2. Park on the **Print - Customers** form and zoom to the Form Editor.
3. For each column:
   a. Press **Alt+Click** on the column.
   b. In the Column properties (**Alt+Enter**), from the **Font** property, zoom and select the **Text Caption** font.
   c. For the **Customer_Code** column, change the title to **Code**.
   d. In the **Color** property, zoom and select the **Text Caption** color.
4. Close, save, and return to the Form Editor.

Page Header

In the Form Editor:

1. Create a line BELOW the **Print - Customers 2** GUI Display form and name it **Page Header**.
2. Park on the **Area** column and select **Page Header** from the combo box.


Task 16 - Print - Customers 2

Data View | Logic | Forms

#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	Print - Customers 2	0		GUI Display
3	Page Header	1	Page Header	GUI Output
4	Print - Customers	1	Detail	GUI Output

The physical placement of Header, Detail, and Footer forms in the Form Editor is important. Each header form must be followed by its matching footer form. All header-footer pairs must precede the detail lines. Therefore, the new **Page Header** form should be placed before the **Detail** line. This form should also use centimeters as the form unit and have a width of 20. Additional forms in this course will also have these settings. Instead of manually changing this for each form, a better way to do this is to create a model with these dimensions. You will now create a model.

1. Open the Model repository and create a line.
2. Name the model **GUI Print Form**.
3. From the **Class** column, select **GUI Output**.
4. From the **Attribute** column, select **Form**.
5. Set the **Form Units** property to **Centimeters**.
6. Set the **Width** property to **20**.

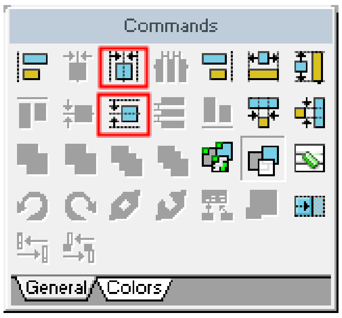
Return to the **Print - Customers 2** program and:

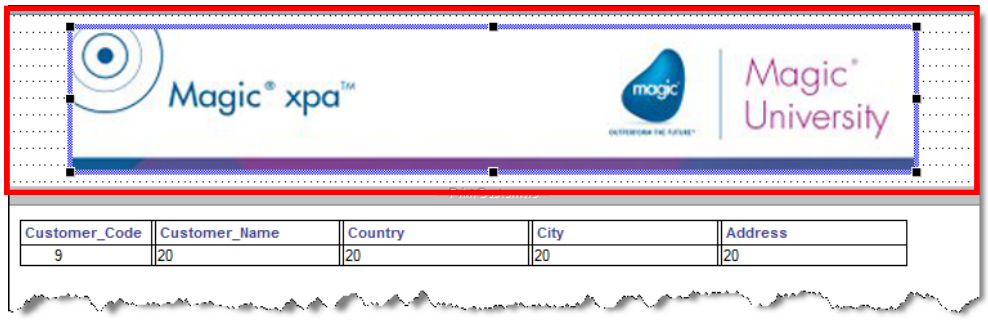
7. Zoom into the Form Editor. Park on the **Page Header** form and select the **GUI Print Form** model.
8. Set the **Height** property to **3.5**.

A **page header**, or simply a **header**, is separated from the main body of text and appears at the top of a printed page. The course data includes the **images** folder, which contains header and footer image files for the report's page header and page footer.

9. Zoom to the **Page Header** form.
10. Place an **Image** control on the form.
11. Open the Image control properties.

12. In the **Default image file** property, type: **%WorkingDir%images\Print_Logo.jpg**.
13. Set the **Image Style** to **Scaled to Fit**.
14. Set the **Width** property to **17.4**.
15. Set the **Height** property to **3**.
16. In the Command palette, click the **Horizontal center of form** command icon.
17. In the Command palette, click the **Vertical center of form** command icon. This will center the Logo image on the form.



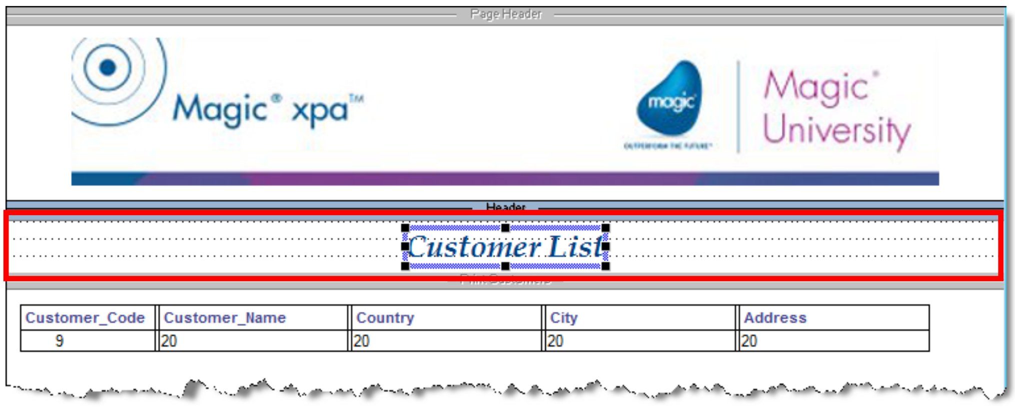


Adding a Report Header

A report header is used to provide a descriptive caption for the report. The Header form is automatically outputted for the second and subsequent pages, whenever a new page of the same class is about to be printed. The Header form will not be printed for the first page. You will now add a Header form to your report.

1. Return to the Form Editor.
2. Park on the new **Page Header** form and create a line.
3. In the **Name** column, type: **Header**.
4. From the **Area** column, select **Header**.
5. Open the **Form Properties** sheet and select the **GUI Print Form** model.
6. Set the **Height** property to **1**.
7. Zoom to the **Header** form.
8. From the Control palette, select a **Text** control and place it on the form.
9. Open the Text control properties and set the following properties:
   a. In the **Text** property, type: **Customer List**.
   b. From the **Font** property, select **Report Header**.
   c. From the **Color** property, select **Report Header**.
10. In the Command palette, click the **Fit Size** (▣) command icon.
11. Click the **Horizontal center of form** (▣) command icon. This will center the text horizontally on the form.

12. Click the **Vertical center of form** (▤) command icon. This will center the text vertically on the form.



Customer_Code	Customer_Name	Country	City	Address
9	20	20	20	20

Adding a Footer

The Footer area is usually used for a report summary.

1. Return to the Form Editor.
2. Park on the **Print - Customers** GUI Output form.
3. Create a line and in the **Name** column, type: **Footer**.
4. From the **Area** column, select **Footer**.
5. Open the **Form Properties** sheet and select the **GUI Print Form** model.
6. Set the **Height** property to **1**.

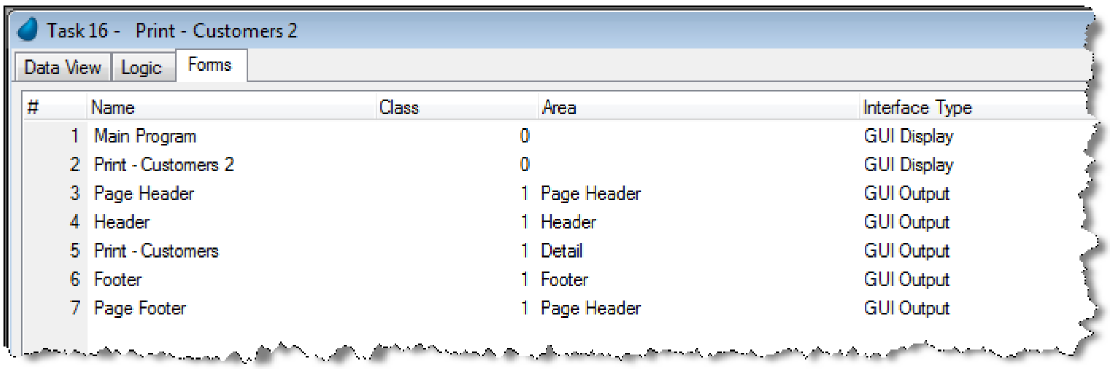
In this example, you will display the total number of printed customers in this report. The **Counter (0)** function returns the number of records processed in the current task. You will use the **Counter (0)** function to return the number of customers printed in the report. The Counter function is only relevant for Batch tasks and non-interactive Rich Client tasks. You will also use the **Str** function to convert the number to a string.

7. Zoom to the **Footer** form.
8. From the Control palette, select an Edit control and place it on the form.
9. Open the Edit control properties and from the **Data** property, enter a new expression: **'Total Number of Customers: '&Trim(Str(Counter(0),'6'))**
10. In the **Format** property, type: **40**.
11. In the **Font** and **Color** properties, select **Text Caption**.
12. Park on the Edit control and move the control to the left.
13. In the Command palette, click the **Fit Size** (▣) command icon.
14. Click the **Vertical center of form** (▤) command icon. This will center the Edit control on the form.

Page Footer

A page footer is usually used for company-related information. In this example you will use an image from the **Images** folder. As with the header forms, the placement of the page footers is important.

1. Return to the Form Editor and park on the **Footer** form and create a line.
2. In the **Name** column, type: **Page Footer**.
3. From the **Area** column, select **Page Footer**.
4. Open the **Form Properties** sheet and select the **GUI Print Form** model.
5. Set the **Height** property to **2**.



Task 16 - Print - Customers 2

Data View | Logic | Forms

#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	Print - Customers 2	0		GUI Display
3	Page Header	1	Page Header	GUI Output
4	Header	1	Header	GUI Output
5	Print - Customers	1	Detail	GUI Output
6	Footer	1	Footer	GUI Output
7	Page Footer	1	Page Header	GUI Output

6. Zoom to the **Page Footer** form.
7. From the Control palette, select an Image control. Close the dialog box.
8. Open the **Image** control properties.
9. In the Default image file property, type: %WorkingDir%images\Print_Footer.jpg.
10. Set the Image Style to Scale to Fit.
11. Set the **Width** property to **18.76**.
12. Set the **Height** property to **1.67**.
13. Close the Image control properties.
14. In the Command palette, click the **Horizontal center of form** command icon.
15. In the Command palette, click the **Vertical center of form** command icon. This will center the Logo image on the form.
16. Close the Form Editor.

Printing the Header and Footer Forms

You have designed the report the way you want it to look. Now you need to print it.

A Header form (a form with the **Area** column set to **Header**) has a unique behavior. The Header form is automatically outputted for the second and subsequent pages, whenever a new page of the same class is about to be printed. The Header form will not be printed for the first page. Therefore, there is a need to manually print it for the first time.

The Task Prefix logic unit is used to perform processes that should be executed once, when starting the program. You will now add a Form Output operation to print the Header form.

1. Open the Logic Editor.
2. Create a **Header** line.
3. Create a **Task Prefix** logic unit.
4. Create a detail line and set the **Form Output** operation to print the **Header** form.

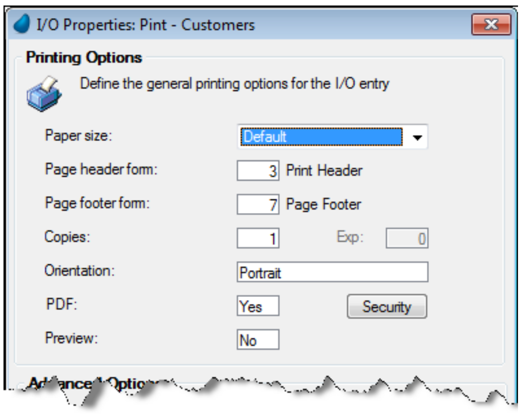
The Footer form is the report footer and it should be printed once on the last page of the report. Therefore, it should be printed from the Task Suffix logic unit, which is executed only once when the program ends.

5. Create a **Task Suffix** logic unit.
6. Create a detail line and set the **Form Output** operation to print the **Footer** form.

Setting the Page Header and Footer

A page header and page footer should be printed on every page regardless of the page content. Therefore, the Page Header and the Page Footer forms are defined in the I/O properties. This way Magic xpa will always include these forms on the printed pages.

1. From the **Task** menu, select **I/O Devices** (Ctrl+I).
2. Park on the **Print - Customers** entry.
3. Open the **I/O Properties** dialog box.
4. Zoom from the **Page header form** property to the Form list and select the **Print Header** form.
5. Zoom from the **Page footer form** and select the **Page Footer** form.
6. Click **OK**. Close the program and save the changes.


I/O Properties: Pint - Customers

Printing Options

Define the general printing options for the I/O entry

Paper size: Default

Page header form: 3 Print Header

Page footer form: 7 Page Footer

Copies: 1 Exp: 0

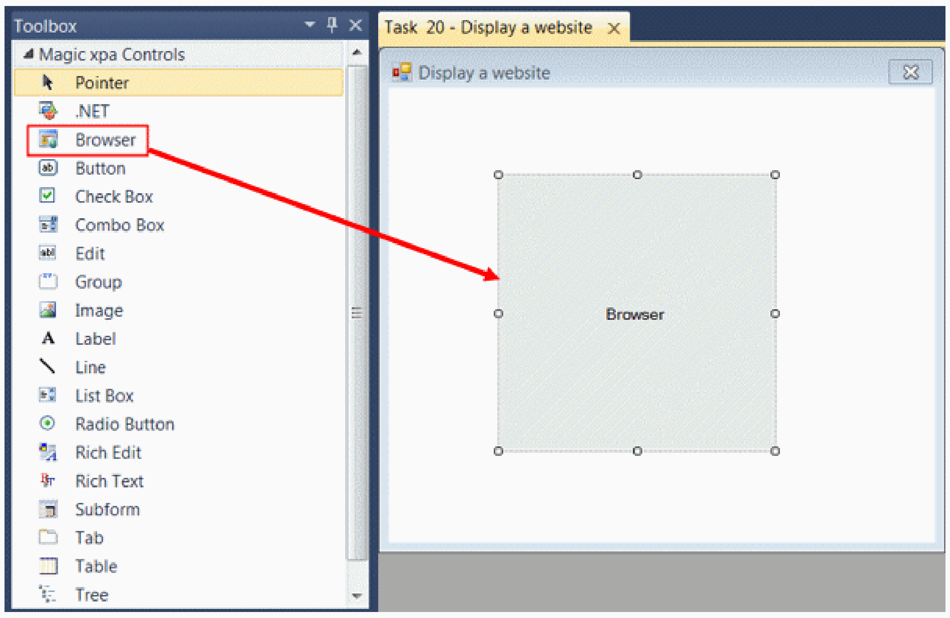
Orientation: Portrait

PDF: Yes

Preview: No

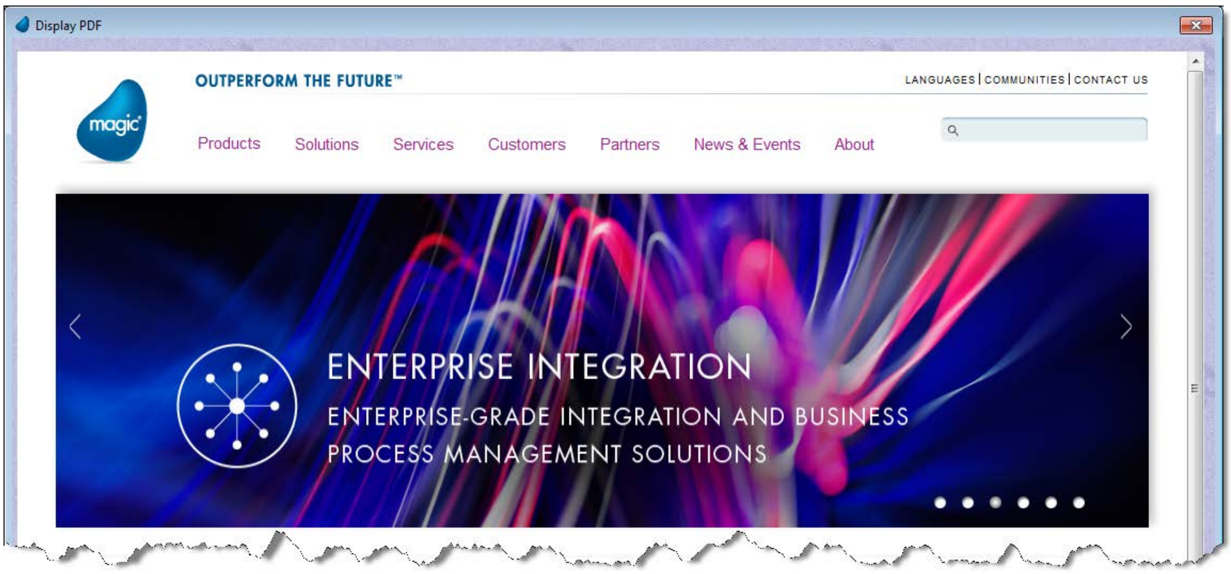
The Browser Control

Printing to a PDF is useful. You can decide when you want to print it or display it. However, you are often in a situation when you need to view the document on your mobile device. One way to handle this is to use the Browser control.



The Browser control is an internal Magic xpa control similar to other controls discussed during this course. The control enables you to display HTML files and any other files that your Internet browser can display.

The Browser control simulates the work of the internet browser. Once this control has been dropped on the form, you have an embedded browser and can use the control to suit your needs. This means that you can use JavaScript with your HTML files.



In the next few pages you will create and call a Rich Client program to display the PDF. To create the **Display PDF** program:

1. Create a new program and name it: **Display PDF**.
2. In the **Task Properties** dialog box, set the **Task type** entry to **Rich Client**.

When you created the PDF, it was created on a location on the server. If the PDF is left on the server then you have a few issues that you need to solve, such as:

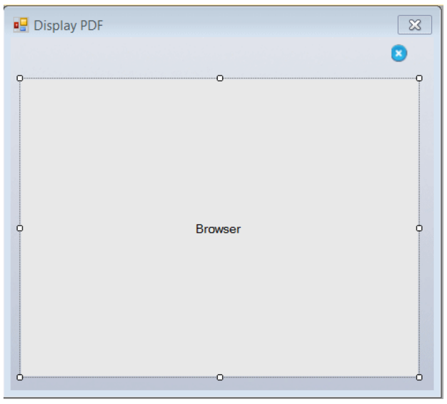
- Naming the PDF – Remember that you may have a number of clients running the same report and to call it **CustomerList.pdf** may not show the same result for another client running the same report. Also what if you are trying to display the report and at the same time another client machine is overwriting?
- Security – Since the directory will be exposed to the web, other clients can see the same report. If you are offering a huge discount to one client, would you want another client to see that? What about if the report is sales figures before reporting them to the stock market?

As a result, the report needs to be copied to the client. Magic xpa provides the **ServerFileToClient** function, which copies a file from the server to the client. The function returns the location of the file on the client. You will now see how to use this:

3. Create a **Parameter** variable named **P.PDF to display** with an attribute of **Alpha** and a size of **255**.

In the Form Editor:

4. Open the **Display PDF** form properties. Use the **Table Display Form** model.
5. Zoom into the Form Designer.
6. Drop a Button control on the form and select the **Image Button** model.
7. Change the name of the Image file to **Exit.png**.
8. Change the name of the **Internal event** to **Exit**.
9. Zoom into the **Placement** property and set the **X** property to **100**. That way when the form increases horizontally, it will move together with the form.
10. Click the Fit Control Size icon.



11. Drop the **Browser control** ( )on the form.
12. Increase the height and width of the control to fill the form as shown in the image above.
13. In the **Placement** property, set the **Width** and **Height** to **100**.
14. Zoom from the **Data** property and create the following expression: **'file:/'&Trim(P.PDF to display)**

This will work on a desktop client as well as an iOS client. It will not work on most Android devices. To view this on an Android device you will need a third party reader.

Displaying the PDF

To display the PDF, you will:

- Create a **Print** push button in the **Customers - Line Mode** program.
- In the **Print** handler:
    - Call the **Print - Customers 2** program.
    - Call the **Display PDF** program.

In this course you will be defining reports for a few programs each with its own Print handler. Instead of defining the handler in each program, you can define it in the Main Program. The definition will then be available for all of the programs.

1. Zoom to the **Main Program**. The Main Program is a parent program for all of the programs. Any object defined here is visible to other programs in the project.
2. Open the **Event** repository and create a new event:
3. In the **Description** parameter, type: **Print**.
4. From the **Trigger type** parameter select **None**.
5. Close the Main Program.
6. Zoom to the **Customers - Line Mode** program.
7. Add a Virtual variable named **PDF Name** with an attribute of **Alpha** and a length of **255**.
8. Open the Form Editor and zoom to the form.
9. Drop a **Button** control on the form. Select the **Image button** model.
10. Open the control property sheet and change the image file name to **Print.png**.
11. From the **Event Type** property, select **User.**
12. From the **Event** property, select the **Print** User event that you defined in the Main Program.
13. Use the **Fit Control Size** icon.
14. Zoom into the Logic Editor.
15. Add a header line for the **Print** User event.
16. Add a detail line and call the **Print - Customers 2** program.
17. Add a detail line and update the **PDF Name** variable with the expression: **ServerFileToClient('%TempDir%CustomerList.pdf')**
18. Add a detail line and call the **Display PDF** program.
19. Zoom into the **Arguments** and add a line. Zoom into the **Var** column and select the **PDF Name** variable.

You can now run the application and view the results.

Displaying a PDF on an Android Tablet

You can view a PDF on an Anroid machine using a third-party PDF viewer application, such as Adobe Acrobat Reader. You do this by using the **Invoke OS command** with the command of **pdf:filename**, where **filename** is a path to a local file that is stored on a public folder on the device, such as the /SDCARD folder.

The **ServerFileToClient** function copies the file to a local folder that only the current client has access to, in this case the Magic xpa client. To be able to view the file in a third party application, you need to copy it to a public folder. You do this by using the **ClientFileCopy** function.

In the **Print** User event logical unit, you used the **ServerFileToClient** function to return the location of the file on the client to a variable named **PDF Name**. For example, the following commands will copy the ReleaseNotes.pdf file from the C:\Magic\ folder on the server to the /SDCAD folder on the client and will open a PDF viewer to show this file:

1. Add a detail line and create an **Evaluate Expression** operation.
2. Set the expression to:
   ClientFileCopy(Trim (PDF Name),'/sdcard/CustomerList.pdf')
3. Add a detail line and create an **Invoke OS Cmd** operation.
4. Set the expression to: 'pdf:/sdcard/CustomerList.pdf'
5. Set the **Execute On** property to **Client**.

When dealing with an Android client, calling the **Display PDF** program will give an error message to the user.

If you are developing an application for both Android and iOS devices you can use an expression to define whether the **Display PDF** program will be called or the Android section. You will learn how to do this in a later lesson.

Printing a PDF on a Mobile Device

Very often you need to actually print the report. You can do this on your iOS device to printers that support the AirPrint protocol and on your Android device to printers that support the Google Cloud Print protocol. When you display the PDF in the Bowser control, you have the print option. Here you can select the printers in your wireless network that support the AirPrint protocol or Google Cloud Print protocol. If you want to print the report without displaying it first, you can do this by using the **Invoke OS Cmd** operation with the syntax of: "**Print:filename**" where **filename** is either:

- A path to a local file, as received from the **ServerFileToClient** function
- A URL to a file on a Web server, such as 'http://example.com/PublishedAplications/CustomerList.pdf' (This option is only available for iOS devices)

One property of the **Invoke OS Cmd** operation is **Execute on**. You can select either **Client** or **Server**. For printing, you need to select **Client**.

Complex Report Concept

When you look at a report, it often seems one-dimensional; but if you look carefully you can see that the data can be arranged in various formats. Moreover, in one report you can have several formats (layers). For example, in an invoice you can see:

- General information, such as the customer name and address, at the top of the page
- A list of the items that were ordered, in the middle of the page
- Summary information, such as the total amount, at the bottom of the page.

The layers can be arranged so that they present the best visualization of the information. The information can be retrieved from one or more data sources, and can be processed in one or more tasks.

There are a number of points to remember when creating complex reports:

- I/O devices should be defined in the parent task only, and shared between the parent and the child tasks.
- Parent task forms can be accessed from the subtask's Form Editor.
- Parent-child forms that share the same class (Class>0) should have the same width and the same interface type.

In this lesson you will create an **Invoice** printout using the **Orders**, **Order Lines**, **Customers**, **Products**, and **Suppliers** data sources.

1. Open the Program repository and create a program named: **Invoice**.
2. Zoom into the program and the **Task Properties** dialog box will open.
3. From the **Task type** property, select **Batch**.
4. Set the Initial mode to Query.
5. Open the Data View Editor and set the **Orders** data source as the **Main Source**.
6. Set Index number **1** as the Main Source's **Index**.
7. Add all of the **Orders** data source's columns to the data view.
8. Park on the last line in the Data View Editor.
9. Select **Create Header Line** (Ctrl+H) and select a **Link Query** to the **Customers** data source.
10. From the **Index** entry, select the first index.

The **Customer_Code** column will be selected automatically within the **Link Query** operation.

1. Add the following columns to the **Link** section: **Customer_Name**, **Address, City** and **Country.**
2. Within the **Link Query**, park on the **Customer_Code** column's **Locate From** entry.
3. Zoom to the Expression Editor and create an expression for the **Customer_Code** from the **Orders** data source.
4. Enter the same expression for the **Locate To** entry.

Forms

Now you will create the **Page Header** and **Page Footer** forms for the report. These are very similar to other reports you created in this lesson.

5. Open the Form Editor.
6. Create a line and in the **Name** entry, type: **Page Header**.
7. Create the **Page Header** as you learned in a previous lesson. Use the **GUI Print Form** model.
8. Create a line and in the **Name** entry, type: **Page Footer**.
9. Create the **Page Footer** as you learned in a previous lesson. Use the **GUI Print Form** model.

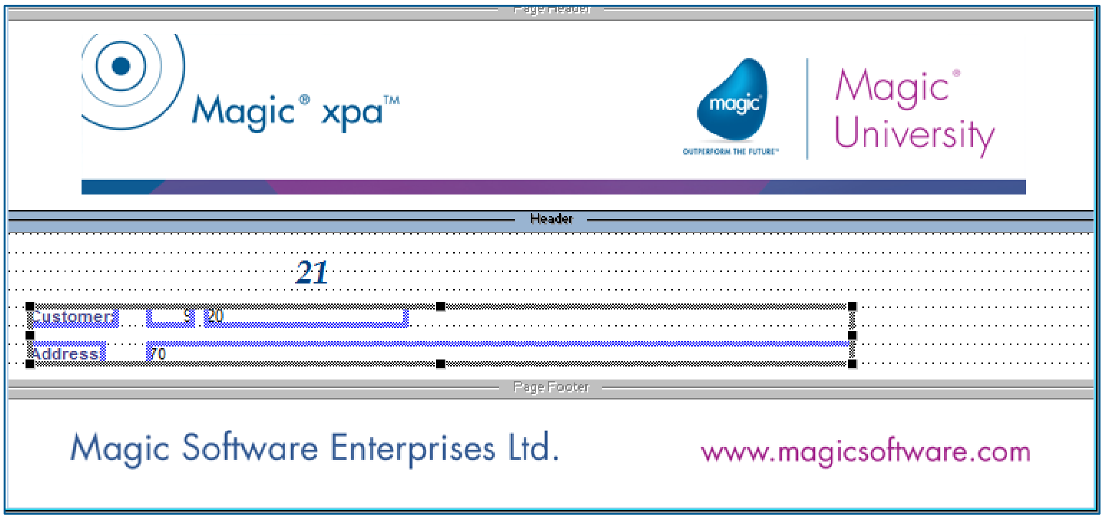
Now you will create the Header form. This form will contain the report name as well as the order title information.

1. Park on the **Page Header** form line and create a line.
2. In the **Name** column, type: **Header**.
3. From the **Area** column, select **Header**.
4. Open the Form Properties and select the **GUI Print Form** model.
5. Set the **Height** property to **2.67**.
6. Zoom to the **Header** form.
7. From the Control palette, select the **Edit** control and place it on the form.
8. Open the **Edit Control** property sheet and from the **Data** property, zoom to the Expression Editor and create the following expression:
   'Order Number: '&Trim(Str(Order_Number,'6'))
9. The **Format** has a default of 30.
   You can change this to 21 since this is the maximum size of the data in the expression.
10. From the **Font** property, select the **Report Header** font.
11. From the **Color** property, select the **Report Header** color.
12. From the Command palette, select the **Fit to Size** command icon. This will fit the control to its defined measurements.
13. From the Command palette, click the **Horizontal center of form** command icon. This will center the Edit control on the form.

The Header form will include more information than just the report name. You will now add customer details to the Header form.

1. From the Variable palette, select the **Customer_Code** variable (from the **Orders** data source) and place it on the form (see the image on the next page).
2. From the Variable palette, select the **Customer_Name** variable and place it to the right of the **Customer_Code** control. Delete the **Customer_Name** caption (The static control only).
3. Open the **Customer_Code** static control properties.
4. In the **Text** property, type: **Customer:**
5. From the **Font** property, select the **Text Caption** font.
6. From the **Color** property, select the **Text Caption** color.
7. From the Control palette, select the **Text** control and place it on the form
8. In the **Text** property, type: **Address:**
9. From the **Font** property, select the **Text Caption** font.
10. From the **Color** property, select the **Text Caption** color.
11. From the Control palette, select the **Edit** control and place it on the form.
12. From the **Edit Control** property sheet, go to the **Data** property, zoom to the Expression Editor and create a line.

13. Type the following expression:
    Trim(Address)&', '&Trim(City)&', '&Trim(Country)
14. In the **Format** property, type the value **70**.
15. From the **Font** property, select the **Text** font.
16. From the **Color** property, select the **Text** color.



The **Footer** form will include information about the order amount and payment terms. You will now add the order amounts and the payment terms to the Footer form.

1. Return to the Form Editor.
2. Park on the **Header** form and create a line.
3. In the **Name** column, type: **Footer**.
4. From the **Area** column, select **Footer**.
5. Open the Form Properties and select the **GUI Print Form** model.
6. Set the **Height** property to **1**.
7. Zoom to the **Footer** form.
8. From the Variable palette, select the **Amount** variable and place it on the form.
9. Zoom into the control properties of the **Amount** Edit control and
    a. From the **Font** property, select the **Text** font.
    b. From the **Color** property, select the **Text** color.
10. Zoom into the control properties of the **Amount** Text control and
    c. In the **Text** property, type: **Amount:**
    d. From the **Font** property, select the **Text Caption** font.
    e. From the **Color** property, select the **Text Caption** color.
11. Use the **Fit to Size** icon for both controls.

Defining the I/O Device

1. From the **Task** menu, select **I/O Devices** (Ctrl+I).
2. Add a line and set the name to **Print Invoice**.
3. From the **Exp/Var** column, zoom and set the following expression: **'%TempDir%Invoice.pdf'**
4. Open the **I/O Properties** dialog box.
5. From the **Page header form** property, zoom and select the **Page Header** form.
6. From the **Page footer form** property, zoom and select the **Page Footer** form.
7. From the **PDF** property, select **Yes**.
8. Exit the I/O Device repository.

Print Invoice Lines Subtask

The information for the Details form needs to be retrieved from the **Order_Lines** data source. Therefore, you now need to create a subtask that will scan and print the Order details.

1. In the Navigator pane, park on the **Invoice** entry and create an entry.
2. In the **Task name** property, type **Print Invoice Lines**.
3. From the **Task type** property, select **Batch**.
4. Set the Initial Mode to Query.
5. Open the Data View Editor.
6. Set the **Order_Lines** data source as the **Main Source**.
7. Select the first index as the **Main Source index**.
8. Add all of the main source's columns to the data view.
9. Park on the **Order_Number** row.
10. From the **Range** property, zoom to the Expression Editor and create a line.
11. Add an expression for the **Order_Number** column from the **Orders** data source, from the parent program.
12. Enter the same expression in the **Range To** property.

You will now link to the **Products** data source to retrieve the Product Name and the Supplier Code. You will use this Supplier Code to get the Supplier Name.

13. Park on the last line in the Data View Editor and create a Header line.
14. Set a **Link Query** to the **Products** data source.
15. From the **Index** property, zoom and select the first index.
16. Add the following columns to the **Link** operation: **Product_Name** and **Supplier_Code**.
17. In the **Link Query** operation, park on the **Product_Code** column's **Locate From** property.

18. Zoom to the **Expression Editor** and create an expression for the **Product_Code** column from the **Order_Lines** data source.
19. Enter the same expression for the **Locate To** entry.

The next section is added so that you can see how to use multiple links in a program.

1. Park on the last line in the Data View Editor and create a Header line.
2. Set a **Link Query** to the **Suppliers** data source.
3. From the **Index** property, zoom and select the first index.
4. Create a line and add the **Supplier_Name** column to the **Link** operation.
5. Park on the **Supplier_Code** column's **Locate From** entry.
6. Zoom to the **Expression Editor** and add an expression for the **Supplier_Code** from the **Products** data source.
7. Enter the same expression in the **Locate To** entry.

Now you will create the details form:

8. Open the Form Editor and create a form.
9. In the **Name** column, type: **Print Invoice Lines**.
10. Open the Form Properties and select the **GUI Print Form** model.
11. Set the **Height** property to **2.2**.

Magic xpa enables you to view the parent task's forms from the subtask's Form Editor in order to make the development process easier.

12. Zoom to the **Print Invoices Lines** Output form.
13. Place a Table control on the form.
14. Attach the following variables to the Table control:
    - **Line Number** (Alt+Click on one of the empty lines and change the name of the column header to **Line**. Change the width of the column to **0.8**.)
    - **Product_Name** (from the **Products** data source)
    - **Supplier Name** (from the **Supplier** data source)
    - **Product_Quantity** (Change the name of the column header to **Quantity**. Change the width of the column to **1.4**.)
    - **Product_Price** (Change the name of the column header to **Price**. Change the width of the column to **1.6**.)
15. In the **Product_Name** column, mark the Edit control and decrease its **Width** property to **9.5**.
16. Mark the Column control and decrease its **Width** property to **9.75**.
17. Change the name of the column header to **Product**.

18. Attach an Edit control to the table and mark it.
19. Open the **Edit control** properties and from the **Data** property, zoom to the Expression Editor.
20. Create the following expression: **Product_Price * Product_Quantity.**
21. In the **Format** property, type: **8.2**
22. In the **Width** property, type: **2**.
23. Mark the column where you placed the Edit control and open the **Column** properties.
24. In the **Column title** property, type: **Line Amount**.
25. In the **Width** property, type: **2.2**.
26. Mark all of the columns in the Table control.
27. Open the Multi Selection Properties sheet (**Alt+Enter**).
28. From the **Font** property, select the **Text Caption** font.
29. From the **Color** property, select the **Text Caption** color.

Now you will output the **Print Invoice Lines** form.

1. Open the Logic Editor.
2. Create a **Record Suffix** logic unit.
3. Add a Form Output operation and print the **Print Invoice Lines** form.

Now you need to output the Header and Footer forms:

4. In the **Navigator** pane, click the **Print Order** parent task.
5. In the **Print Order** task, open the Logic Editor and create a **Record Suffix** logic unit.
6. Add a **Form Output** operation and print the **Header** form.
7. Add a **Call Subtask** operation and call the **Print Invoice Lines** form.
8. Add a **Form Output** operation and print the **Footer** form.

Short Summary

The example showed you how to create a complex report.

In this example, the report header and report footer were created in one task (parent task) and the report details were created in another task (child task). Both tasks output the forms to the same I/O device that was set in the parent task.

Unlike the previous example, where the Report Header form was output in the **Task Prefix** logic unit and the Report Footer form was output in the **Task Suffix** logic unit, in this example both the Header and Footer forms were output in the parent task's **Record Suffix** logic unit. This is because, in this example, the Header information and the Footer information are processed from the **Orders** data source. The information from the Details form is handled in the subtask; therefore, you called the subtask after the Header form output and before the Footer form output.

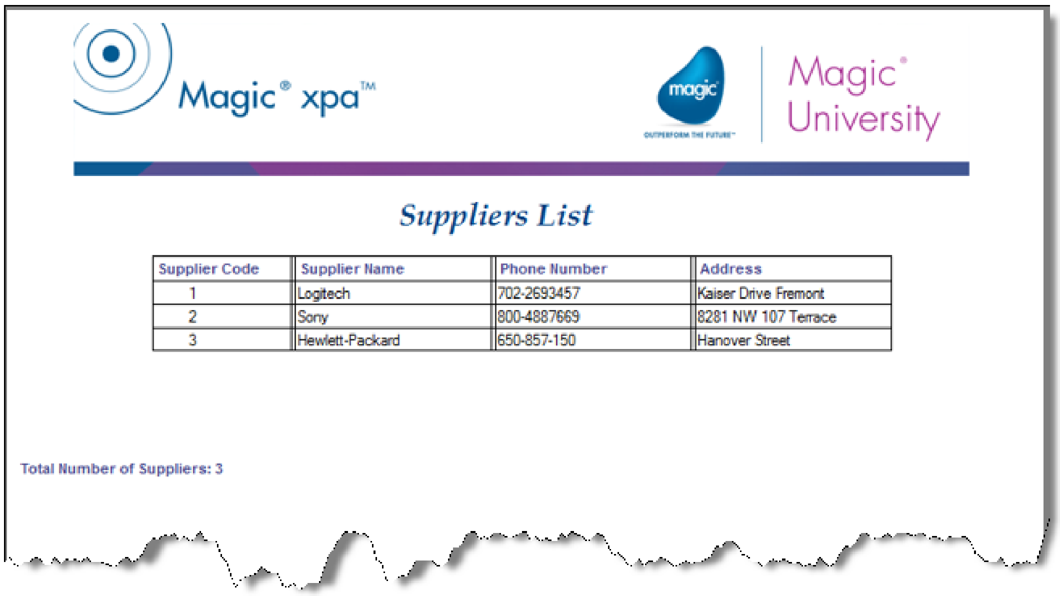
In the subtask, you extended the data view by linking to the **Products** data source and to the **Suppliers** data source. As you may have noticed, you can link to a data source (**Suppliers** in this example) based on information retrieved from another linked data source (**Products** in this example).

The result is a complex report, which displays all of the orders in the data source and the order lines for each order.

Exercise

1. The correct way to print an invoice would be to print a specific invoice. Print an invoice for a specific order from the **List of Orders** program.
2. Using what you learned so far in this course, create a **Suppliers List** report.
3. You should create the report in the same way that you created the **Customers List** report. Invoke the **Suppliers list** report from the **Suppliers - Line Mode** program.

When you are finished, the **Suppliers List** report should look similar to the image below.



Suppliers List

Supplier Code	Supplier Name	Phone Number	Address
1	Logitech	702-2693457	Kaiser Drive Fremont
2	Sony	800-4887669	8281 NW 107 Terrace
3	Hewlett-Packard	650-857-150	Hanover Street

Total Number of Suppliers: 3

Summary

In this lesson you learned how to create a basic report.

You started the lesson by generating a report program using the Generate Program utility.

You learned about the Browser control and how to use the control to display a PDF

Then, you manually created the same report.

You learned about the I/O Device repository.

You learned about forms with the **Area** column bigger than zero, which are used for I/O devices.

You learned about the Form Output operation.

Then, you learned how to enhance your report by adding the:

- Page header
- Header
- Footer
- Page footer

You learned about the role of each form on the page.

You learned that Header forms are printed automatically for every new page from the same area, beginning with the second page and onwards.

You learned where to print the Header form and the Footer form.

You learned where to set a page header and a page footer so that they will be printed on every page for the same I/O device.

You learned how to display the PDF on your device by first copying the PDF to the client and then displaying it using the Browser control for iOS devices..

Lesson 17

Processing Data by Groups

In some cases, several records stored in a data source can have a common denominator. For example, in the **Order_Lines** data source, records that belong to the same order can be referred to as one group. Another example is cities and countries; all of the cities for the same country can be referred to as one group.

Magic xpa enables you to process data segmented as groups, using the **Group** logic unit. The Group logic unit can be defined in Batch tasks only.

You can define the following Group logic units:

- Group Prefix
- Group Suffix

This lesson covers various topics including:

- Grouping data in Batch programs
- Using the Group logic unit
- Group logic unit operations in the Task execution process
- The correlation between the Sort criteria and the Group logic units
- Defining a Sort for a task

About the Group Logic Unit

The Group logic unit creates breaks during the data view processing.

You can use the Group logic unit to calculate sums at certain levels, to display data by groups, or to print breaks in a report.

You define a Group logic unit for a variable. The Group logic unit is executed when the variable value changes.

You can define several Group logic units for different variables in the same task.

The order of the Group logic units needs to match the order of the segments in the task's sort criteria. For example, if the task's **Main Source** index segments are **Country_Code** and then **City_Code**, the Group logic unit should be defined in the same order; first the Group logic unit for the **Country_Code** variable and then the Group logic unit for the **City_Code** variable.

Grouping data according to part of the segments in the sort criteria is allowed, as long as the segments are used from top to bottom. Grouping data starting from the middle of the sort criteria segments may give an undesirable result.

Group Logic Unit Execution Order

Before you begin using the Group logic unit, it is important that you understand how the Magic xpa engine behaves when a Group logic unit is defined in a task.

The Magic xpa engine scans the Main Source records according to the sort criteria (a specified index or the specified task sort).

- The **Group Prefix** operations are executed before the first record in the group is processed.
- The **Group Prefix** operations are executed before the **Record Prefix** operations.
- The **Group Suffix** operations are executed after the last record in the group is processed.
- The **Group Suffix** operations are executed after the **Record Suffix** operations are executed.

When the value of the variable defined in the Group logic unit is changed, the Group logic units are executed accordingly.

The following displays the execution of the logic units:

- Task Prefix
    - Group Prefix (for the first record or when the group's variable value is changed)
        - Record Prefix
        - Record Suffix
    - Group Suffix
- Task Suffix

Engine Flow for a Batch Task

Previously you learned about the engine flow in a Batch task. This section explains the processes that are executed during a Batch task's life cycle when a Group logic unit exists.

Task Initialization

The Task Initialization operations are similar to the operations in an interactive task.

Group Processing

- The first record is fetched into memory.
- If the **Evaluate condition** task property is set to **Before entering record**, the End Task condition is checked.
- The Exit action is checked, whether or not the user has pressed **ESC**.
- The Group Prefix logic unit operations are executed for the first task cycle and whenever the Group logic unit's variable value is different from the previous record value.

Record Processing

- The Record Prefix logic unit operations are executed.
- The Record Suffix logic unit operations are executed.
- The record is saved with the new values. If the Task mode is set to **Delete**, the record is deleted.

End Group Processing

- When a new record is fetched and the Group logic unit's variable value is different from the previous record value, the Group Suffix logic unit operations are executed for the previous record and the Group Prefix logic unit operations are executed for the current record.

Sorting the Data

The data view can be sorted in two ways:

- Using the **Main Source** index – In this case, the order of the index segments must match the order of the Group logic units.
- Using the task's **Sort Indicator** repository – This option can be used when you do not have an index that matches the order of the Group logic units.

Group logic units must follow two basic rules:

- The order of the index segments should match the order of the Group logic units. The higher segment in the index should be the higher Group logic unit.
- The **Group Suffix** should follow the **Group Prefix**. You should **not** open a new Group logic unit between them.

You do not have to create a **Group Prefix** logic unit or **Group Suffix** logic unit if you do not have any operations to define in them.

The following example will help clarify the above.

Assume you have a data source that contains a list of customers with address details, including the country, city, and street name. You want to print the list of customers according to the country and the city that they live in.

You need to define two Group logic units. The first Group logic unit will be for the **Country** variable and it should be the topmost Group logic unit in the Logic Editor. The second Group logic unit will be for the **City** variable and it should be defined after the first Group logic unit in the Logic Editor.

The index should have the following segments and in the following order: **Country** and then **City**.

Using the Sort option with large data sources may slow the task's performance since Magic xpa builds a temporary index to sort the data source records.

You will now create a program that prints the countries and their cities.

1. Create a program named **Print Countries and Cities**.
2. In the **Task Properties** dialog box, set the **Task type** property to **Batch**.
3. Set the **Initial mode** property to **Query**.
4. In the Data View Editor, set the **Cities** data source as the **Main Source**.
5. Select the second index as the **Main Source index**.
6. Create a line and add all of the columns of the **Cities** data source to the data view.

To be able to show the country name, you need to link to the **Countries** data source.

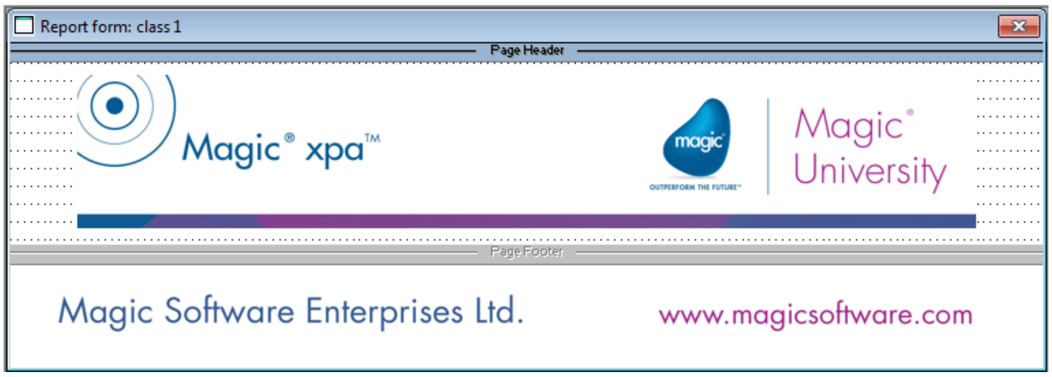
1. Park on the last line in the Data View Editor.
2. Create a **Header line** (Ctrl+H) and create a **Link Query** to the **Countries** data source.
3. Select the first index.
4. In the **Link Query** operation, create another line and add the **Country_Name** column.
5. Set the link's **Locate** properties of the **Country_Code** column to the **Country_Code** variable from the **Cities** data source.

You need a Virtual variable to hold the number of cities for each country.

1. Park on the last line in the Data View Editor.
2. Create a Virtual variable named **Number of Cities** with an **Attribute** of **Numeric** and a **Size** of **6**.

Now you will prepare the form as you did in the previous lesson:

1. Open the Form Editor and create a line. In the **Name** entry, type: **Page Header**. Set the **Model** to **GUI Print Form**.
2. Create the form as you did in the previous lesson.
3. Create a line and in the **Name** entry, type: **Page Footer**. Set the **Model** to **GUI Print Form**.
4. Create the form as you did in the previous lesson.



5. Park on the **Page Header** form and create a line.
6. Create a **Header** form and name it **Country Header**. Set the **Model** to **GUI Print Form**.
7. In the **Height** property, type: **1**.
8. Zoom to the **Country Header** form.
9. From the Control palette, select the Edit control and place it on the form.
10. Open the **Edit** Control Properties.
11. From the **Data** property, create an expression:
    'List of cities for: '&Trim(Country_Name)
    (select the **Country_Name** variable from the list).
12. In the **Format** property, type **43**.
13. In the **Font** property, select the **Report Header** font.
14. In the **Color** property, select the **Report Header** color.
15. From the Command palette:

    a. Click the **Fit to Size** command icon.
    b. Click the **Horizontal center of form** command icon.
    c. Click the **Vertical center of form** command icon.

Creating the City Detail Form

1. Return to the Form Editor.
2. Park on the **Country Header** form and create a line.
3. Create a **Detail** form and name it **City List**. Open the Form Properties and attach the **GUI Print Form** model.
4. In the **Height** property, type: **2.2**.
5. Zoom to the **Cities List** form.
6. From the Control palette, select a Table control and place it on the form.
7. Select **City_Code** and **City_Name** variables and place them on the Table control.
8. Mark the **City_Code** column and set the **Color** to **Text Caption** and the **Font** to **Text**.
9. Set the **Column Title** property to **Code**.
10. Mark the **City_Name** column and set the **Color** to **Text Caption** and the **Font** to **Text Caption**.

Creating the Country Footer Form

1. Park on the **City List** form and create a line.
2. Create a **Footer** form and name it **Country Footer**. Attach the **GUI Print Form** model.
3. In the **Height** property, type: **1**.
4. Zoom to the **Country Footer** form.
5. From the Control palette, select the Edit control and place it on the form.
6. Open the Edit control's property sheet and in the **Data** property, enter the following expression:
   'Total number of cities in '&Trim(Country_Name)&': '&Trim(Str(Number of Cities,'6'))
   Select the **Country_Name** and **Number of Cities** from the Variable list.
7. Set the **Format** to **60** and the **Font** and **Color** to **Text Caption**.
8. Click the **Fit to Size** command icon.

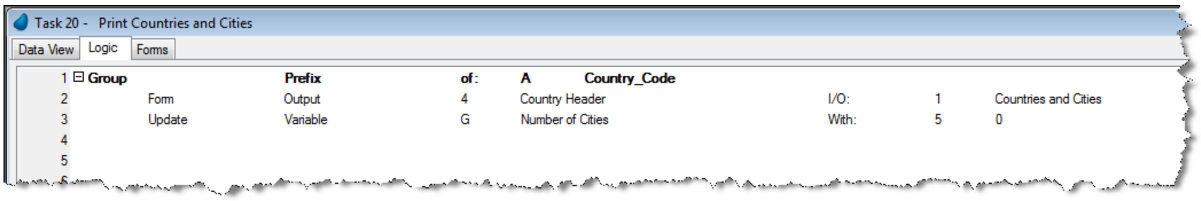
Defining the I/O Devices

1. From the **Task** menu, select **I/O Devices**.
2. Create a line and set the **Name** to **Countries and Cities**.
3. From the **Exp/Var** column, zoom and set the following expression:
   '%TempDir%CountryCityList.pdf'
4. Open the **I/O Properties** dialog box.
5. From the **Page header form** property, zoom to the Form list and select the **Page Header** form.
6. From the **Page footer form** property, zoom to the Form list and select the **Page Footer** form.
7. Set the **PDF** property to **Yes**.

Defining the Group Logic Units

Each group of cities should have both a header and a footer. To print the group header, you will use the Group logic unit for the **Country_Code** variable. This way, every time the **Country_Code** variable value is changed, a new group Header form will be printed.

1. Open the Logic Editor and create a Header line.
2. Set a **Group Prefix** logic unit for the **Country_Code** variable (from the **Cities** data source).
3. Add a **Form Output** detail operation and print the **Country Header** form.
4. Add an **Update Variable** detail operation and update the **Number of Cities** variable with zero. You are resetting the counter every time the **Country_Code** changes.



Task 20 - Print Countries and Cities

Data View | Logic | Forms

```
1 ⊟ Group          Prefix     of:  A   Country_Code
2       Form       Output     4    Country Header     I/O:   1   Countries and Cities
3       Update     Variable   G    Number of Cities   With:  5   0
4
5
```

Now you will print out the Detail form for each city in the group. In addition, you need to increase the **Cities Counter** variable value by one, for each city in the group. This is performed in the **Record Suffix** logic unit.

1. Create a **Record Suffix** logic unit.
2. Add a **Form Output** detail operation and print the **City List** form.
3. Add an **Update Variable** detail operation and update the **Number of Cities** variable with **Number of Cities+1**. This increases the counter of the number of cities.

Now you need to print the footer showing how many cities there are for that country. This is performed in the **Group Suffix** logic unit.

1. Create a **Group Suffix** logic unit for the **Country_Code** variable.
2. Add a **Form Output** detail operation and print the **Country Footer** form

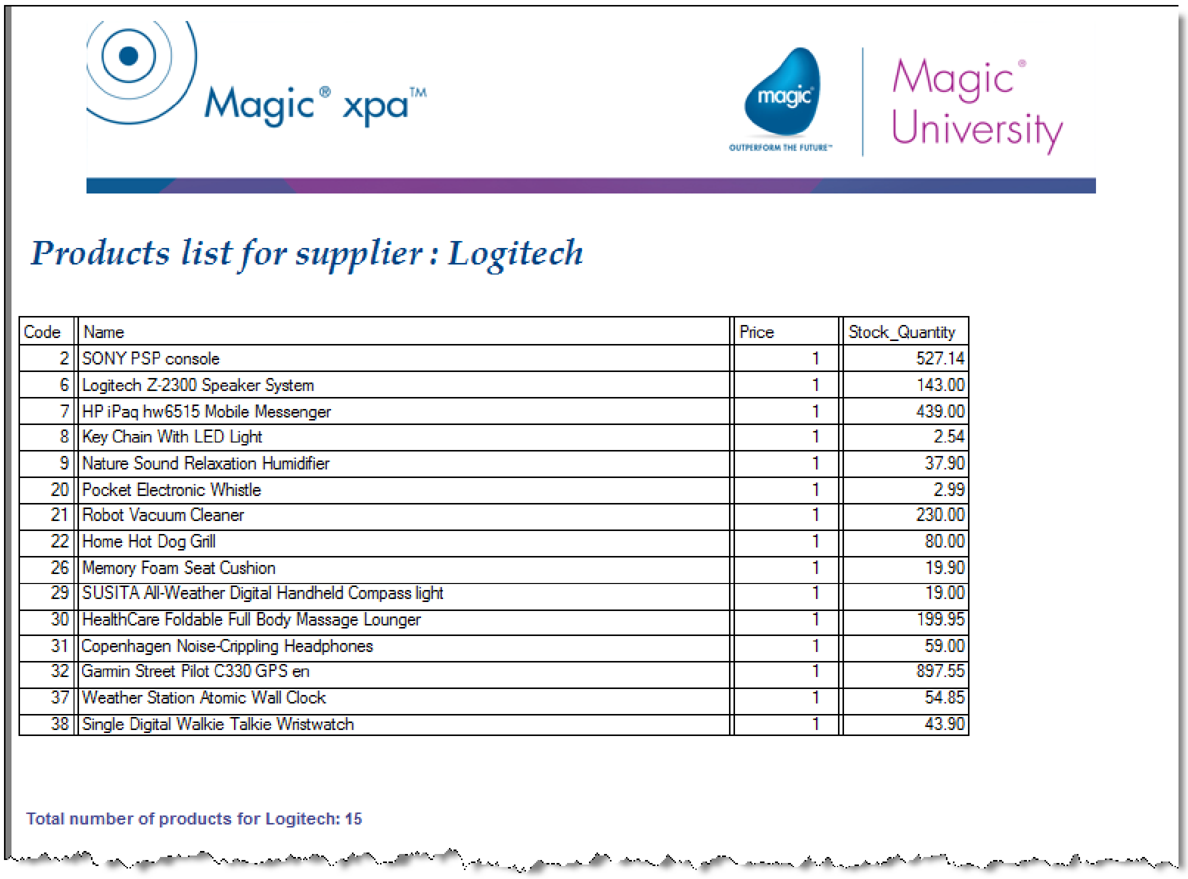
You can now run the program. You can also execute the program using the same methods that you used in the previous lesson, such as adding a Print icon to the **Countries** program.

Exercise

Create a report that displays each supplier and a list of its products. At the end of each supplier, add the number of products for the supplier.

The image below displays a sample of the result report.

As a small challenge, print each new supplier on a new page.



Products list for supplier : Logitech

Code	Name	Price	Stock_Quantity
2	SONY PSP console	1	527.14
6	Logitech Z-2300 Speaker System	1	143.00
7	HP iPaq hw6515 Mobile Messenger	1	439.00
8	Key Chain With LED Light	1	2.54
9	Nature Sound Relaxation Humidifier	1	37.90
20	Pocket Electronic Whistle	1	2.99
21	Robot Vacuum Cleaner	1	230.00
22	Home Hot Dog Grill	1	80.00
26	Memory Foam Seat Cushion	1	19.90
29	SUSITA All-Weather Digital Handheld Compass light	1	19.00
30	HealthCare Foldable Full Body Massage Lounger	1	199.95
31	Copenhagen Noise-Crippling Headphones	1	59.00
32	Garmin Street Pilot C330 GPS en	1	897.55
37	Weather Station Atomic Wall Clock	1	54.85
38	Single Digital Walkie Talkie Wristwatch	1	43.90

Total number of products for Logitech: 15

Summary

In this lesson you were introduced to the Group logic unit.

You learned when and how to use the Group logic unit.

You learned how to define a Group logic unit.

You used the Group logic unit in a Batch task in order to group data and print a report with breaks.

The following information is important to keep in mind when defining a Group logic unit:

- The Group variables must match the Main Source index segments or the Sort segments.
- You can define several Group logic units for different variables.
- The order of the Group logic units within the Logic Editor defines the grouping order (the topmost Group logic unit is the highest Group level and the bottommost Group logic unit is the lowest Group level).
- The Group Prefix logic unit is executed before the first record is processed and each time its variable value changes.
- The Group Suffix logic unit is executed after the last record is processed from the same group.

Lesson 18

Menus

Pulldown menus are the type of menus commonly used in menu bars (usually near the top of a window or screen), which are most often used for performing actions. This type of menu is not used in mobile devices.

Context menus are accessed by right-clicking. The choices presented in the context menu are automatically modified according to the current context.

Magic xpa enables you to define menus in your project. The project menus serve as an interface for the end user to run the project programs during runtime.

This lesson will introduce you to the following Magic xpa menu options:

- Default Pulldown menu – Magic xpa provides a default pulldown menu when creating a project. This menu contains all of the basic options that are needed during runtime.
- Toolbar menu – A set of buttons displayed below the pulldown menu, which serve as shortcuts to pulldown menus.
- Additional menu structures – These can be created as required. These menus can be used from the project as context menus.

This lesson covers various topics including:

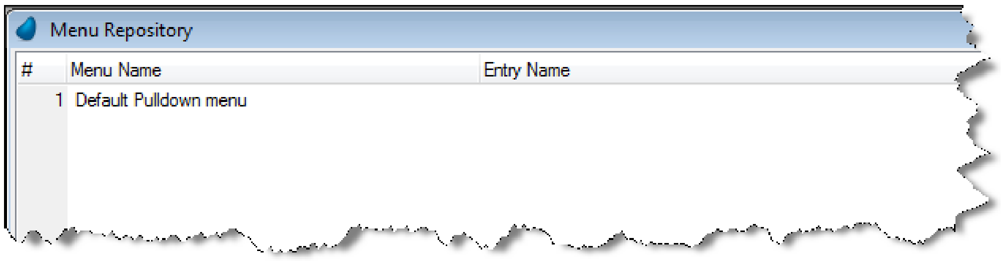
- What menus are used for
- Defining pulldown menus
- Defining SDI forms
- Defining context menus
- Attaching a context menu to a control or a form
- Raising an event from a Menu entry
- Adding a menu to the toolbar menu
- Defining MDI forms

The Menu Repository

The Menu repository contains the **Default Pulldown menu** and enables you to define additional menus.

1. From the **Project** menu, select **Menus** (Shift+F6).

Property	Description
Menu Name	This column displays the logical name of the menu structure.
Entry Name	This column is used by various Magic xpa menu-related functions to control the menu's display. This column should have a unique value for each menu entry. The use of the backslash character (\\) is not allowed in this column.



The Default Pulldown Menu

Every new project is generated with a **Default Pulldown menu**. The **Default Pulldown menu** includes all of the basic options that are needed during runtime. For example, the **File** menu contains all of the file options, such as: **Open Application, Close Application**, and **Exit**.

The **Default Pulldown menu** is only used in desktop applications. For mobile applications this needs to be removed since it is still processed by the engine.

2. Park on the **Default Pulldown menu**.
3. Delete the entry so that you now have an empty repository.

A pulldown menu is not used in a mobile environment. Later in this lesson you will learn how to implement a pulldown menu for a regular Rich Client task.

Context Menu

In a smart device, context menus are used to perform actions that are usually considered to be general actions. In some cases there is a need to have additional options in a specific program or part of a program. Context menus are used to provide the additional options. On a desktop computer, context menus are invoked by clicking the right mouse button. On smart devices, they are activated differently. You will see how after seeing an example.

For mobile devices it is only possible to define a context menu for forms and not for controls.

In the **List of Orders** program, you added buttons for adding an order, printing an invoice and so on. To print an invoice, the screen design displays the print icon. However, it is not clear that the print button will print an invoice and the end user may think that it's for printing out the list of orders. A context menu will avoid this issue.

On iOS devices, if the form has a context menu, a button is added to the right corner of the form.

As an example you will first see how to print an invoice from the context menu.

1. Open the Menu repository.
2. Add a line and in the **Menu Name** column, type **Orders context menu**.
3. Zoom to the Menu Definition repository and add a line.

You can define a menu entry to call a specific program or to raise an event. It is good practice to use events.

4. From the **Entry Type** column, select **Event**.
5. In the **Entry Text** column, type **Print Invoice**.
6. From the **Menu Params** column, zoom to the **Event** dialog box.
7. From the **Event type** field, select **User**.
8. From the **Event** field, zoom to the Event list.
9. Select the **Print** event.
10. Click **OK**.
11. Close the Menu Definition repository and the Menu repository.

Attaching a Context Menu to a Form

Now you will attach the **Orders context menu** context menu to the **List of Orders** form. This will make the context menu available anywhere in the **List of Orders** program.

1. Open the Program repository.
2. Zoom to the **List of Orders** program.
3. Open the Form Editor and park on the **List of Orders** form.
4. Open the form properties.
5. From the **Context Menu** entry, zoom to the Menu list.
6. Select the **Print Invoice** menu.

Handling the Print Event

Now you will create the logic unit that handles the **Print** event as you have done in earlier lessons.

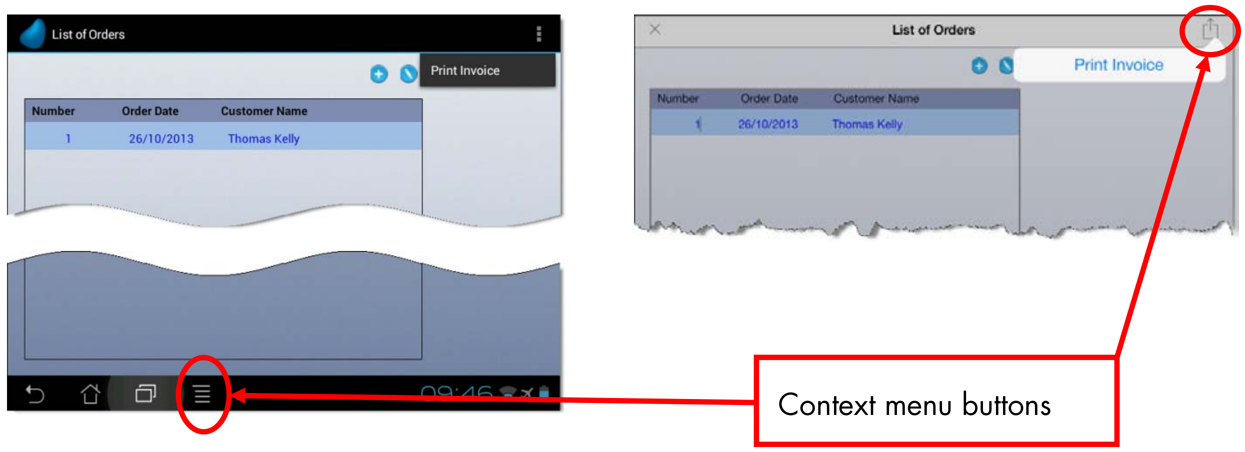
1. Open the Logic Editor and create an **Event** logic unit.
2. From the **Event** entry, zoom to the **Event** dialog box.
3. From the **Event type** field, select **User**.
4. From the **Event** field, zoom and select the **Print** event.
5. From within the **Event** logic unit, create a line.
6. Add a Virtual variable named **PDF Name** with an attribute of **Alpha** and a length of **255**.

During this course you have not created a variable within a logic unit but, as you can see, it is possible. The definition and value within the logic unit are only valid within the scope of the logic unit itself.

7. Add a detail line and call the **Invoices** program.
    a. Zoom into the **Arguments** and add a line.
    b. Zoom into the **Var** column and select the **Order_Number** variable.
8. Add a detail line and update the **PDF Name** variable with the expression: **ServerFileToClient('%TempDir%Invoice.pdf')**
9. Add a detail line and call the **Display PDF** program.
    a. Zoom into the **Arguments** and add a line.
    b. Zoom into the **Var** column and select the **PDF Name** variable.

You can now run the application and see how to use the context menu.

Android iOS



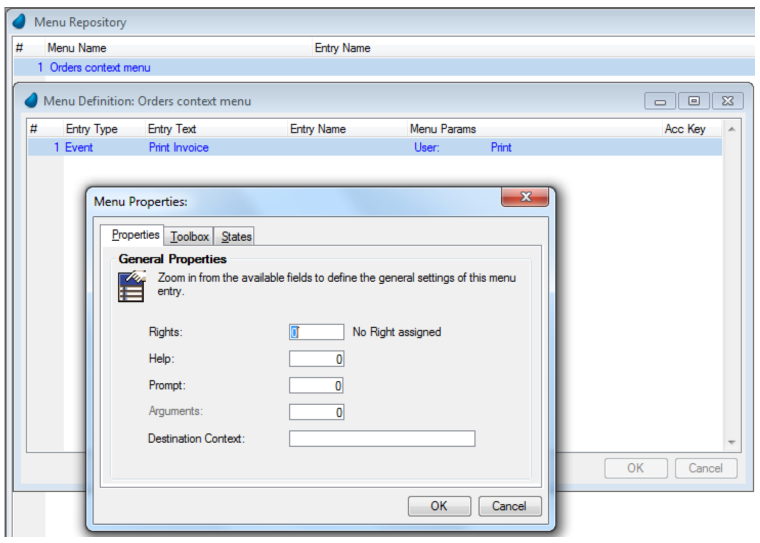
Menu Properties

The menu properties are divided into three tabs:

- Properties – General settings, such as assigning a right or a help screen to a menu
- Toolbox – Image and toolbar related settings
- States – Initial state of the menu entry (Checked, Visible, and Enabled)

As an example, you will provide an image for the Print Invoice context menu.

1. From the **Project** menu, select **Menus**.
2. Park on the **Orders context menu** menu and zoom to the Menu Definition repository. Park on the **Print Invoice** menu entry.
3. Open the **Menu Properties** dialog box by pressing **Alt+Enter**.



Menu Repository

#	Menu Name	Entry Name
1	Orders context menu	

Menu Definition: Orders context menu

#	Entry Type	Entry Text	Entry Name	Menu Params		Acc Key
1	Event	Print Invoice		User:	Print	

Menu Properties:

Properties | Toolbox | States

General Properties

Zoom in from the available fields to define the general settings of this menu entry.

Rights:	0	No Right assigned
Help:	0	
Prompt:	0	
Arguments:	0	
Destination Context:		

OK Cancel

4. Click the **Toolbox** tab.

5. From the **Image for** property, select **Menu**. In mobile applications there are no toolbars, so there is no need to set an image for the bar.
6. In the **Tool images** property, type: **%WorkingDir%images\PrintInvoice.png**.
7. Click **OK**. Close the Menu repository and save the changes.

Not every operating system currently supports images in mobile devices.

Application Menus

The application menu appears at the top of the screen and it is how end users run the programs in your application. As an example, in the Magic xpa Studio you have Project, Options, Debug, Tools and so on. The menu options change when you enter a program and you have a different set of menus. During runtime execution, you may have, for example, a menu for Orders, Reports, Tools and so on. This is how your user navigates through your application.

In the smart device world, application menus are no longer displayed, at least in the format you were used to. A lot of thought needs to be given to the application menu since this is the first item that the end user will see when loading your application. You need to think about the methodology of what you display. Remember that in the smart device environment, what you knew as application menus are now no longer visible throughout the application.

Button-Driven

One option is to have a set of push buttons, each loading a different program. In this case you could have a program called MyMenu that is the program initially executed and this program displays buttons such as Orders, Products, Reports and Tools.

This is simple to implement and provides you with a simple solution. Once the user taps a button, the program is loaded and when it is closed, it returns to the initial program.

Context Menu

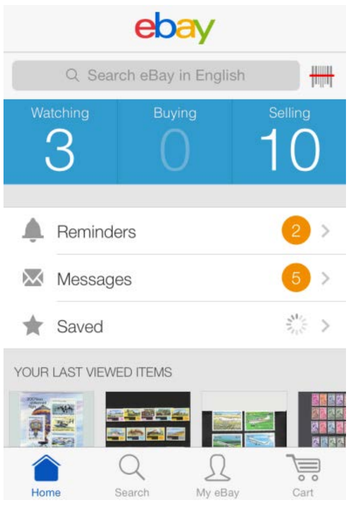
You already know how to add a context menu. Instead of having buttons or in addition to buttons, you could provide a context menu and display that context menu on each form.

Content-Driven

Many applications prefer to have their initial pages displaying content with little or no navigation. For example, you can open your email client. If you have not configured more than one email, it immediately shows you your emails. This is content. If you have more than one mailbox, in iOS you will see a list of the mailboxes with the number of unread emails listed. This is a page with statistics.

In iOS, the Notes application shows you the last place you were currently parked on; if it was in the middle of a note, it will show you that note, if not, it will display the list of notes.

For example, when you view the eBay application, as you can see in the image on the right, they show you a page of statistics, such as what you are watching, buying or selling in one row and in a list format, they show reminders, messages and saved items. At the bottom, they display the navigation functions. A lot of thought was provided to preparing the correct methodology for the initial program.



Course Menu

There is no correct way as to what to display in the course scenario; it depends on your company's methodology. You could use any one of the scenarios discussed. If you want to display content, you could show a list of orders or you could show statistics.

Summary

Menus are useful when you want to enable the end user to load other programs from the current program. You can use push buttons to perform similar actions, but that often takes up screen real estate that on smaller devices is important.

The context menu provides you with a useful way of providing more options without taking up place on the screen.

You learned about the initial program and that special thought needs to be given to what to display when your end users load your application. When this course was being written, the trend of smartphone applications was to display minimal navigation options and display content instead. Content can be a list of orders, a list of people who are currently in the club or statistics, or any other option.

Lesson 19

Using the Device Functionality

The advantage of working with a mobile device is that you are able to use the device's special features such as the GPS and camera. You also can change between mobile phones and tablets and between Portrait and Landscape modes, thereby increasing what is viewed on the form.

This lesson covers various topics including:

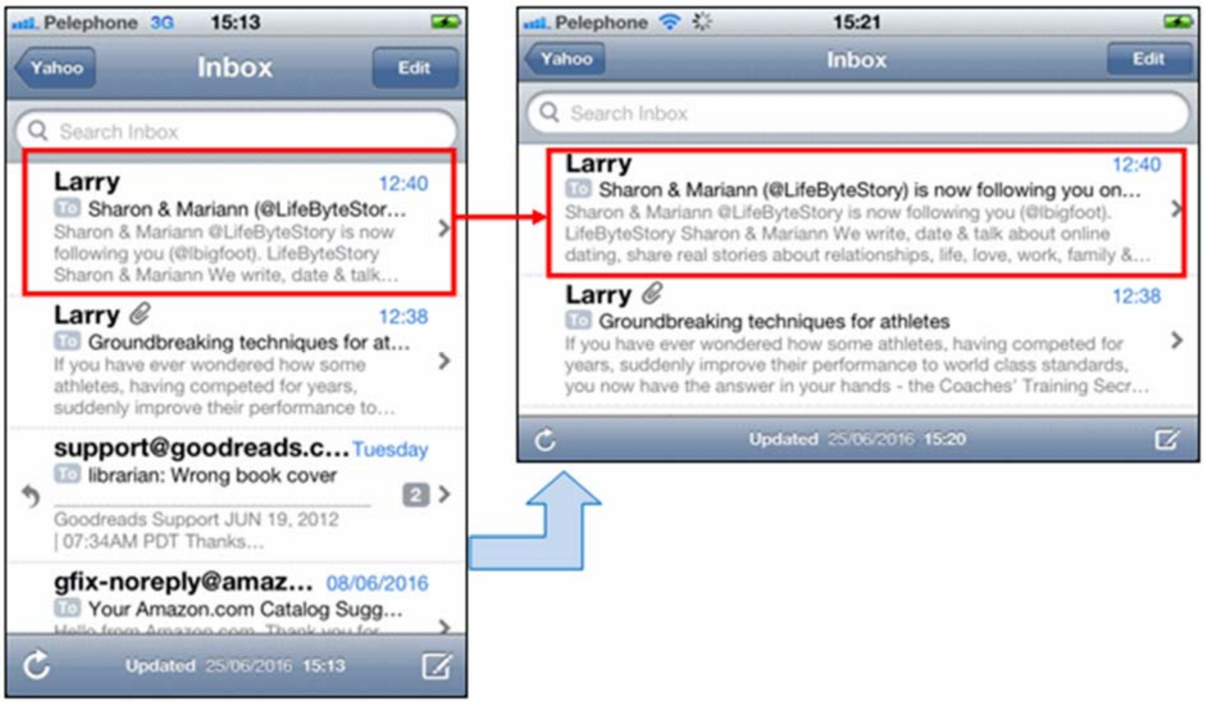
- Querying device characteristics
- Using the GPS and camera
- Mail, SMS, HTTP and other apps

Fetching the Device Orientation

During this course you used your tablet in Portrait mode. Turning it sideways turns it into Landscape mode and you immediately have more screen real-estate to be able to display more data.

Landscape Mode

When you rotate your device to Landscape mode, you are able to view more information on each line. When you use placement on the table you are able to view more of the text. For example, have a look at the email list below.



Pelephone 3G 15:13
Yahoo Inbox Edit
Search Inbox
Larry 12:40
To Sharon & Mariann (@LifeByteStor...
Sharon & Mariann @LifeByteStory is now following you (@lbigfoot). LifeByteStory Sharon & Mariann We write, date & talk...
Larry 12:38
To Groundbreaking techniques for at...
If you have ever wondered how some athletes, having competed for years, suddenly improve their performance to...
support@goodreads.c... Tuesday
To librarian: Wrong book cover 2
Goodreads Support JUN 19, 2012 | 07:34AM PDT Thanks...
gfix-noreply@amaz... 08/06/2016
To Your Amazon.com Catalog Sugg...
Hello from Amazon.com. Thank you for...
Updated 25/06/2016 15:13

Pelephone 15:21
Yahoo Inbox Edit
Search Inbox
Larry 12:40
To Sharon & Mariann (@LifeByteStory) is now following you on...
Sharon & Mariann @LifeByteStory is now following you (@lbigfoot). LifeByteStory Sharon & Mariann We write, date & talk about online dating, share real stories about relationships, life, love, work, family &...
Larry 12:38
To Groundbreaking techniques for athletes
If you have ever wondered how some athletes, having competed for years, suddenly improve their performance to world class standards, you now have the answer in your hands - the Coaches' Training Secr...
Updated 25/06/2016 15:20

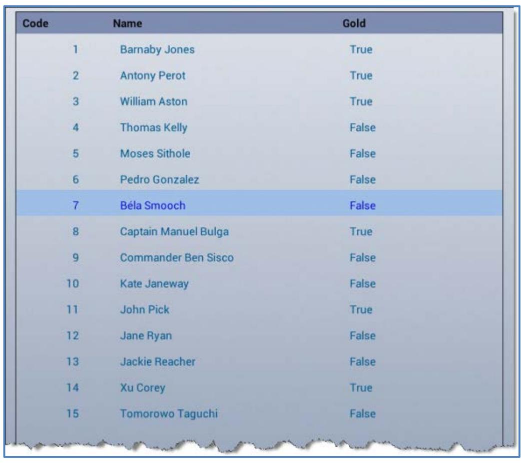
You also have the option of displaying more information on the table. Remember that you can use the **ClientOSEnvGet ('device_orientation')** to know whether you are currently in Portrait or Landscape mode.

In this exercise, you will modify the **Customers - Line mode** program so that if the device is in Landscape mode, it will also display the full address of the customer since there is more room for display.

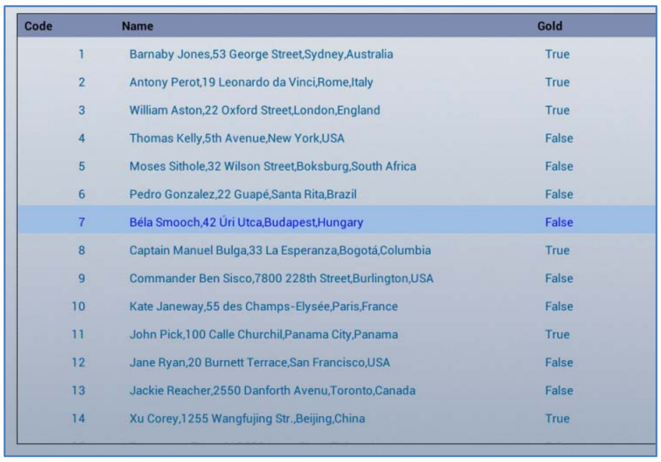
1. Zoom into the **Customers - Line mode** program.
2. Zoom into the Form Editor.

What you are going to do in this example is to provide an expression for the **Data** property of the **Customer Name** Edit control such that when the device is in Portrait mode, only the name will be displayed, but when it is in Landscape mode, the name and full address will be displayed. So you need to retrieve the device's orientation. Magic xpa provides a function that returns the specific environment setting: **ClientOSEnvGet**. Now you will see how to use it.

3. Select the **Customer Name** Edit control and enter the following expression in the **Data** property:
   If (ClientOSEnvGet ('device_orientation') = 'Portrait', Customer Name, Trim (Customer Name)&','&Trim (Address)&','&Trim(City)&','&Trim (Country))
4. Set the **Format** to **100**. You do not need to increase the table or the actual column. Placement will do this for you.



Code	Name	Gold
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False
11	John Pick	True
12	Jane Ryan	False
13	Jackie Reacher	False
14	Xu Corey	True
15	Tomorowo Taguchi	False



Code	Name	Gold
1	Barnaby Jones,53 George Street,Sydney,Australia	True
2	Antony Perot,19 Leonardo da Vinci,Rome,Italy	True
3	William Aston,22 Oxford Street,London,England	True
4	Thomas Kelly,5th Avenue,New York,USA	False
5	Moses Sithole,32 Wilson Street,Boksburg,South Africa	False
6	Pedro Gonzalez,22 Guapé,Santa Rita,Brazil	False
7	Béla Smooch,42 Úri Utca,Budapest,Hungary	False
8	Captain Manuel Bulga,33 La Esperanza,Bogotá,Columbia	True
9	Commander Ben Sisco,7800 228th Street,Burlington,USA	False
10	Kate Janeway,55 des Champs-Elysée,Paris,France	False
11	John Pick,100 Calle Churchil,Panama City,Panama	True
12	Jane Ryan,20 Burnett Terrace,San Francisco,USA	False
13	Jackie Reacher,2550 Danforth Avenu,Toronto,Canada	False
14	Xu Corey,1255 Wangfujing Str.,Beijing,China	True

When the device's orientation is changed, Magic xpa raises the **Window Resize** internal event. You can add your own code in the logic unit to update variables as a result of the current orientation.

Here are some of the keywords that can be used within the ClientOSEnvGet function.

Keyword	Description
device_os	Returns the device's operating system
device_screen-width	Returns the device's screen width in Portrait mode in pixels
device_screen-height	Returns the device's screen height in Portrait mode in pixels
device_physical-width	Returns the device's physical screen width in portrait mode in inches
device_physical-height	Returns the device's physical screen height in portrait mode in inches
device_orientation	Returns the device's screen orientation
device_os-version	Returns the device's OS version number
device_model	Returns the device's model number or name
device_touch	Returns "1" if the device has a touch screen
temp	Returns the local cache folder on the device
device_location	Returns the current device's location
device_magic-version	Returns the RIA client version number
device_udf|getargs	Returns the query parameters when the application was launched from another application
device_udf|getpushid	Returns the device ID, which can be used for sending push notifications to the device
device_udf|my_string	Calls a user defined function
device_resource-folder	Returns the value of the resource folder

As an example of the use, you might decide that one feature is not available on iOS or Windows 10 Mobile and so you can use the **ClientOSEnvGet ('device_os') = 'Android'** expression.

Note that on Android devices, you can also use Java predefined keys (for example: ClientOSEnvGet ('java.io.tmpdir')) to get additional information.

Sometimes there are scenarios where you want to lock the orientation of the device such that when you run a certain program it always displays in Landscape mode. To do this, you can set the **Lock orientation** form property to **Landscape**.

Accessing the Camera

One of the useful features of a smartphone is the ability to take pictures and then to store them. As an example you will use the camera to take a picture of whatever you want and display the picture.

1. Zoom into the **Customers - Line mode** program.
2. Zoom into the Data View Editor and add a Virtual variable named **Picture Location** with an attribute of **Alpha** and a size of **250**.
3. Create a User event named **Take Photo**.
4. Zoom into the Form Editor.
5. At the top of the form, to the left, add a push button.
6. In the **Format** property, type **Take Photo** and raise the User event, **Take Photo**.

Now you need to handle the Take Photo event and initiate the camera. It is possible to initiate a camera from the device using the **ClientFileOpenDlg** function. This will enable the user to take a picture and select it.

1. Zoom into the Logic Editor
2. Add a logic unit for the **Take Photo** User event.
3. Create an Update operation with the **Picture Location** variable and enter the following expression: **ClientFileOpenDlg('','camera','','FALSE'LOG,'FALSE'LOG)**.

The full path name of the picture will be returned from the function.

Now you need to display the image. There are many ways to do this and, here, one method is described. You will display the image in a subtask.

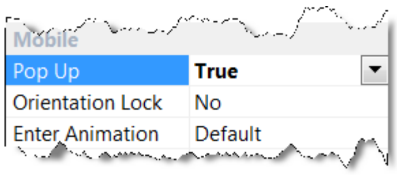
1. Create a subtask named **Display Photo**.
2. Add a **Parameter** named **P.Image Location** with an **Alpha** attribute and a size of **255**.
3. Zoom into the Form Editor.
4. Drop an Image control on the form.
5. Zoom from the **Data** property and select the **P.Image Location** parameter.
6. Set the **Load image from** property to **Client**.
7. Set the **Image Style** property to **Scaled to Fit**.
8. Return to the parent task and zoom into the **Take Photo** User event.
9. Add a line after the Update operation of the **Picture Location** variable.
10. Call the **Display Photo** subtask and pass the **Picture Location** variable as an argument.

You can now run the application.

In your program, you should upload this picture to the server using the **ClientFileToServer** function.

Displaying the image as a full screen is not always necessary. Sometimes you simply want to display it as a popup. You can do this by using additional information:

1. Zoom into the **Display Photo** subtask and then zoom into the Form Designer.
2. Set the **Pop Up** form property to **True**.


Mobile

Pop Up	True
Orientation Lock	No
Enter Animation	Default

Instead of using the **camera** value in the **ClientFileOpenDlg** function, you can use the **images** value and the operating system will display the photo gallery for you to select an image.

Telephone, SMS, Mail and HTTP

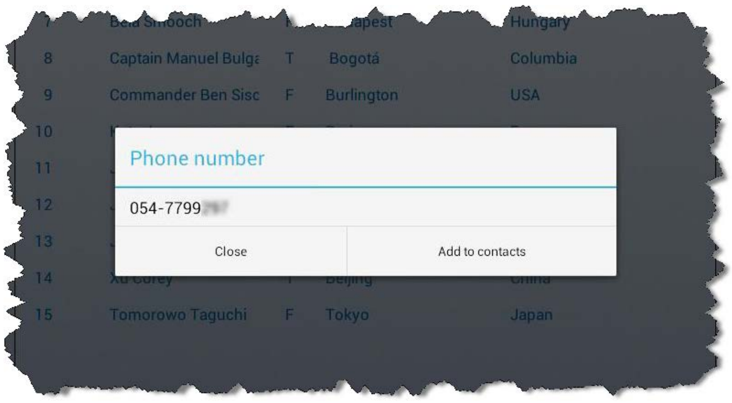
Sometimes you want to be able to call the number listed in your database or send an SMS (text message) or an email. This is simple to do. You will now see an example using the dialer.

1. Zoom into the **Customers – Line mode** program.
2. Create a User event named **Dial Out**.
3. Zoom into the Form Designer.
4. At the top of the form, add a push button.
5. In the **Format** property, type **Dial Out** and raise the User event, **Dial Out**.

Now you need to handle the **Dial Out** event and initiate the device caller.

6. Zoom into the Logic Editor
7. Add a logic unit for the **Dial Out** User event.
8. Add an Invoke operation and select **OS Cmd** from the combo box.
9. Set the expression to: **'tel:###-#######'** meaning a phone number, such as **'tel:+1-949-250-1718'**.
10. Set the **Execute on** property to **Client**.

When you run the application and tap the Dial Out button, the device will invoke the dialer of that device. Remember that each device and each operating system is different.


8 Captain Manuel Bulgz T Bogotá Columbia
9 Commander Ben Sisc F Burlington USA
Phone number
054-7799[redacted]
Close
Add to contacts
15 Tomorowo Taguchi F Tokyo Japan

To send an SMS, meaning a text message, you would simply change the tel to sms.

Sending an email is simple. It uses the **mailto** functionality of HTML. The syntax is:

mailto: email_address?subject=Comments&body=text

The only mandatory part is the email address. Here are some examples:

- mailto:mui@magicsoftware.com
- mailto:mui@magicsoftware.com&subject=Getting Started with Magic xpa 3.x for Mobile&body=We want to take this course

Accessing a website

You can also access a website using the Invoke OS Cmd operation by providing a full URL such as http://www.magicsoftware.com. In runtime, the website will be opened in the default browser of the device.

Using the GPS

The GPS or Global Positioning System has been around for longer than mobile devices have been with us, but today your mobile device uses GPS to be able to locate your device and offer you services in your vicinity, such as places to see, restaurants, or even how to get to a different location. It is up to you how to use the GPS options of your local device. Bear in mind that GPS applications often drain the device's battery life.

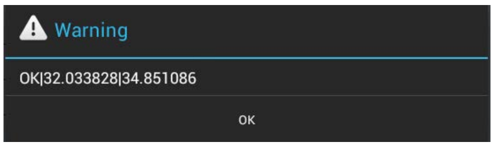
To get the device location, you use the ClientOSEnvGet function to query the current device location. This fetches the location using the internal or connected GPS device. **ClientOSEnvGet ('device_location')** returns the current device location, using any of the available location options, such as GPS, Network, and so on. The result is a string in the following format: **OK|Latitude|Longitude**, where **OK** is a fixed part for testing to see if a result was returned, and **Latitude** and **Longitude** are the coordinates of the current location. If a location could not be obtained, for any reason, an error message will be returned.

1. Zoom into the **Customers – Line mode** program.
2. Zoom into the Data View Editor and add a Virtual variable named **GPS Location** with an attribute of **Alpha** and a size of **50**.
3. Create a User event named **Use GPS**.
4. Zoom into the Form Designer.
5. At the top of the form, to the left, add a push button.
6. In the **Format** property, type **Use GPS** and raise the User event, **Use GPS**.

Now you need to handle the **Use GPS** event and initiate the internal GPS.

7. Zoom into the Logic Editor
8. Add a logic unit for the **Use GPS** User event.
9. Create an Update operation with the **GPS Location** variable and enter the following expression: **ClientOSEnvGet ('device_location')**.

Assume that the operation returned a value similar to the image below. You first need to check whether a GPS value was returned:



OK|32.033828|34.851086

10. Add a **Block If** operation and use the expression: **Left (Trim (GPS Location),2)='OK'**

The **Left** function takes X characters from the left, in your case, 2 characters.

Within the block you know that a valid GPS position was returned. You now need to retrieve the GPS data. Like everything in Magic xpa, there are many ways of doing this. In this case, you will first remove the first three characters, **OK|** and then you will replace the second pipe character (**|**) with a comma (,).

11. Add an Update operation with the **GPS Location** variable and update it with **Mid (GPS Location, 4, 50)**
    Remember that you were first introduced to the **Mid** expression in the lesson about the **Block operation**. The expression above takes the original text from the fourth character, thereby disregarding **OK|**.

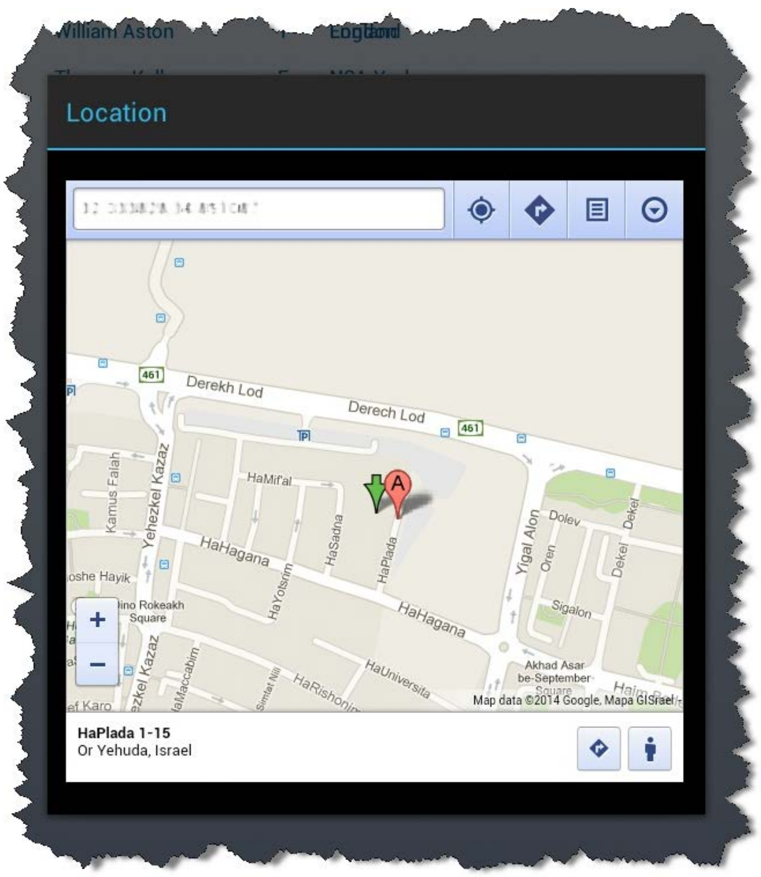
Now you need to replace the **|** character with a comma. To do this you will use the **RepStr** function, which replaces all occurrences of a string within text, with a different string. The syntax is:
RepStr (Text data, String to change, string to replace)

12. Add an Update operation with the **GPS Location** variable and update it with **RepStr (GPS Location, '|', ',')**

These are the coordinates that you can now use in Google Maps. Of course you could have broken the string and fetched the longitude and the latitude values and carried out any manipulation you wanted. You can now display this in a number of methods. For this example you will use the Browser control in a subtask.

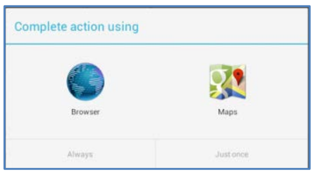
13. Create a subtask named **Display Location**.
14. Add a Parameter named **P.Current Location** with an **Alpha** attribute and a size of **50**.
15. Zoom into the Form Editor.
16. Set the **Pop Up** form property to **True**.
17. Drop a Browser control on the form.
18. Zoom from the **Data** property and add the following expression:
    'https://maps.google.com/?q='&Trim (P.Current Location).
19. Return to the parent task and zoom into the **Use GPS** User event.
20. Add a line before the end of the Block operation.
21. Call the **Display Location** subtask and pass the **GPS Location** variable as an argument.

You can now run the application.



After you have the GPS location, you can show a map with this location using a 3rd party application, for example by using the **Invoke OS Cmd** operation with **Execute On=Client** with any of the following expressions:

- You can directly enter: **https://maps.google.com/?q='&Trim (GPS Location)**. If the Google Maps application is installed on the device, you will be asked to decide in which application to display the location.
- You can use **'geo:'&Trim (GPS Location).** This will display each application that can display a geographical address.



- You can load a specific application directly by using its name, such as: **'waze://?q=London'** to open the Waze GPS application and navigate to London.

Location queries can sometimes take time to respond, because the GPS device is searching for satellites. During this time, the client is blocked, waiting for a response. You should make proper indications for the user that this is the situation. As an example, before invoking the GPS service, you can display a form such as "Searching for GPS location…"

Location queries have a built-in timeout of 20 seconds.

Multiple Forms

When your user uses a tablet or a smartphone, the amount of information that can be displayed on the device differs greatly. If all you are going to display is a list of names, then there is no difference in the screen design. However, if you want to display an order entry system using subforms or you want to have a dashboard showing everything that is happening in your office at the same time, then the size of the device matters. You also know that iOS and Android devices have different functionalities and the look and feel is different. Therefore, you need to decide whether you want to define different forms for different devices.

Magic xpa enables you to create different forms and then according to an expression, you can programmatically decide which one to display. Bear in mind that the form is loaded *when* the task is loaded and you are unable to change the form once the task is up and running.

Of course, the type of device does not change once you start using it and, therefore, you can use a Main Program variable to hold the type of device:

1. Zoom into the Main Program.
2. Add a Virtual variable with a **Logical** attribute and name it **Tablet?**

Now you need to update the variable with True when you are running on a tablet. The question is where?

Manager Task

You will often find that you need a manager task. This task is the first task that is called and performs some initiation functionality and then loads your program. This is the program that will be the initial program. You will see how to use this.

1. Add a program named **Start**.
2. Set the **Public name** to **RunMe** and check the **External** box.
3. Set the **Task type** to **Rich Client**, but uncheck the **Interactive** box. This will make this a type of Batch task, but it will run on the client.
4. Set the **End task condition** to **Yes** and the **Evaluate end task condition** to **After updating record**. If you do not do this, the task will never end.

A Rich Client task needs at least one variable:

5. Add a **Numeric** variable named **Screen Width** with a size of **3.2**.
6. Add a **Record Prefix** logic unit.
7. Create an Update operation with the **Screen Width** variable and enter the following expression:
   Val (ClientOSEnvGet ('device_physical-width'),'3.2')
   This fetches the device's physical width of the screen when the device is in Portrait mode.

How do you know whether you are using a smartphone or a tablet? There is no real way of knowing, but tablets are generally physically larger devices than phones and so you can use their physical dimensions. For this example assume that a smartphone has a physical width of up to 3.5".

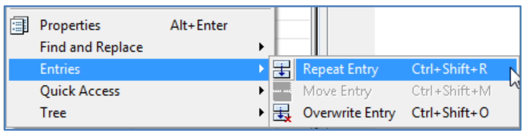
1. Update the **Tablet?** variable with **Screen Width > 3.5**.
2. Call the **All Products** program.

Now you will see how to create other forms. As an example, if the **All Products** program is launched from a smartphone, the list of products will be displayed, which is the way that the program currently behaves. However, if the same program is launched from a tablet, the product's image will be displayed to the right of the table.

1. Zoom into the **All Products** program.
2. Zoom into the Form Editor by pressing **Ctrl+3** or clicking the **Forms** tab.
3. Park on the **All Products** form.
4. Change the Form name to **All Products - Smart Phone**.

You are now going to copy this entry:

1. From the pulldown menu, select **Edit**, then **Entries** and then select **Repeat Entries**.
2. In the dialog box, enter **2**, which is the form that you are about to copy. This will duplicate the current entry.
3. Park on the last form.
4. Change the **Form name** to **All Products - Tablet**.
5. Zoom into the **All Products - Tablet** form.
6. Increase the width of the form so that you can add an Image control.
7. Drop an Image control to the right of the table.
8. Set the **Data** property to **'%WorkingDir%'&'products\'&Trim(Product name)&'.jpg'**.
9. Set the **Color** to **Text Caption**.
10. Set the **Image Style** to **Scaled to Fit**.



Remember that the table is set to increase by 100% if the size of the device increases; therefore, you need to adjust the image accordingly:

11. Zoom into the **Placement** property and set the **X** property to **100**.
12. Exit the Form Editor.
13. Open the **Task Properties** dialog box by pressing **Ctrl+P**.
14. Click the **Interface** tab.

The **Main Display** property governs which form will be displayed. Remember that this is calculated before the task is displayed.

15. Zoom into the **Main Display** property and then into the Expression Editor.

You are now going to enter an expression, so that if you are running on a tablet, the **All Products - Tablet** will be displayed. If not, you will display the **All Products – Smart Phone** form. Like with the **Task mode** property, when you use a form number inside any expression, to update it automatically, it is good practice to use the **FORM** literal. If you have an expression, such as **IF (A, 3, 2)**, this will work and is correct. However, if you add a new form you will need to make changes. By using the **FORM** literal, it is updated automatically. You can also select the form in the Expression Editor by clicking the icon.

16. Enter an expression, such as **IF (Tablet?, '3'FORM, '2'FORM)**.

Now when you run the application, you will see a different form on different platforms. You will see other uses of the manager program later in this course.

Summary

In this lesson you learned how to interact with the device using Magic xpa functions. The **ClientOSEnvGet** function has predefined keywords that enable you to use some of the device's options, such as the GPS and camera.

The built-in keywords enable you to fetch details about the device and display different information. For example, you may decide that a certain program is only available if you are using an iPhone.

Lesson 20

Offline Implementation

Magic xpa mobile applications work in a client server mode in which the client, the mobile device, is the interface that the user sees and contains part of the code. However, the server is at a remote location, has the database and handles any heavy workload. There is constant communication between the client and the server depending on the type of work involved. As you already know, when you call a new task, this process involves accessing the server to fetch the task details and then to display the task on the client.

What happens when you do not have internet communication? That is where offline programming is necessary. Magic xpa enables you to develop offline tasks. Offline programs allow users to continue to be productive in areas with intermittent, limited or unavailable internet connectivity. While working offline, data is stored locally on a local database, and periodically, when internet connectivity resumes, you can synchronize it back to the server.

This lesson covers various topics including:

- The Offline concept
- Local data sources
- Offline programs
- Synchronization of data
- Synchronization issues

Concept

When planning an offline implementation, it is important to understand the challenges and constraints that you need to overcome to enable applications to work completely offline without a server connection. Unlike a connected application, server connectivity is either non-existent or intermittent, and applications need to be adjusted to handle this scenario properly, without compromising usability and data integrity. As a programmer, you need to take into account the technical aspect of being disconnected, and the data consistency aspect, as follows:

- You can store a subset of relevant server data or client-only data on the client. You need to understand what data is constant data, such as a list of countries and which is not updated often, such as suppliers and items that are updated often, such as stock. However, even when dealing with stock, you may not need all of the data on the client. For example, a salesperson who deals with orders involving computer hardware only needs the products dealing with computers, while a salesperson who handles cleaning products only needs that subset. You can also have a scenario where the salespeople have access to the entire catalog but only current stock quantities is kept on the server and needs to be synchronized.
- On systems that require user authentication, consider storing user credentials securely on the client.
- Allow data entry on the client and update client data with server data for data consistency.
- Working under intermittent network connectivity (network disconnects, slow connections) while allowing uninterrupted operation and data consistency.
- Keeping application resources such as application metadata, image resources and so on, locally on the client, while allowing updates during connectivity periods.

The above challenges define an offline pattern that is different from patterns used when network connectivity is guaranteed, and require the developer to handle additional usage scenarios. Magic xpa provides tools and features that allow developers to tackle these challenges and provide a complete offline experience.

Some programs are totally client programs and some are server-based and others are both. An example of a client program can be something like a university student's lesson timetable. This timetable is only updated every semester and, therefore, the data can be kept on the client. An auction-based client depends entirely on server data and, therefore, you need an internet connection for this. A customer order can have elements of both server and client data. You will learn more about this later.

How Does It Work?

Unlike connected applications, offline applications are designed to work without (or with intermittent) network connectivity. This limitation defines a different execution flow for offline applications. Typical offline applications will work as follows:

- On first invocation, offline applications must download and synchronize all necessary resources required for the offline operation. These include application metadata, images, client-side data and so on. This requires an offline application to be connected at least once before it can work offline. If there was no initial connection, the programs will not work.
- For applications that require user authentication, user credentials should be securely stored on the client to allow for operation without server authentication. To ensure validity, such credentials should be re-checked when connected.
- Following initial invocation, all user interactions must be done using local resources only (local data, local images and so on). By using local resources exclusively, the application is guaranteed to work, regardless of the internet connectivity state and without requiring server access. All data updates should be stored locally on the local database.
- Periodically, at an application-dependent timing, the application must synchronize modified local data back to the server and download server data that was modified since the last synchronization. Remember that data objects can be updated by multiple clients and by the server simultaneously and therefore two clients may update the same data at the same time. You need to take this scenario into account.
- Since Magic xpa automatically synchronizes metadata objects while connected, an application that runs offline must be allowed to periodically synchronize changes to its metadata objects. Typically, if an internet connection is available on startup, metadata objects will be synchronized automatically. The developer should plan for allowing metadata updates to happen when the application changes.

The following sections describe, in more detail, how to implement each of the above, and the supporting Magic xpa features that enable each capability.

Terminology and Definitions

The following terminology is used:

- **Offline program** – A program that runs only on the client. This program will not access the server and cannot use server resources.
- **Local data source** – A data source pointing to a database that is stored locally on the client.

Local (Offline) Storage

The RIA client supports local data sources. Using a local data source, you can save data on the client and use it later on, mainly in Offline programs that do not access the server. Using local storage can also improve performance by locally caching data. The local database is defined by setting the **DBMS** column in the Database repository to **Local** in the same way that you defined one in Lesson 7, Viewing Data Source Content. A local DBMS database's data source is a client data source, and will be stored locally in the client cache.

To open the Database repository:

1. Verify that no projects are open.
2. From the **Options** menu, select **Settings** and then **Databases**.
3. Park on any line, such as the last line, and create (**F4**) a database entry.
4. In the **Name** column, type **Country Data**.
5. In the **Data Source Type** column, make sure that **DBMS** is selected.
6. Zoom from the **DBMS** column to open the **DBMS List**, park on the **Local** entry, and click the **Select** button (or press **Enter**).
7. The **Location** column shows the exact path to the database data. Set the **Location** to: **Countries.dat**.

The country and cities data do not change often and therefore there is little synchronization needed. To support the **Countries** and the **Cities** data sources on the client, you need to duplicate their entries in the Data repository:

8. Zoom into the **Getting Started** project.
9. Zoom into the Data repository.
10. Park on the last line and from the pulldown menu select the **Edit→Entries→Repeat Entries** option.
11. Zoom from the **From option** and select the **Countries** data source and then zoom from the **To option** and select the **Cities** data source.

You will see that the two entries are duplicated.

12. Park on the new **Countries** data source and change the name to **Local Countries**.
13. Zoom from the **Database** column and select **Country Data**.
14. Park on the new **Cities** data source and change the name to **Local Cities**.
15. Zoom from the **Database** column and select **Country Data**.

In this example the structure of the data sources is identical. Now you need to setup the program that synchronizes the data from the server to the client.

Synchronization Programs

You need to define programs for each data source that you want to synchronize. The following two programs need to be created for each data source:

- Server to Local synchronization – Update the local data source with server data.
- Local to Server sync synchronization – Update the server data source with local data.

Both of these programs are necessary if the update is bidirectional; meaning, the data can be updated on both the server data source and the local data source. If the update is done only in one location, then only one program is required. In the example that you are currently using, **Countries** and **Cities**, you only need to synchronize from the server to the client. This will be done by using a non-interactive, non-Offline Rich Client program with a server data source as the Main source.

1. Zoom into the Program repository and create a program named **Sync Countries from server**.
2. Set the **Task type** to **Rich Client**.
3. Uncheck the **Interactive** box to make this a non-interactive task.
4. Set the **End task condition** to **Yes**.
5. Set the **Evaluate condition** to **Before entering record**.
6. Zoom into the task and select **Countries** as the **Main source**.
7. Select all of the fields from the **Countries** data source. In this case there are only two fields.

Now you need to perform the synchronization. To do this, you will use a function named **DataViewToDataSource**. The **DataViewToDataSource** function takes the entire data view and copies in to a data source. You will now see the use of this function:

8. Zoom into the Logic Editor and add a **Task Suffix** logic unit.

Now you need to use the **DataViewToDataSource** function. The syntax of the function is:

DataViewToDataSource (<generation>, <variable names>, destination data source number, destination data source name, destination columns names)

The variable names are the names referred to in the Data View Editor in the **Name** column (**Field Description** property). These are case sensitive.

- The **generation** will be zero since you are referring to the current task.
- **Variable names** will be the names referred to in the **Data View name** column of the task, such as Country_Code, Country_Name.
- The **destination data source number** will be the entry of the Local Countries data source in the Data repository. In this case as with others, it is good practice to use the DSOURCE literal. You can select the data source from within the Expression Editor by click the ▦ icon.
- The **destination data source name** is the name of the table, if you do not use the number. You generally do not need to specify a name.
- The **destination columns names** are the names of the fields in the data source that the variable names will be copied to. The first name from the **Variable names** argument will be copied to the first entry here. If all the names are identical to the **Variable names**, you can use ''.

9. Add an Evaluate Expression operation:

   DataViewToDataSource (0, 'Country_Code,Country_Name','8'DSOURCE, '','')

   '8'DSOURCE is the Local Countries data source

- If the destination data source does not exist, it will be created.
- If a record exists in the destination data source, it will be updated.

That is the scope of the synchronization program from the server to the client. Now that you have the synchronization program, where do you call it from?

In the previous lesson you created a manager program named **Start**. You can use this program to run the synchronization programs.

10. Zoom into the program named **Start**.
11. Zoom into the **Task Suffix** logic unit and after the **Update** operation for the **Tablet?** variable, add a **Call** operation to the **Sync Countries from server** program.

If you now run the application, the **Local countries** data source will be created on the client, but the **All Products** program will be displayed. You want to be able to display the list of countries using the local data on the client device.

> In Offline tasks, as opposed to non-Offline tasks, the Task Prefix logic unit executes on the client. Therefore, it can contain client-side logic.

Offline Programs

For a program to work without server access, it must be specifically defined as an **Offline** program. An Offline program is downloaded to the client automatically on initial connection and kept cached at the client. Unlike non-Offline programs that are validated against the server on each invocation, Offline programs are always loaded from the client cache and do not require any server connectivity.

1. Zoom into the Program repository and create a program named **Offline Countries**.
2. Set the **Task type** to **Rich Client**.

You define a Rich Client program as offline by checking the **Offline** check box in the **Task Properties** dialog box or the Program repository.

3. Check the **Offline** box to make this an Offline task. You can now only use local storage data sources.
4. Zoom into the task and select **Local Customers** as the **Main source**.
5. Select all the fields from the **Customers** data source. In this case there are only two fields.
6. Zoom into the Form Editor, add a Table control and add the **Customer_Code** and **Customer_Name** fields. Set the look and feel of the form as you have during previous lessons.

Now you can call this program from the **Start** program.

7. Zoom into the program named **Start**.
8. Zoom into the **Record Prefix** logic unit and change the **Call** operation to the **All Products** program, to call the program named **Offline Countries**.

You can now run the **Start** program and you will see the list of countries available on the local device.

Offline Task Limitations

The following limitations apply to Offline tasks:

- You cannot use server data sources in an Offline task.
- You cannot use server-side operations and expressions in an Offline task.
- Only Offline programs can be used in an Offline program's subform.
- Offline programs cannot be used in a non-Offline program's subform.
- Offline tasks can only have Offline Rich Client subtasks.
- When testing offline capabilities in the Studio, the server needs to run in Background deployment mode. If the application was started on the client without server access, and the access to the server then resumes, the application will not be able to access the server when the server is running in Online deployment mode.
- The Tree control and Frames forms are not supported in an Offline task.

Main Program Flow

Applications with Offline programs have a slightly modified startup and execution sequence for allowing offline startup while also supporting a connected startup.

- On startup:
  - If the client can connect to the server, the Main Program will run both on the server and on the client, so a context will be opened on the server ready to serve future requests.
  - If the client cannot connect to the server, the Main Program will run only on the client, without opening a context on the server. In future server access, the client will try again to connect to the server as detailed below.
- When closing the application:
  - If the application was connected to the server, it accesses the server to close the context.
  - If there is no connection to the server, but there is no server logic in the Main Program's Task Suffix logic unit, then the application will be closed on the client without showing an error. In this case, the context on the server will remain open until the context inactivity timeout is reached.

?

Have you noticed some issues that arose with this small example?

- Every time you run the application, the synchronization program will run and all country data will be copied from the server to the client. The **Countries** data source does not change much. How can you change this so that synchronization is only performed when needed?
- If you do make a change to the country data, such as adding a new country or renaming one, how do you update the server about only that change?

In actuality, both issues are the same issue but from different aspects.

Synchronization Issues

When dealing with a static data source such as the **Countries** data source, you very often want to simply copy the data source over once and then forget about it. In essence, all you need to do is to check whether there are records in the local data source and if there are records, then there is no need to copy the data source. A simple solution to handle this would be to perform a **Link Query** operation and if there are records in the local file, then you do not need to perform data synchronization. Actually, country names can change, such as in 2012 the *Somali Republic* changed its name to *The Federal Republic of Somalia*. Sometimes, new countries are created such as *South Sudan* in 2011. You still need a mechanism to be able to update records when needed.

Data synchronization between a server data source and a local (client-side) data source depends on identifying the modified records. Being able to properly keep track of changes requires making changes to both data structures and to the logic that displays and updates data. You need what is commonly known as an audit trail. In this section you will learn about one method of doing this by keeping a string in the format YYYYMMDDHHMMSS.

1. Zoom into the Model repository.
2. Add a **Field** model named **Timestamp** with an attribute of **Alpha** and a picture of **14**.
3. Zoom into the Data repository.
4. Park on the **Countries** data source.
5. Zoom into the Column repository and park on the last line, the **Country_Name** row.
6. Add a line and name the entry **Last Modified** and link it to the **Timestamp** model.
7. Press **Enter** and you will return to the Data repository.
8. If you press **Enter** or move to a different line, you will get a new dialog box, **Confirm Convert Operation**. Click **Yes**.
9. You do not need to create a backup, so you can ignore the next dialog box.
10. Click **OK** on the last dialog box, which asks you which index to use.

The operation above converts the actual data source to meet the new definition in which you added a field. This new field is also used for when you update records in the data source.

1. Park on the **Local Countries** data source.
2. Zoom into the Column repository and park on the last line, the **Country_Name** row.
3. Add a line and name the entry **Last Modified** and link it to the **Timestamp** model.
4. Press **Enter** and you will return to the Data repository.

Magic xpa does not convert this data source since it exists on the client. It will automatically be converted once the application loads in runtime.

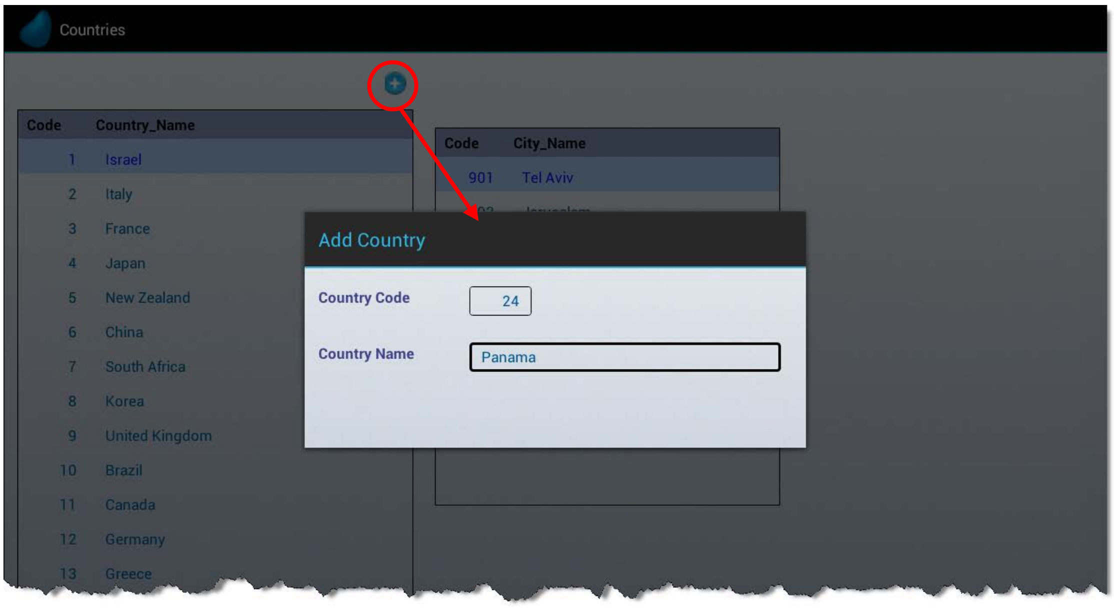
It is important that the names in both data sources are the same, including upper and lower case.

How do you use the new field?

You are going to add data to the **Countries** data source:

1. Zoom into the **Countries** program that you created in an earlier lesson.
2. Add a Rich Client subtask and name it **Add Country**.
3. Set the Initial mode to Create.
4. Select **Countries** as the Main source and select all three records.
5. Zoom into the form and display only the **Country_Code** and the **Country_Name** variables. You can set the **Pop Up** form property to **True.**
6. Zoom into the Logic Editor and add a **Record Suffix** logic unit.
7. Add an **Update Variable** operation and update the **Last Modified** variable with the following expression: **DStr (Date (),'YYYYMMDD')&TStr (Time(),'HHMMSS')**
   This will create a string concatenating the data and time.
8. Return to the parent program and add a User event named **Add Country**.
9. Zoom into the Logic Editor and add an **Event** logic unit for the **Add Country** User event.
10. In the **Add Country** logic unit, call the **Add Country** subtask.
11. Raise the **View Refresh** internal event.
12. Zoom into the Form Designer and place a **Button** control at the top of the form.
13. Zoom into the control properties. Set the **Model** to the **Image button** model.
14. Park on the **Image List file name** property and change the image to **Add.png**.
15. From the **Event Type** property, select **User**.
16. From the **Event** property, select **Add Customer**.
17. Click the **Fit Control Size** icon.

If you executed the application and ran this program (and not the **Start** program), you will see the modified program. As an example, add Panama as country number 24.


Countries

Code	Country_Name
1	Israel
2	Italy
3	France
4	Japan
5	New Zealand
6	China
7	South Africa
8	Korea
9	United Kingdom
10	Brazil
11	Canada
12	Germany
13	Greece

Code	City_Name
901	Tel Aviv

Add Country

Country Code: 24

Country Name: Panama

The timestamp has also been written to the **Countries** data source. This mechanism is valid also for updating the data in the data source and not only adding data. The issue now is how do you transfer *only* the modified record, meaning the newly added country. For this you need a new data source that will be saved locally:

1. Zoom into the Data repository and park on the last entry.
2. Add a new data source and name it **Local sync table**.
3. Set the **Database** column to **Country Data**. This is the local database.
4. Zoom into the Column repository and add a line.
5. Name the entry **Idx** with an attribute of **Numeric** and a picture of **2**.
6. Add another line, name the entry **Last Modified Country** and link it to the **Timestamp** model.

Now when you access the synchronization program, you need to first check this data source to see when the last update was made.

You are now going to make some changes to the **Start** program and to the **Sync Countries from server** program:

- From the **Start** program, you are going to fetch the last synchronization timestamp from the **Local sync table**.
- You will pass this as a parameter to the **Sync Countries from server** program so that you can perform a range starting from that timestamp.
- From the **Start** program, you will update the **Local sync table** with the current timestamp. This will then be the *last synchronization timestamp*.

1. Zoom into the **Start** program.
2. Add a Virtual variable named **Last Country timestamp** and link it to the **Timestamp** model.
3. Add a subtask named **Fetch last sync data** and uncheck the **Interactive** box.
4. Set the **End task condition** to **Yes** and the **Evaluate condition** to **After updating record**.
5. Zoom into the Data View Editor and set the Main source to the **Local sync table**.
6. Select all of the columns from this data source.
7. Add a **Record Suffix** logic unit, add an **Update** operation and update the **Last Country timestamp** variable from the parent task with the **Last Modified Country** column from this subtask.
8. Return to the parent task.
9. Zoom into the **Task Suffix** logic unit and after the **Update** operation for the **Tablet?** variable, add a **Call Operation** to the **Fetch last sync data** subtask, before the call to the **Sync Countries from server** program.

Now you are going to make some changes to the program that synchronizes the data so that it uses the timestamp information.

1. Zoom into the **Sync Countries from server** program.

2. Add the **Last modified** column from the **Countries** data source to the data view.
3. Add a **Parameter** named **P.Last Sync** with a model of **Timestamp**.
4. Park on the **Last modified** column and zoom into the **Range from** property, and set the following expression: **P.Last Sync**. This ranges all of the entries since the last synchronization.
5. Zoom into the Logic Editor and zoom into the expression you previously defined. You previously defined the expression as:
   DataViewToDataSource (0, 'Country_Code,Country_Name','8'DSOURCE, '','').
   You now need to add the **Last Modified** variable to the list **'Country_Code,Country_Name,Last Modified'** so that you have the same data on the server and the client.

You now need to update the **Local sync table** with the new timestamp so that the next time you perform synchronization it will take only the most recent changes.

1. Zoom into the **Start** program and zoom into the **Task Suffix** logic unit.
2. Park on the **Call** operation to the **Sync Countries from server** program.
3. Zoom into the **Arguments** property, add a line and select the **Last Country timestamp** variable.
4. Add a new line, add an **Update** operation and update **Last Country timestamp** with the following expression: **DStr (Date (),'YYYYMMDD')&TStr (Time(),'HHMMSS')**.
5. Add a subtask named **Update last sync data** and uncheck the **Interactive** box.
6. Set the **End task condition** to **Yes** and the **Evaluate condition** to **After updating record**.
7. Zoom into the Data View Editor and add a **Link Write** operation to the **Local sync table**. With a **Link Write** operation, if no record exists, one will be added.
8. Select the **Idx** column and set an expression of **1** for the **Locate from** and **Locate to** columns. Use the same expression for the **Init** expression.
9. Select the **Last Modified Country** column, since this is the variable you want to update.
10. Add a **Record Suffix** logic unit, add an **Update** operation and update the **Last Modified Country** column with the **Last Country timestamp** variable from the parent task.
11. Return to the parent task.
12. Add a **Call** operation after the **Update** operation to the **Last Country timestamp** variable and call the **Update last sync data** subtask.

You can now execute the application and call the **Start** program as you did in the previous lesson. When you execute the application, all country data is synchronized from the server to the client. You will see that the new country was added to the local database.

Server Access Failure

When you execute the application and try to perform a task that requires server access, such as data synchronization and there is no internet access, meaning no server, then all server programs will crash. This is because their metadata does not exist on the client. Only the Offline programs will be available. You will get an error message such as "Cannot access the server" and the end user cannot do anything about that. This includes the manager program. You defined the manager program named **Start** as a regular Rich Client program but if there is no server access, this program will not load and your application will not run.

1. Zoom into the **Start** program.
2. In the **Task Properties** dialog box, check the **Offline** box to make this an Offline program.

You now have another problem. The **Sync countries from server** program is a server-side program and you cannot call a server-side program from an Offline program.

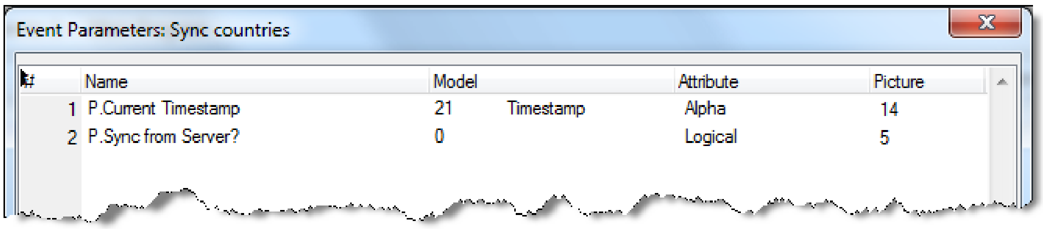
As you know, the Main Program is the parent task for the whole application. You can use this program to call server-side programs.

3. Zoom into the Main Program.
4. Add a **User event** named **Sync countries**.

The program you created was to synchronize the countries data from the server to the client. So you could have named this event **Sync countries from server**. However what if you also wanted to implement the opposite direction in which you synchronize the server with client data?

Spoiler: You are going to do that in the exercise. In this case, you can use the same event and pass it a parameter. Do you remember that the **Sync countries from server** program receives the current timestamp as a parameter? You can solve this in two ways, either you can define the **Last Country timestamp** variable as a Main Program variable or you can pass the value as a parameter to the **Sync countries** event. In this example, you will add it as a parameter to the event.

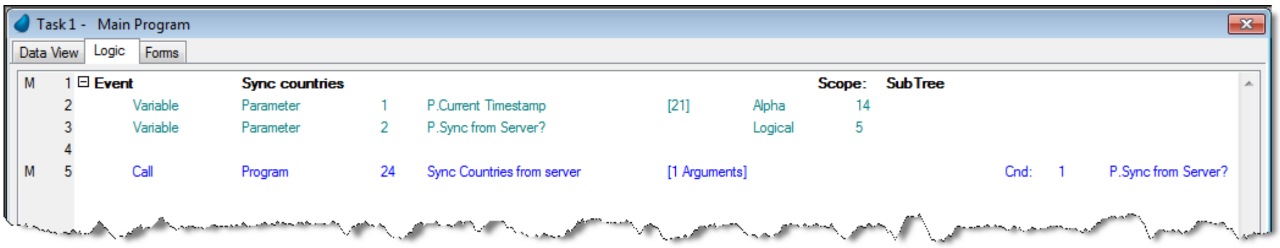
5. Zoom into the **Parameters** column.
6. Add a new line, set the **Name** to **P.Current Timestamp** and select the **Timestamp** model.
7. Add a new line, set the **Name** to **P.Sync from Server?** and set the **Attribute** to **Logical**. You will be passing a value of True when you want to synchronize from the server to the client and False when you synchronize from the client to the server.



Event Parameters: Sync countries

#	Name	Model		Attribute	Picture
1	P.Current Timestamp	21	Timestamp	Alpha	14
2	P.Sync from Server?	0		Logical	5

8. Zoom into the Logic Editor.
9. Create a logic unit for the **Sync countries** User event.
10. A dialog box appears asking: *Create Parameter variables to match parameters to the event?* Click **Yes**. The parameters you defined will be added to the logic unit.
11. Add a line and use a **Call** operation to call the **Sync countries from server** program.
12. Zoom into the **Arguments** property, add a line and select the **P.Current Timestamp** parameter.
13. Set the condition on the line to **P.Sync from Server?**.



Task 1 - Main Program

Data View | Logic | Forms

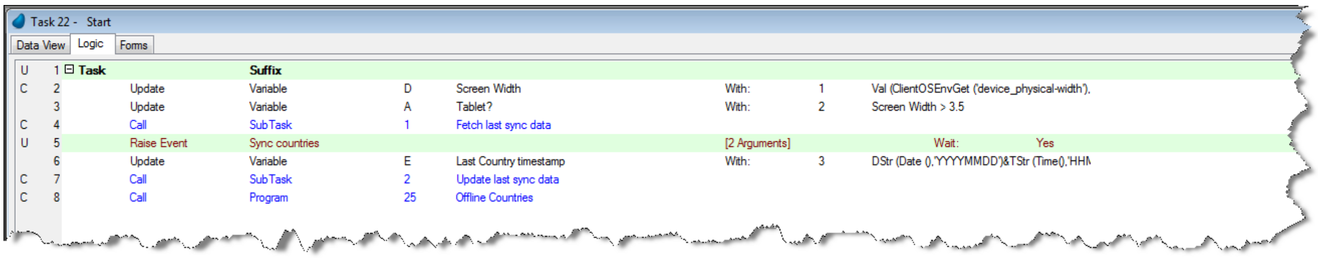
```
M  1 ⊟ Event      Sync countries                                              Scope:  SubTree
   2    Variable  Parameter  1   P.Current Timestamp       [21]           Alpha    14
   3    Variable  Parameter  2   P.Sync from Server?                      Logical  5
   4
M  5    Call      Program    24  Sync Countries from server  [1 Arguments]              Cnd:  1   P.Sync from Server?
```

14. Zoom into the **Start** program again and zoom into the Logic Editor.
15. Park on the **Call Program** operation to the **Sync Countries from server** program.

You can delete this line and add a new one or modify the line.

16. Instead of the **Call Program** operation, add a **Raise event** operation and select the **Sync countries** User event.
17. Zoom into the **Arguments** property, add a line and select the **Last country timestamp** variable to match the **P.Current Timestamp** parameter.

18. Add a second line, zoom into the **Exp** column and create an expression of True to match the **P.Sync from Server** parameter.



Task 22 - Start

Data View | Logic | Forms

U	1	⊟ **Task**		**Suffix**						
C	2		Update	Variable	D	Screen Width	With:	1	Val (ClientOSEnvGet ('device_physical-width'),	
	3		Update	Variable	A	Tablet?	With:	2	Screen Width > 3.5	
C	4		Call	SubTask	1	Fetch last sync data				
U	5		Raise Event	Sync countries			[2 Arguments]		Wait:	Yes
	6		Update	Variable	E	Last Country timestamp	With:	3	DStr (Date (),'YYYYMMDD')&TStr (Time(),'HHN	
C	7		Call	SubTask	2	Update last sync data				
C	8		Call	Program	25	Offline Countries				

Now you can run your application.

Network Unavailable Errors

When server access fails for any reason, such as when there is no network, the client device will display an error message:

- Magic xpa will display an error dialog box and the user will be prompted to retry or abort the operation. The client will retry the operation as many times as the user requests.
- All non-Offline programs running on the client will be terminated.

This means that if you try to perform data synchronization, it will also fail since these are non-Offline programs. You can decide whether or not to display the dialog box.

You can display the dialog box by using the **ClientSessionSet()** function, which can disable the dialog box for the entire session.

1. Zoom into the **Start** program.
2. Zoom into the **Task Suffix** logic unit.
3. Add a line after the **Update** operation for the **Tablet?** variable.
4. Add an **Evaluate Expression** operation and set the following expression:
   ClientSessionSet ('EnableCommunicationDialogs','FALSE'LOG)
   You are disabling the display of the dialog box for the entire session.

Even with this function, the server-side programs are still performed even though there is no server access. The only difference is that there is no error message. You can solve this by using another function that will return whether or not there is server access. The **ServerLastAccessStatus()** function will return an error code of zero if the last access to the server was successfully completed or a non-zero value if an error occurred.

1. Zoom into the **Start** program.
2. Zoom into the **Task Suffix** logic unit.
3. Add a line after the **Evaluate** expression operation where you used the **ClientSessionSet** function.

4. Add a **Block If** operation and set the condition to **ServerLastAccessStatus()=0**. This condition ensures that the access to the server will be made only if the client successfully accessed the server on the last attempt. If the client failed to connect to the server, then by having this condition, the client will not try (and fail) to reconnect to the server.
5. Delete the **Block End** operation.
6. Add a line before the **Call** operation to the **Offline Countries** program and set a **Block End** operation.

All the server-side programs will now be encased within the **Block If** operation. This allows the user to continue working.

Deleting a Record

In this lesson, so far, you only learned about adding or modifying records, yet often you need to delete records and synchronize those records between the client and the server. There are a few solutions to this; however, it is good practice to use a logical delete system. This system does not physically delete the record but indicates that the record has been deleted. To implement this method you would:

- Add a column to the server table named **Deleted** with an attribute of **Logical**.
- Add a column to the local table named **Deleted** with an attribute of **Logical**.
- When you delete a record, you do not physically delete the record but update the **Deleted** column with a value of True and updated the Timestamp with the current date and time.
- When you display the values of the table, you will always display only those rows where the **Deleted** variable is FALSE.

These records will also be synchronized when you perform the server synchronization. Because you are not physically deleting the records, the table will have redundant records. You can consider periodically doing a clean-up of all deleted records.

Offline Images

Images need to be transferred to the local device. These include images needed for the Offline programs, such as image buttons as well as data images, such as product images.

To make sure all images used in Offline programs are available when disconnected, you should copy, on initial startup, all server images that are used in Offline programs to the client using the **ServerFileToClient()** function. This function compares server and client file timestamps and will download new or updated server files to the client cache. Files that are unchanged will not be downloaded. This function also accepts a folder as a parameter, so you can use that to copy all content of the folder to the client.

Exercise

You will now practice some of the issues that you learned in this lesson.

1. Add an option in the **Offline Countries** program so that you are able to add a country to the local data.
2. Add a button so that in the **Add Country** task, the update will immediately be updated in the server table.
    - Take into account that there may be no server access, so you also need to implement client to server synchronization in the **Start** program.
3. Test this by adding country **#25** named **Mexico**.

Summary

If you decide to give offline capabilities to your application, you need to give thought as to how to implement the programs and of course which programs are offline and which are not.

You need to think about the following issues and take into account:

- Which programs are fully offline and which need to be online.
- An Offline program cannot call a server program and you need to use the Main Program to call a server program.
- Your implementation of the synchronization operations and how to have the same data on the server and the client. Remember that when you have multiple clients you may encounter a situation where the same record is updated by two different clients.

Synchronizing data involves the following steps:

- Fetch the last timestamp that a successful synchronization was performed.
- Use the DataViewToDataSource function to synchronize the data according to the timestamp.
- Update the timestamp with the current date and time.

Lesson 21

Best Practices

You now have a working knowledge of developing a Magic xpa program for use on a mobile device and you know how to differentiate between different operating systems. During this lesson, you will learn about some best practices to help cut down on programming time as well as enhance runtime performance.

This lesson covers various topics including:

- Using models
- Using logical names
- On-demand one-to-many forms
- Defining multiple main forms
- Code reuse
- Performance enhancements

Models

You learned about models in this course and you used them in various lessons. The use of models is optional, but using them will benefit you throughout the development and maintenance of your projects. Some advantages to using the Model repository are:

- **Time savings** for project development. Once you have created an object model, you no longer have to set the same property values for other objects of that class.
- **Ease of maintenance**. Once you have defined the properties associated to a specific model class, any modification to the model is automatically inherited by all of its associated objects.
- **Ensure matching of columns** in different tables for Field models. When columns are compared for link purposes, or passed as parameters, their attributes must match exactly. A good way to ensure that they will match is to define them with the same model.

Field Models

You used Field models to define a unified size and properties so that you did not need to define the same properties over and over again, such as with the Code model that you defined as a Numeric attribute of 9 digits. If you now decide that you want to define this as 5 digits, you only need to make the change in one place. You can also define the look and feel of this object on the form so that you define the colors and font of the object when it is dropped on the form. You can easily make changes to the font in one place and not in every place it is used throughout the application. Magic xpa will update all of the corresponding objects.

Form Models

You defined a model for a form so that when the model is used you have a unified look and feel for all forms without the need to remember what the location of the wallpaper is or what colors to use. It is then simple to change the name of the wallpaper.

Code Reuse

In the last lesson you synchronized data in two places, in the **Start** program and in the **Offline Countries > Add Country** program. In essence, you duplicated the code. You could design this better so that you can reuse the same logic. You could either use the Main Program **Sync Countries** event or create a **Sync Countries Manager** program that will encapsulate all of the logic in one place. This will cut down on development time and if you need to make changes, it will cut down on maintenance, since there will only be one section that needs to be dealt with.

Logical Names

During the course you often referred to a reference, such as **%WorkingDir%**, which is known as a logical name. For example, the form's wallpaper was set to **%WorkingDir%env\Wallpaper.jpg**. The **%WorkingDir%env** logical name is a location that changes from computer to computer and device to device. If you had defined the path of the wallpaper image as c:\Temp\Wallpaper.jpg, that may be valid on your specific computer but will not work on other computers and almost certainly not on a mobile device.

Logical names help in writing portable applications. The Logical Names facility allows development of applications without any explicit relation to physical storage media or operating system naming conventions. Magic xpa achieves such portability by translating project logical names at runtime, according to the Logical Name repository used as the translation repository of the installation. A logical name can be used whenever a file name is to be used.

When you used image buttons during the course you referred to the location **%WorkingDir%images** and added the **\Name.png** for every image. It would be a much simpler issue to remember if you had a logical name setup named **buttons** that points to **%WorkingDir%images** so that within the form you could simply use **%buttons%Add.png**. By using this method, if you decide to clean up the folders and move the button images to a different location, you only have to make a change to the Logical Names repository. This will make things easy if you decide to move to a Unix facility in which the slash is in the opposite direction. Bear in mind that when you use the ServerFileToClient () function you will now need to use **ServerFileToClient ('%buttons%')**.

Internal Logical Names

The working directory (the directory of the main project) and the engine's main directory are retrievable using the following reserved internal logical names:

- **%EngineDir%** – This logical name will be translated to the path of the Magic xpa engine.
- **%WorkingDir%** – This logical name will be translated to the path of the working directory.
- **%TempDir%** – This logical name will be translated to the path of the system's temporary directory.

One-to-Many Forms

You learned about using the Subform control to display a one-to-many operation, such as a country and the cities attached to that country or an order in which you have the header and the lines. In the example where you display the list of countries and in the subform you displayed the list of cities, remember that when you move through the list of countries, you are displaying the cities. This invokes server access to fetch the range of cities and then display them.

In this case you can decide whether you need to display the list of cities immediately in a subform, or to display them on demand by clicking a push button or by handling the **Tap** event. In the **List of Orders** program, you displayed the orders in a list but only displayed the order number, the order date and the customer name. When you tapped the order, you were shown the full order including the lines. In this case you were displaying the order details on demand. You could develop this further and display the order lines by tapping a button. This is all part of your own screen design.

Also remember that this will depend on your device. If you have a large device, such as an iPad, you may decide that you want to use subforms, but when you use an iPhone, you may want to implement the on-demand system.

Defining Different Forms

Different devices have different amounts of information that can be displayed. As previously mentioned, an iPad and an iPhone cannot display the same amount of information and an iOS machine and an Android machine have different displays. Therefore, what works on one device will work differently on another, or not at all.

In an earlier lesson, you learned how to differentiate between a smartphone and a tablet. You may decide that you also want to differentiate between iOS and Android. You can consider preparing different forms for different scenarios.

Remember to use the **Mobile Form Preview** pane to help you develop your applications for various platforms.

Manager Program

During this course you defined a manager program, which was the first program you executed. This program enabled you to perform certain session operations so that you could ensure ease of use during the rest of the session. This is where, for example, you fetched the device characteristics, such as what the operating system is or whether you are using a smartphone or a tablet. You also used this same program for synchronization of data between the client and the server and vice versa. You can use the same program to transfer all of the task images from the server to the client so that they are there when necessary and will not be transferred on demand. You could also transfer all of the product images.

Performance Enhancements

It is always a good idea to take into account the issue of performance. During this course you implemented some of the issues while progressing through the various lessons.

Remove redundant components and menu entries

When you learned about menus, you were asked to delete the **Default Pulldown Menu**. This is important for mobile devices since, by default, a new application is created with a default pulldown menu. This is not used in mobile applications and should be removed.

You did not learn about components during this course because this is an advanced issue, but when a new application is created, Magic xpa automatically adds a reference to the User Functionality component. This is invalid in a mobile device and should be removed. To do this:

1. Access the **Component Resource Repository** by clicking **Shift+F7** or by selecting it from the **Project** menu.
2. Park on the **UserFunctionality** line and delete the line by pressing **F3**.
3. Press **Enter** to exit the repository.

Minimize calls to the server

Group Server-side operations together

A logic unit can have both server-side operations and client-side operations. If you group the server-side operations together, you can minimize the server access so that instead of going back and forth to the server, all operations are executed and then control is returned to the client.

Avoid redundant calls to the server

In Offline programs, if you have no server access there is no point in calling the server program. Magic xpa will attempt to access the server and will fail and while this is happening the end user will be waiting. You can condition server access programs with the following condition: **ServerLastAccessStatus()=0**.

This condition will ensure that the access to the server will be made only if the client successfully accessed the server on the last attempt.

Therefore, if the client failed to connect to the server on the last attempt made, then by having this condition, the client will not waste time trying to reconnect to the server to perform the operation.

Cache application resources

To make sure that images used in Offline programs are available when disconnected, you should copy, on initial startup, all server images that are used in Offline programs to the client using the ServerFileToClient() function.

To improve the performance of copying the files to the client, you should download a full folder and not several separate files.

The ServerFileToClient() function supports folders and wildcard characters.

Executing the function once for a folder with several files means that there will be one request to fetch the modification timestamp of all of the files and then consecutive requests for each file that was changed. This means that if the files were not changed, the second time the application is started, only one request will be made.

Executing the function once per file means that for each file there will be one request to fetch the modification timestamp and then a consecutive request to fetch the file if it was changed. This means that for the first time the application is started, there are more timestamp check requests as compared to using a folder. For the second time the application is started, if the files were not changed, there will be a request to fetch the timestamp for each file as compared to a single request when using a folder.

Avoid Record level transactions on non-interactive client processes

When you have a non-interactive process that updates a lot of records, it is advised to use a Task level transaction when possible.

Using a Record level transaction will open a transaction for each record, which can reduce the performance. In addition, when the task updates a server data source, a Record level transaction will access the server after each record in order to commit the transaction and, therefore, will reduce the performance even more.

Copy a set of records between the server and the client

If you need to copy several records from the server to the client or from the client to the server, it is best to do it using the DataviewToDataSource() function.

This function copies a set of records (according to range criteria) as a whole and not one by one, which results in improved performance.

UI Improvements

- Avoid using a transparent color on the Table control and controls attached to the table – Using a transparent color is not recommended since it will slow down the performance when scrolling a table.
- Avoid using an alternate color on the Table control – Using an alternate color is not recommended since it will slow down the performance when scrolling a table.
- Avoid using borders for controls placed on a Table control – On iOS devices, for controls on a table, the controls' border and especially the corner radius will slow the table's performance.

Summary

This lesson summarized the best practices and you have used most of them throughout this course. These ideas will aid you in creating better and more readable applications that will be easily maintainable.

Lesson 22

Customization and Installation

In previous lessons you used the MagicDev client for your tests and you used the environment that the installation process provided. In this lesson you will learn how to provide your own environment.

This lesson covers various topics including:

- The execution properties file
- Signing the keystore file
- Creating your own Android package file
- Adding your own code

Note: In this lesson you will only be practicing on an Android device. There are similar methods for creating iOS and Windows 10 Mobile customizations. See the *Magic xpa Help* for additional information.

Creating a Cabinet File

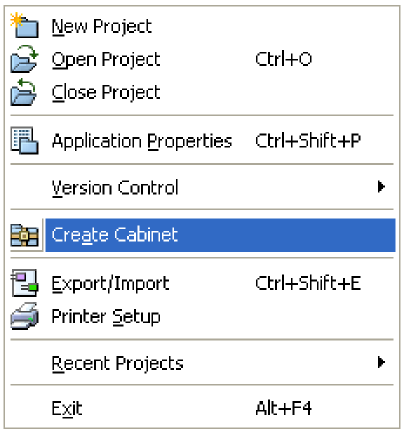
To execute a project in runtime, you need to create a project cabinet file.

A cabinet file is a file of a project in cabinet format, which can be used at runtime only, using the Magic xpa Runtime engine.

You cannot open a cabinet file using the Magic xpa Studio.

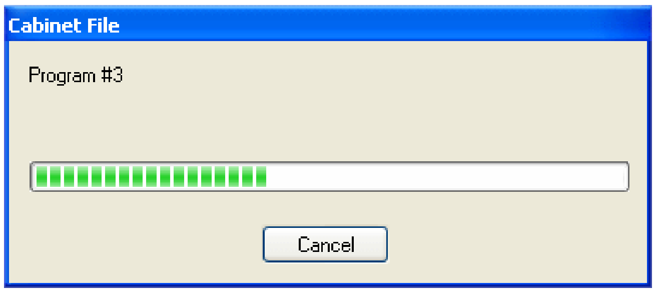
Now, you will create a cabinet file for your project.

1. Open your project.
2. From the **File** menu, select **Create Cabinet**.



The **Cabinet File** dialog box opens. You need to specify the cabinet file name and location in this dialog box.

3. Select the directory where the cabinet file will be saved.
   It is recommended to save the cabinet file (.ecf) in the same directory as the project file.
4. In the **File name** entry, make sure that it shows **Getting Started**.
   The default cabinet file name is the project name.
5. Click the **Save** button.



At this stage, the cabinet file is created.

If the cabinet file exists in the directory, you will be asked to confirm the overwriting of the existing file.

Setting Up the Server

For detailed information about deploying your applications, you'll find in the **Support** folder of Magic xpa the following concept paper:

- Deploying Rich Client Applications

In addition to the Broker middleware, Magic xpa 3.x supports an In-Memory Data Grid (IMDG). GigaSpaces' XAP in-memory computing technology is the middleware that implements Magic xpa's functionality on the In Memory Data Grid. To learn more about this, see the following concept paper:

- Deploying Applications on GigaSpaces

Running the application on the same machine that you developed the project can be misleading. The Runtime engine and the Studio engine share the same environment and the same configuration. Therefore, you can create a cabinet file (.ecf) and execute the same project using the Runtime engine.

Your final application is actually divided into two parts: the server and the client.

The server side runs on a remote machine, which is accessed via URLs. This is where the deployment engine and environment will be configured. This is also where the database will reside. The client is any client machine and nothing is deployed there.

All of the settings and configurations that you set in the **Magic.ini** file are set on the server computer. There are some other settings that need to be set in the **Magic.ini** file specifically for the server:

1. Set the **DeploymentMode** property to **B**. This ensures that the server will run in the background. Note that when an application runs in the background, there is no user interaction. As you learned when discussing reports, displaying a print preview or a print dialog box is meaningless.
2. Set the **ActivateRequestsServer** property to **Y**. This property informs the Magic xpa engine that it will run as an enterprise server and will receive requests through the Request broker. You will learn about this later in the lesson.
3. Set the **InternetDispatcherPath** property to **/MobileScripts/MGrqispi.dll**. This property is the HTTP requester and is set by the installation process. It defines the name and relative Web path to the Internet requester.

Setting Up the Web Server

Before installing Magic xpa for Mobile, you need to ensure that you have a Web server, such as IIS, up and running. When you install Magic xpa Deployment on the server, the installation procedure takes care of configuring the Web server to execute RIA applications. However, there are some configuration issues that you need to take into account.

The Rich Client application modules need to be deployed so that they can be executed on the native deployment environments.

Since the Rich Client environment is intended for use on the Internet, it is useful to understand the technology that is being used.

Rich Client Folders

The installation procedure creates three subfolders in the Magic xpa installation folder:

- **Scripts** – This folder holds the Internet requester files and RIA prerequisites files.
- **RIACache** – This folder holds the cache data for tasks, images and menus.
- **PublishedApplications** – This folder holds the application files that should be exposed, such as the HTML, manifest and RIA modules files.

Although these are installed under the installation directory, other locations can be used. This is managed by the Web aliases. The following Web aliases were configured by the course installation process and are required to deploy this course's Rich Client application:

- **MobileScripts** – This alias points to the **Scripts** directory of your Magic xpa installation.
- **MobilePublishedApplications** – This alias points to the **PublishedApplications** directory of your Magic xpa installation. You used this alias during the course.

These locations can be changed to meet your application requirements.

The Rich Client cache location may be tweaked to suit your needs. This is manipulated via a **Magic.ini** setting, **RIACacheFilesPath**. This defines the location where the Magic xpa server will write the Rich Client cache files.

Customizing the Application

Throughout this course, you used the icon provided by the installation, which includes the Magic icon and wallpaper. However, the client may already have an installation of another application or you may want to provide your own icon. Creating a custom application requires compilation using tools provided by the mobile devices.

You can change items such as:

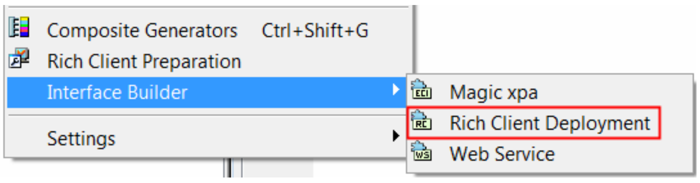
- Icon
- Startup splash screen
- Execution properties – You learned about this in the previous section.
- Client title
- Application version
- Package name

Here is an explanation about how to do this for an Android-based mobile application. For other mobile devices, please see the *Magic xpa Help*.

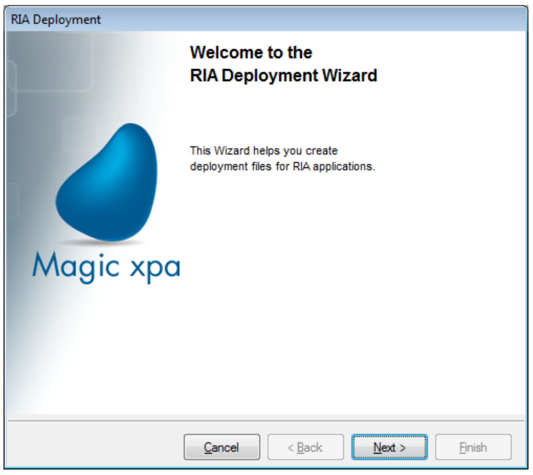
When you execute your application on a client, you need to deploy some modules on the client. This is the module that enables the logic that is defined on the client, such as displaying the form and client-side logic. It is also responsible for accessing the server when more information needs to be displayed.

Magic xpa includes a wizard to assist you in creating the deployment files.

1. To access the wizard, from the **Options** menu, click **Interface Builder** and then **Rich Client Deployment**.



From the **Welcome** screen, click **Next**.


RIA Deployment

Welcome to the RIA Deployment Wizard

This Wizard helps you create deployment files for RIA applications.

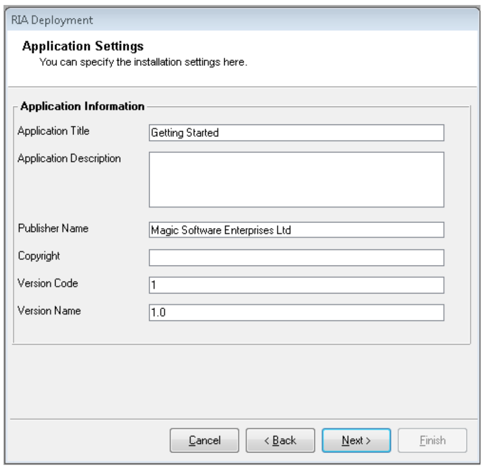
The **Available Configurations** screen will appear. This is a list of all the manifest files created from this application. The list should be empty.

2. Click **New** to create a new configuration.

Application Settings

The **Application Settings** screen contains information about the configuration and the project settings.

The following properties need to be defined:



RIA Deployment

Application Settings

You can specify the installation settings here.

Application Information

Application Title	Getting Started
Application Description	
Publisher Name	Magic Software Enterprises Ltd
Copyright	
Version Code	1
Version Name	1.0

- **Application Title** – This field has two purposes. The name provided here will be the name that you will see in the list of **Available Configurations**. This will also be the name of the mobile package that will be created.
- **Application Description** – This is a simple description of the application.
- **Publisher Name** – This is an important property and is part of the ClickOnce signature. It is the publisher name that the end user will see. This is relevant for desktop RIA deployment.
- **Copyright** – This is not applicable for mobile applications.
- **Version Code** – This is the internal code for your version.
- **Version Name** – This is the version that your user sees.

Once a Publisher Name has been entered, you will be able to progress to the next screen.

Server Information

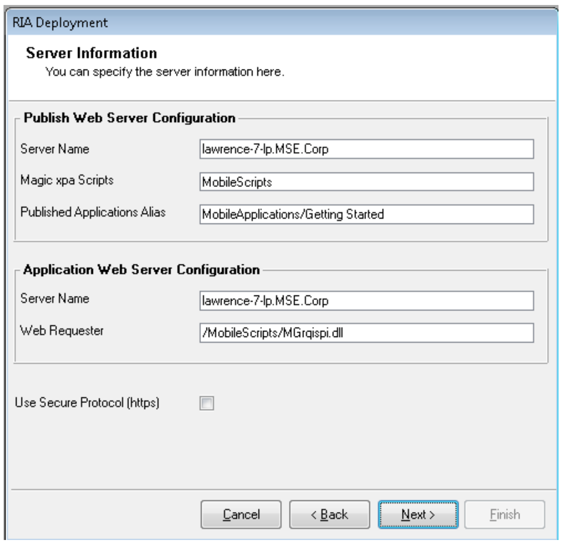
The server settings are important and need to be valid for each installation.

The deployment of the application can be done to two separate servers:

- Published server – Contains the RIA modules and the application manifest and HTML files.
- Application server – Contains the Web requester for the application.

If you move your installation to a different Web server address, you need to redefine the following settings:

Publish Web Server Configuration


RIA Deployment

Server Information

You can specify the server information here.

Publish Web Server Configuration

Server Name	lawrence-7-lp.MSE.Corp
Magic xpa Scripts	MobileScripts
Published Applications Alias	MobileApplications/Getting Started

Application Web Server Configuration

Server Name	lawrence-7-lp.MSE.Corp
Web Requester	/MobileScripts/MGrqispi.dll

Use Secure Protocol (https) ☐

- **Server Name** – The name or IP address of the server in which the files will be stored and accessed by the end users. This can also be in the format of **www.example.com**.
  This field is mandatory and cannot be left blank.
  If you update this field with **http://** or **https://**, it will be removed.
- **Magic xpa Scripts** – The Web alias that points to the **Scripts** directory.
- **Published Applications Alias** – The alias of the Published Applications folder.

Application Web Server Configuration

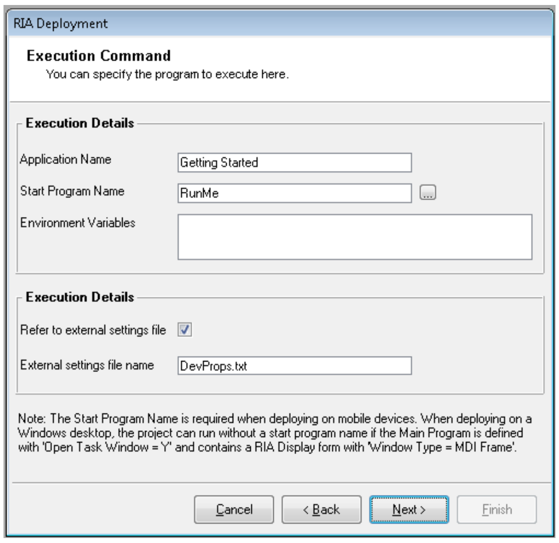
- **Server Name** – The name or IP address of the server in which the Rich Client application will be stored. This field is mandatory and cannot be left blank. This can also be in the format of **www.example.com**.
  The default value is the publish server name.
- **Web Requester** – The Web requester name to use. The default value is the **InternetDispatcherPath** from the **Magic.ini** file.
- **Use Secure Protocol (https)** – This check box determines whether to use the https or http protocol when running the application.

Execution Details

Now that you have entered all of the details about the server, you need to enter specific information about the application's execution.

The following properties need to be defined:

- **Application Name** – This is the name that the Web requester will use.
- **Start Program Name** – *This property is mandatory for mobile applications.* There is a selection list to select the initial program. The programs that will appear in this box have the following properties:
    - Type set to Rich Client
    - Public name is not blank
    - External check box is checked



RIA Deployment

Execution Command
You can specify the program to execute here.

Execution Details

Application Name	Getting Started
Start Program Name	RunMe
Environment Variables	

Execution Details

Refer to external settings file	✓
External settings file name	DevProps.txt

Note: The Start Program Name is required when deploying on mobile devices. When deploying on a Windows desktop, the project can run without a start program name if the Main Program is defined with 'Open Task Window = Y' and contains a RIA Display form with 'Window Type = MDI Frame'.

During the course, you were asked not to make changes to the name of the first program and leave it as **RunMe**. In this dialog box, you can make any changes that you want.

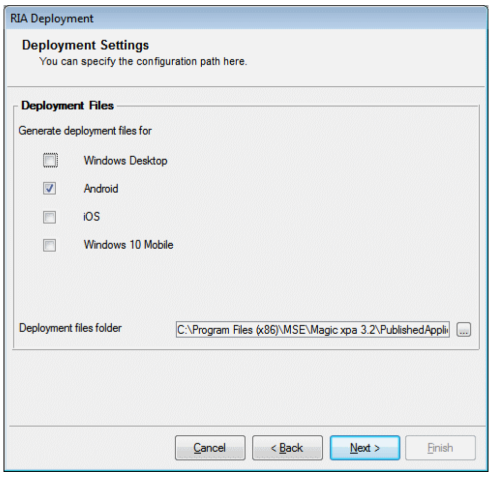
- **Environment Variables** – A comma-delimited list of the environment variables that the Rich Client will read from the client when it loads. These variables are not used in mobile applications.

At the beginning of the course you were asked to point to the location where the **DevProps.txt** file existed. In this dialog box you can change the name of the **DevProps.txt** file to a different name, such as **GettingStart.txt**.

You will learn how to manually change this file during this lesson.

Generate deployment files

In this dialog box you select which environment you are going to create files for. For this course, select **Android**.



RIA Deployment

Deployment Settings
You can specify the configuration path here.

Deployment Files

Generate deployment files for

- [ ] Windows Desktop
- [✓] Android
- [ ] iOS
- [ ] Windows 10 Mobile

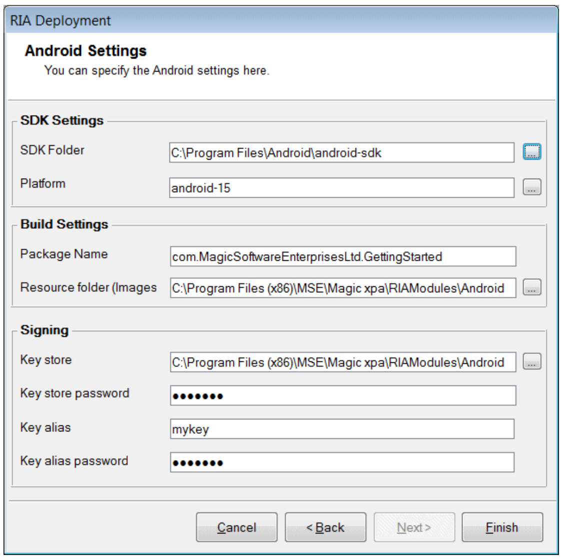
Deployment files folder: C:\Program Files (x86)\MSE\Magic xpa 3.2\PublishedAppli

Android Settings

This dialog box creates important customization settings. It writes the information to a file named **settings.properties**.

The following properties need to be defined:

- **SDK Folder** – The folder where you installed the Android SDK.
- **Platform** – The version in which the compilation will be made. This will be under the **Platform** folder.
- **Package Name** – This is taken from other information you have provided during this session. The package name must be in lower case and unique across all packages installed on the Android system. If you use upper case letters, the build script will convert them to lower case. The package name cannot include numbers and some reserved words such as 'new'.



Android Settings

You can specify the Android settings here.

SDK Settings

SDK Folder	C:\Program Files\Android\android-sdk
Platform	android-15

Build Settings

Package Name	com.MagicSoftwareEnterprisesLtd.GettingStarted
Resource folder (Images	C:\Program Files (x86)\MSE\Magic xpa\RIAModules\Android

Signing

Key store	C:\Program Files (x86)\MSE\Magic xpa\RIAModules\Android
Key store password	•••••••
Key alias	mykey
Key alias password	•••••••

- **Resource folder** – You can change icons and logos by copying them to the relevant location.
    - **Icon** – To change the icon, replace the icons' files in the **res\drawable-XXX** subfolders. You will need to provide icon files in all of the following sizes: 36x36, 48x48 and 72x72 pixels.

    - **Startup screen logo** – To change the startup screens, replace the **logo.png** files located in the **res\drawable-XXX** subfolders. You need to provide logo files in different sizes corresponding to the device's size.

Signing

To continue with the creation of the package, you need to have a Key Store file. The keystore file is a file that enables you to "sign" your APK file. To do this, you use your own keys and certificates. A sample keystore file is provided in the installation. You can use this by selecting the **<installdir>\Projects\Customization\Source\test.keystore** file, but leave all of the other properties as-is.

Uploading to Your Android Device

There are a few ways to install the APK client on the Android client:

- Run the APK directly on the Android device. For example, by receiving the APK file as an email attachment and clicking on it.
- Download the APK via the mobile device's browser.

- Download your application from the Android market.

In this course you will be using the browser method.

1. Navigate to the Android browser.
2. In the URL, enter: **http://server_name/MobileScripts/GettingStarted.apk**.
3. Go to the Downloads area in the emulator and your APK file will be there. Click it and follow the instructions.

After successfully installing the APK you will see a new application icon on your device.

Directly Executing on an Android Device

As mentioned in the early lessons, you can run a program on an Android device by toggling the **Execution on Android** icon from the **Debug** menu and then, when this entry is toggled:

- Executing a Rich Client program from the Studio (F7) will switch the engine to Runtime mode and launch the application on the mobile device or simulator using the selected program as the start program.
- Executing the project from the Studio (Ctrl+F7) will switch the engine to Runtime mode and launch the application on the mobile device or simulator using the start program that is defined in the **execution.properties** file.

For this to work, you need to:

1. Install the application on the mobile device or simulator. You can install the generic **MagicDev.apk**, which is available in the **RIAModules\Android** folder.
2. Connect the device to your network so that the application will be able to connect to the Magic xpa server to run the project.
3. From the **Android settings** menu, enable the **USB Debugging** option.
4. If you are using a device, connect the device to the PC using a USB cable. You may need to install the generic Google USB driver (from the Android SDK Manager, install the **Google USB Driver** package from the **Extras** folder) or the driver that came with the device.
5. Define the application package name in the **Run.bat** file located in the **RIAModules\Android** folder under the installation folder. To do this, open the file using a text editor and change the value of the **PackageName** variable to your package name, as it is defined in the **settings.properties** file. For example, when using the **MagicDev.apk**, which is available in the **RIAModules\Android** folder, you can set the name to **com.magicsoftware.magicdev**.

To check if your PC can access your device, open the command prompt, navigate to the **RIAModules\Utils\ADB** folder under the installation folder and execute the following command: **adb devices**.

You should see your device in the devices list.

The execution of the application on the Android device or simulator is done using the Android Debug Bridge (ADB) utility. For more information about this utility, please see the *Magic xpa Help*.

You can change the ADB commands by altering the **Run.bat** file located in the **RIAModules\Android** folder under the installation folder.

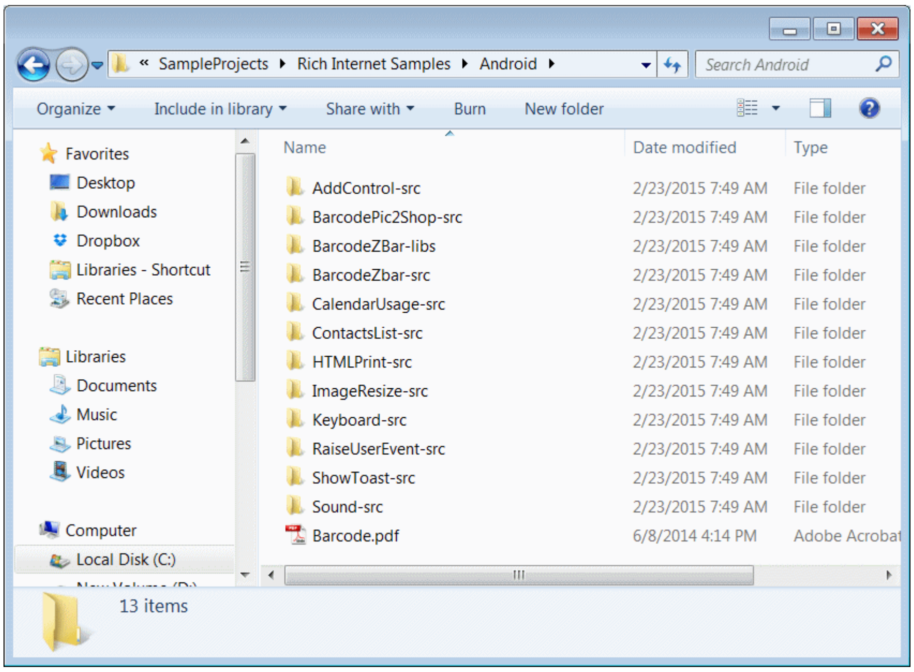
If the server on which Magic xpa is running is not defined in DNS, the Android mobile client will fail to access the Magic xpa server, since the device cannot translate the machine name to the IP address and you will get the following error: "Unable to resolve host name". If you encounter this error, you need to set the full URL with your machine's IP address in the **HTTP Requester** environment setting. For example: http://192.168.0.111/MagicScripts/MGrqispi.dll.

Usually, when using the Genymotion simulator, you can define the following value as the IP address: http://192.168.56.1/MagicScripts/MGrqispi.dll (192.168.56.1 is the default host IP address that the Genymotion VirtualBox supplies).

Using Native OS Code in Mobile Apps

Magic xpa provides an excellent platform for developing mobile applications, but sometimes you need features that are not provided out-of-the-box. You can add native OS code to a Magic xpa application. The code must be in the native language of the device such as Java for Android devices and Objective-C for iOS.

The installation of Magic xpa includes samples of using native OS code. The image below shows you the **Android** folder where you can find the samples. There is a similar folder for **iOS**.


« SampleProjects ▸ Rich Internet Samples ▸ Android ▸

Name	Date modified	Type
AddControl-src	2/23/2015 7:49 AM	File folder
BarcodePic2Shop-src	2/23/2015 7:49 AM	File folder
BarcodeZBar-libs	2/23/2015 7:49 AM	File folder
BarcodeZbar-src	2/23/2015 7:49 AM	File folder
CalendarUsage-src	2/23/2015 7:49 AM	File folder
ContactsList-src	2/23/2015 7:49 AM	File folder
HTMLPrint-src	2/23/2015 7:49 AM	File folder
ImageResize-src	2/23/2015 7:49 AM	File folder
Keyboard-src	2/23/2015 7:49 AM	File folder
RaiseUserEvent-src	2/23/2015 7:49 AM	File folder
ShowToast-src	2/23/2015 7:49 AM	File folder
Sound-src	2/23/2015 7:49 AM	File folder
Barcode.pdf	6/8/2014 4:14 PM	Adobe Acrobat

13 items

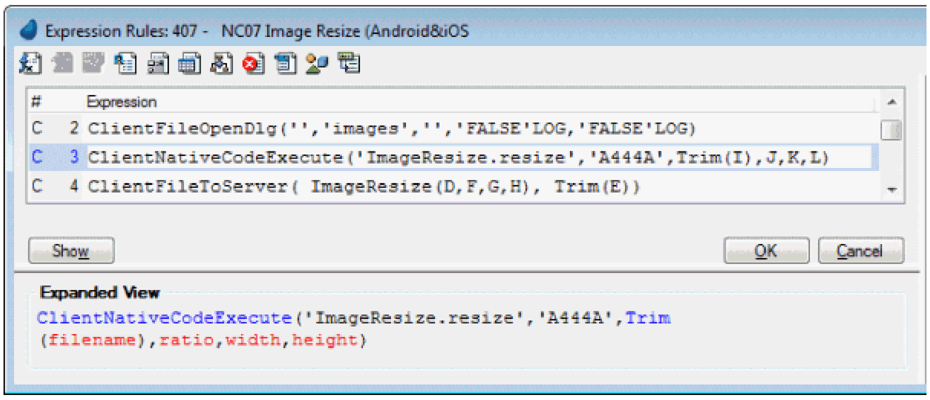
You can use native OS code in your application in two ways:

- Call from your Magic xpa application to native OS code
- Raise a Magic xpa user event from the native OS code

1. Call from your Magic xpa application to native OS code

You can do this in your application by evaluating the **ClientNativeCodeExecute** function with the name of the native code method and parameters.

So, for example, in the image below, the ClientNativeCodeExecute() function will execute the resize method in the **ImageResize** class in the mobile application code and pass it the arguments of **filename**, **ratio**, **width** and **height**.


Expression Rules: 407 - NC07 Image Resize (Android&iOS

#		Expression
C	2	ClientFileOpenDlg('','images','','FALSE'LOG,'FALSE'LOG)
C	3	ClientNativeCodeExecute('ImageResize.resize','A444A',Trim(I),J,K,L)
C	4	ClientFileToServer(ImageResize(D,F,G,H), Trim(E))

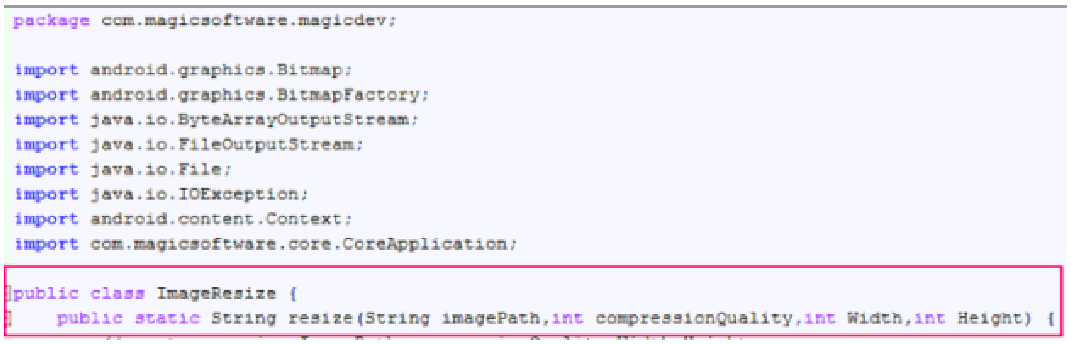
Expanded View

```
ClientNativeCodeExecute('ImageResize.resize','A444A',Trim
(filename),ratio,width,height)
```

The native code for this function looks as follows:

Android

The **ImageResize.java file** located in the **Rich Internet Samples\Android\ImageResize-src** folder:

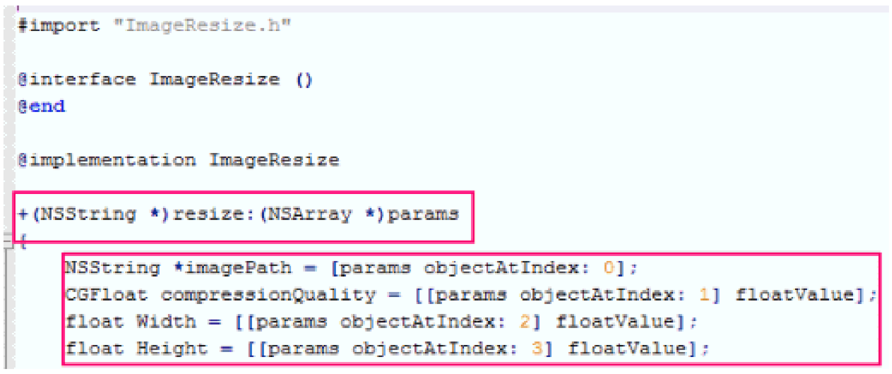


```
package com.magicsoftware.magicdev;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.File;
import java.io.IOException;
import android.content.Context;
import com.magicsoftware.core.CoreApplication;

public class ImageResize {
    public static String resize(String imagePath,int compressionQuality,int Width,int Height) {
```

iOS

For iOS, although you are sending several different parameters to the method, you need to define the called method with only one parameter of NSArray type. All of the Magic xpa parameters are automatically set into this array. Then you can retrieve the parameters from the array

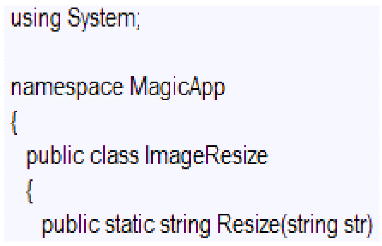
The **AppDelegate.mm** file located in the **RIAModules\iOS\Source\MagicApp** folder:



```
#import "ImageResize.h"

@interface ImageResize ()
@end

@implementation ImageResize

+(NSString *)resize:(NSArray *)params
{
    NSString *imagePath = [params objectAtIndex: 0];
    CGFloat compressionQuality = [[params objectAtIndex: 1] floatValue];
    float Width = [[params objectAtIndex: 2] floatValue];
    float Height = [[params objectAtIndex: 3] floatValue];
```

Windows 10 Mobile

The ImageResize.java file located in the Rich Internet **Samples\Windows10Mobile\ImageResize-cs** folder:



```
using System;

namespace MagicApp
{
  public class ImageResize
  {
    public static string Resize(string str)
```

2. Raise a Magic xpa user event from the native OS code

You can then raise a Magic xpa event from your code that will handled in your Magic xpa application. You do this by adding the following lines to your native code:

Android

1. Add the declaration: **import com.magicsoftware.core.CoreApplication;**
2. Raise the event:
   CoreApplication.getInstance().invokeUserEvent(event_name,param1,param2); where **event_name** is the user event name and **param1** and **param2** are the values that will be passed to the user event handler.

iOS

1. Add the declaration: **#import "Magicxpa.h"**
2. **Add an array that will hold all of the parameters' values, ending with a nil value:** **NSArray *params = [NSArray arrayWithObjects:param1, param2, nil];** where **param1** and **param2** are the values that will be passed to the user event handler.
3. Raise the event: **[Magicxpa invokeUserEvent:event_name Params:params];** where **event_name** is the user event name.

Windows 10 Mobile:

Raise the event: **com.magicsoftware.richclient.CoreMethodInvoker.InvokeUserEvent(event_name, param1, param2);** where **event_name** is the user event name and **param1** and **param2** are the values that will be passed to the user event handler.

Fonts

The Magic xpa mobile RIA client uses the same font table as other interfaces. Each device has a set of available fonts that are usually different from the fonts found on a Windows desktop and different from one another.

Since it is not possible to select mobile fonts using the Windows fonts dialog box, to enter these fonts into the font table, you need to directly edit the fonts table file (such as **fnt_rnt.eng**) using a text editor.

Here are some example font table entries:

- Android Droid Serif font: *Android font,Serif,14,0,0*
- Android Bold Droid Serif font: *Android bold font,Serif,14,0,0,Bold*
- iOS Helvetica font: *iOS font,Helvetica,14,0,0*
- iOS Helvetica font: *iOS bold font,Helvetica-Bold,14,0,0*

If the font defined in the font table is not found on the mobile device, the default font will be used with the size defined in the font table.

On Android devices you need to send the font family name and the style of the font you want. The system will find a matching font for you. For example, to use a Droid Serif font use: "Android Serif bold,Serif,12,0,0". You can see the font list in the **System/Fonts** folder in the device file system.

Specific Font files for each operating system

You learned that you can define specific forms for the different operating systems. You can also maintain multiple different font files for each mobile platform so that you can give it a native look and feel. If you want a different set of fonts for each platform, create a separate font file and put it in a subfolder with the name of the platform under the folder containing the defined font file.

For example: If the runtime font file is **Support\fnt_rnt.eng**, then you can:

- Put the Android font file under **Support\Android\fnt_rnt.eng**.
- Put the iOS font file under **Support\iOS\fnt_rnt.eng**.

At runtime, if it exists, the appropriate font file will be used automatically.

If you decide to use separate font entries for each device, you need to edit the fonts table manually using an external text editor such as Notepad. For example, Android uses a font known as **Sans**:

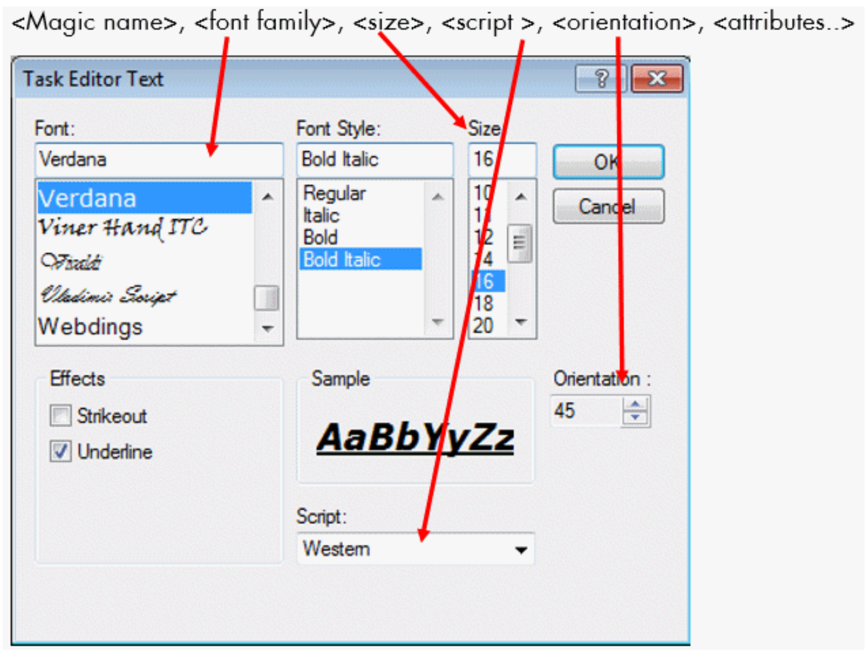
1. Open the Android **fnt_rnt.eng** file in a text editor.
2. Add the following line: **Android Edit Labels,sans,8,0,0,Bold**
3. Save the file.
4. Close the **Getting Started** project and reopen it so that the new font list will be reloaded.
5. Zoom into the **Select Customers** program.
6. Zoom into the Form Designer and for the two column headers, select the new font. Because the font is now bold, you need to increase the width.
7. Execute the project.

You will see that the label is now bold.

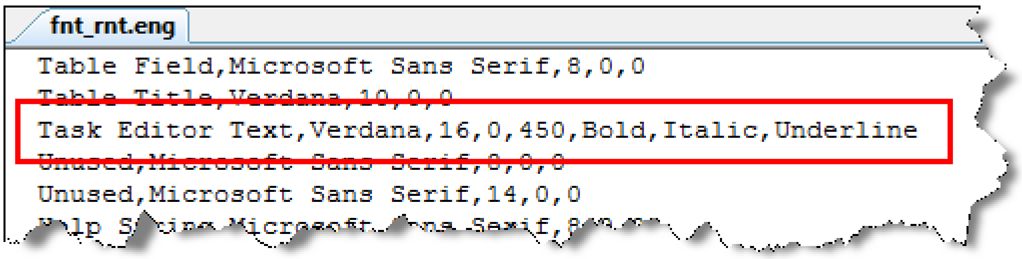
Adding a font manually

When you added the font manually, you added the line:
Android Edit Labels,sans,8,0,0,Bold to the Font list.

Each entry in the list is defined in the following way:



The attributes are the **Font Style** and the **Effects**. The dialog box above will be translated to the following entry in the Font list:


fnt_rnt.eng

```
Table Field,Microsoft Sans Serif,8,0,0
Table Title,Verdana,10,0,0
Task Editor Text,Verdana,16,0,450,Bold,Italic,Underline
Unused,Microsoft Sans Serif,8,0,0
Unused,Microsoft Sans Serif,14,0,0
Help String,Microsoft Sans Serif,8,0,0
```

If you use different fonts for different devices, you will need to use an expression for the font of a control. You can use the ClientOSEnvGet function in the expression:
IF (ClientOSEnvGet ('device_os')='android', '1','4')

You may decide that working with the CASE function will better suit your application.

Summary

In this lesson you learned how to customize the environment to suit your own needs. Remember that the steps to prepare your application are:

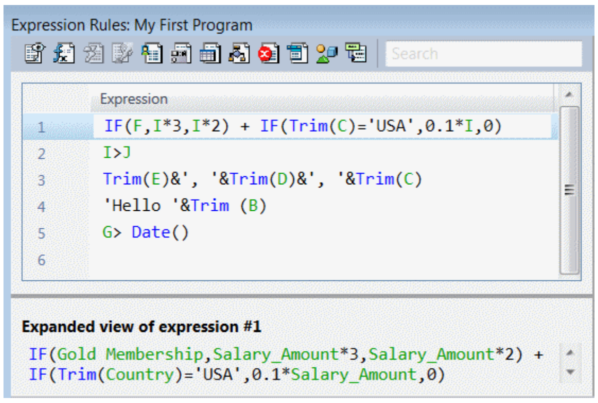
1. Create a cabinet file.
2. Prepare the icons and splash images.
3. Define the **settings.properties** file and the **execution.properties** file by using the builder or by modifying them manually.
4. Define the **DevProps.txt** file to suit your environment.
5. Get a keystore file. Check this link for more details: http://developer.android.com/tools/publishing/app-signing.html
6. Create the APK file.
7. Move the file to an IIS alias exposed folder, such as the scripts alias.
8. On your Android device, navigate to your browser and upload using the following path: **http://server_name/magic_scripts_alias/apkname**.

Solutions

Lesson 4 – Data Manipulation

The USA banks decided to give their clients a benefit. Each USA client will receive an additional credit amount, which is 10 percent of the client's salary.

1. In the Logic Editor, go to the **Update Variable** line and zoom into the **With** column.
2. Find the line with the IF function, which should look something like this: IF(F,I*3,I*2).
3. To this IF function add the following:
   + IF(Trim(C)='USA',0.1*I,0)
   Where **C** = **Country** and **I** = **Salary_Amount**
   Your expression should now look like this:
   IF(F,I*3,I*2) + IF(Trim(C)='USA',0.1*I,0)


Expression Rules: My First Program

```
Expression
1  IF(F,I*3,I*2) + IF(Trim(C)='USA',0.1*I,0)
2  I>J
3  Trim(E)&', '&Trim(D)&', '&Trim(C)
4  'Hello '&Trim (B)
5  G> Date()
6
```

Expanded view of expression #1

```
IF(Gold Membership,Salary_Amount*3,Salary_Amount*2) +
IF(Trim(Country)='USA',0.1*Salary_Amount,0)
```

To make your project friendlier, add a personal welcome announcement to the form.

4. Park on the form and press **Ctrl+A**. All of the controls on the form should now be marked.
5. Unmark the controls above the **Country** field, by pressing the **Ctrl** button and clicking on each control.
6. Move the other controls down.Play around with the dragging until the placement looks right to you.
7. From the Toolbox, drag an Edit control onto the form and place it beneath the **Customer_Name** Edit control.

You will now assign an expression to the Edit control that displays a concatenation of the word 'Hello' and a trimmed **Customer_Name**.

1. Expand the new Edit control's **Data** property zoom from the **Expression** line and open a new line.
2. Enter the following expression. Make sure that there is one space between the word Hello and the customer name: **'Hello '&Trim(B)**
   where **B** is the **Customer_Name** variable.

You will now move the Customer Address concatenated control that you added in the Customer Address example during the lesson to below the **Address** Edit control.

3. Move the controls around to make room below the **Address** Edit control.
4. Drag the **Customer Address** concatenated control from the bottom of the form and move it into place.

Add a validation to **My First Program** so that the end user will have to type in a **Customer_Code** before continuing to the next control.

5. Add a **Control Suffix** logic unit to the **Customer_Code** control.
6. Add a **Verify** operation in **Error** mode that will check if the **Customer_Code=0** and alert the end user.

Maintain the data consistency of the **Membership_Date** and **Membership_Time** variables, so that if the date is changed, the time is cleared.

7. Add a Variable Change logic unit to the **Membership_Date**.
8. Within the **Variable Change** logic unit, add an **Update** operation that updates the **Membership_Time** with **0**.

Do not allow the cursor to park on the **Membership_Time** Edit control if the **Membership_Date** is empty. In Magic xpa, the default value of a date variable is **'01/01/1901'Date**, therefore:

9. In the **Membership_Time** Edit control's **Allow Parking** property, set the following expression: (**Membership_Date<>'01/01/1901'Date**).

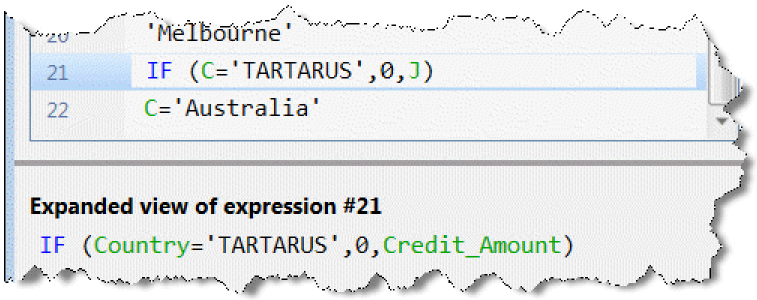
Lesson 5 – Initializing a Variable

You were asked to update the **City** variable to Melbourne only if the **Country** is Australia.

1. In the Logic Editor, go to the **Control Prefix** logic unit for the **City** variable.
2. Add a condition for the **Update** where **Country='Australia'**.

If the country is Tartarus, then set the initial credit to zero.

3. Zoom into the Data View Editor.
4. Park on the **Credit_Amount** variable.
5. Zoom into the **Init** property and set the following expression:
   IF (Country='Tartarus',0, Credit_Amount)



```
'Melbourne'
IF (C='TARTARUS',0,J)
C='Australia'
```

Expanded view of expression #21

```
IF (Country='TARTARUS',0,Credit_Amount)
```

The **Country** variable is an **Alpha** field. This is case sensitive. This means that the values Tartarus, tartarus and TARTARUS are ***not*** the same.

You can check what the user entered by changing the value of the input to either all upper case letters or all lower case letters. The **Upper** function changes all characters to upper case.

6. Park on the expression you just entered,
   IF (Country='Tartarus',0, Credit_Amount) and change it to:
   IF (Upper (Country) ='TARTARUS',0, Credit_Amount)

This only changes the value within the **IF** function and does not affect the actual value.

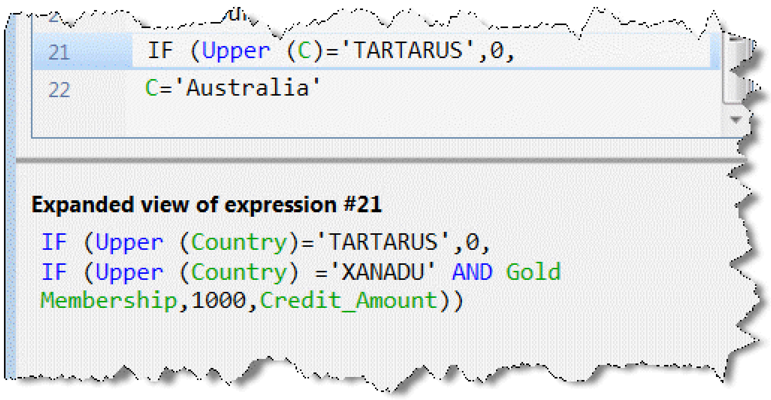
If the country is Xanadu and the customer has a gold membership, then you need to set the initial credit to 1000. To do this you need to make some changes to the previous expression:

7. Park on the expression you just entered,
   IF (Upper (Country) ='TARTARUS',0, Credit_Amount)

The **IF** function's syntax is IF (expression, THEN, ELSE). In the example you asked "*If the country is Tartarus then set the credit to zero.*" You did not take into account other possibilities, such as if the country is another specific value.

8. In the expression above, replace **Credit_Amount** with:
   IF (Upper (Country) ='XANADU' AND Gold Membership,1000, Credit_Amount)

   The expression will be more complex and will look like this:
   IF (Upper (Country) ='TARTARUS',0,
   IF (Upper (Country) ='XANADU' AND Gold Membership,1000, Credit_Amount))



```
21    IF (Upper (C)='TARTARUS',0,
22    C='Australia'
```

Expanded view of expression #21

```
IF (Upper (Country)='TARTARUS',0,
IF (Upper (Country) ='XANADU' AND Gold
Membership,1000,Credit_Amount))
```

Lesson 6 – Setting the Form's Appearance

Defining a new color for the Edit controls and name it **Editable control**:

1. From the **Options** menu, select **Settings** and then **Colors**.
2. Click the **Application** tab.
3. Park on the last color line. Create a line by pressing **F4**.
4. In the **Name** column, type: **Editable control**.
5. Zoom from the **FG** column.
6. Select the first (empty) entry from the **System** drop-down list.
7. Set the following RGB colors:
   a. Red = 0
   b. Green = 90
   c. Blue = 146.
8. Click **OK**.
9. Zoom from the **BG** column.
10. Select the first (empty) entry from the **System** drop-down list.
11. Check the **Transparent** check box.
12. Click **OK**.

Changing the Color and Font of all Edit controls:

1. Select all of the **Edit** controls on the form.
2. Change the **Color** of the controls to **Editable control**.
3. Change the **Font** of the controls to **Text**.

In order for the color and font files to be specific for this project, you will now copy the color and font file to your project directory and set the color and font files in the Application properties using a logical name.

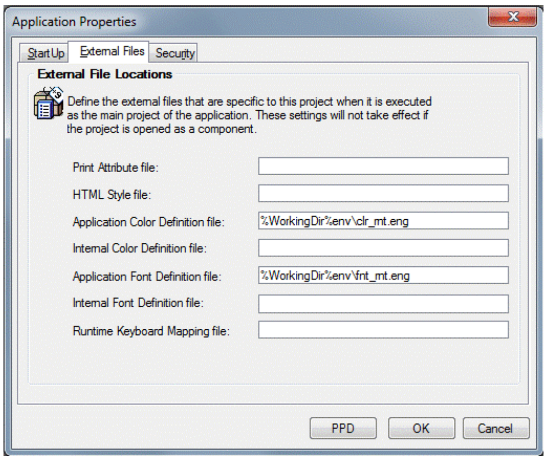
Making the Color and Font Files Application Specific

Using the **Windows Explorer** navigator:

1. In your **Project** (**Getting Started**) directory, make sure that you have the **env** folder. This should be where the **Wallpaper.jpg** image is found.
2. From the **Support** directory (located in the Magic xpa installation directory), copy the **clr_rnt.eng** and the **fnt_rnt.eng** files to the **Env** directory that you just created.

Setting the Color and Font Files in the Application Properties

3. Open the **Getting Started** project.
4. From the **File** menu, select **Application Properties** (Ctrl+Shift+P).
5. Click on the **External Files** tab.
6. In the **Application Color Definition file** property, type: **%WorkingDir%env\clr_rnt.eng**
7. In the **Application Font Definition file** property, type: **%WorkingDir%env\fnt_rnt.eng**
8. Zoom into each definition to check whether it displays the font and color file that you created.



Define the external files that are specific to this project when it is executed as the main project of the application. These settings will not take effect if the project is opened as a component.

Print Attribute file:	
HTML Style file:	
Application Color Definition file:	%WorkingDir%env\clr_rnt.eng
Internal Color Definition file:	
Application Font Definition file:	%WorkingDir%env\fnt_rnt.eng
Internal Font Definition file:	
Runtime Keyboard Mapping file:	

You will not see any changes in your project. However, from now on, all changes that you make to the color or the font files, will apply to the files belonging to your project and not to the generic files in the **Support** directory.

Placement

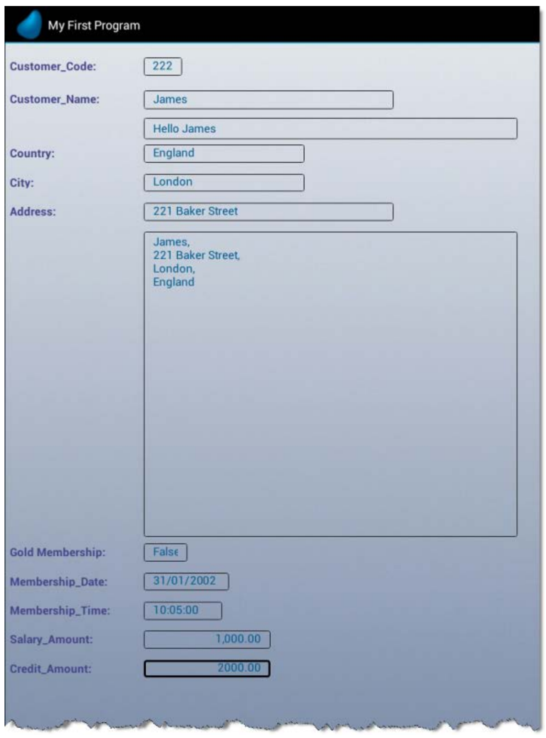
Defining the placement of **Membership_Date**, **Membership_Time**, **Salary_Amount**, and **Credit_Amount** so that they move together with the **Gold_Membership**:

1. Select all the Edit controls of **Membership_Date**, **Membership_Time**, **Salary_Amount** and **Credit_Amount** together with the Label controls.
2. Zoom into the control property sheet.
3. In the **Navigation** section, zoom into the **Placement** property.
4. Set only the **Y** property (the one at the top). The property will be set to **0,0,100,0**.

Defining the **Customer_Name**, **welcome customer** and **Address** controls so that their width increases as the form increases but they remain in place:

5. Select the Edit controls of **Customer_Name**, **Address** and the **Hello Customer** control, but not their Label controls.
6. Zoom into the control property sheet.
7. In the **Navigation** section, zoom into the **Placement** property.
8. Set only the **Width** property (the one at the bottom). The property will be set to **0,100, 0,0**.
9. Execute the program.

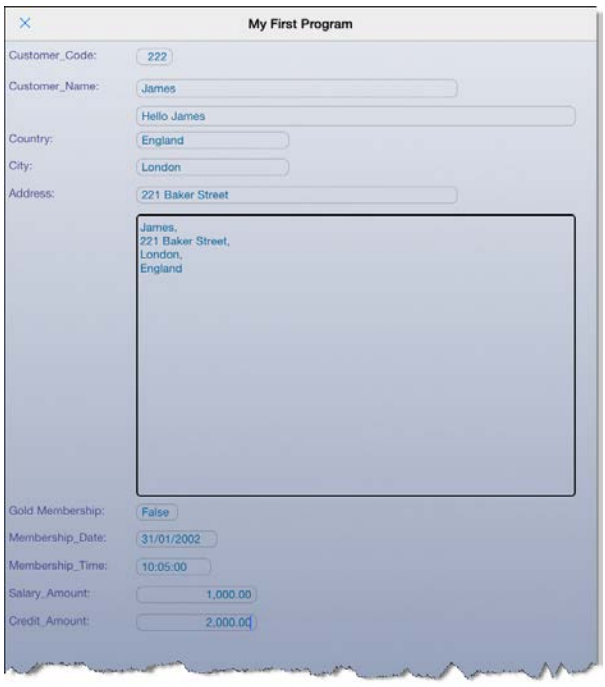
Android



My First Program

Customer_Code:	222
Customer_Name:	James
	Hello James
Country:	England
City:	London
Address:	221 Baker Street
	James, 221 Baker Street, London, England
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2000.00

iOS



My First Program

Customer_Code:	222
Customer_Name:	James
	Hello James
Country:	England
City:	London
Address:	221 Baker Street
	James, 221 Baker Street, London, England
Gold Membership:	False
Membership_Date:	31/01/2002
Membership_Time:	10:05:00
Salary_Amount:	1,000.00
Credit_Amount:	2,000.00

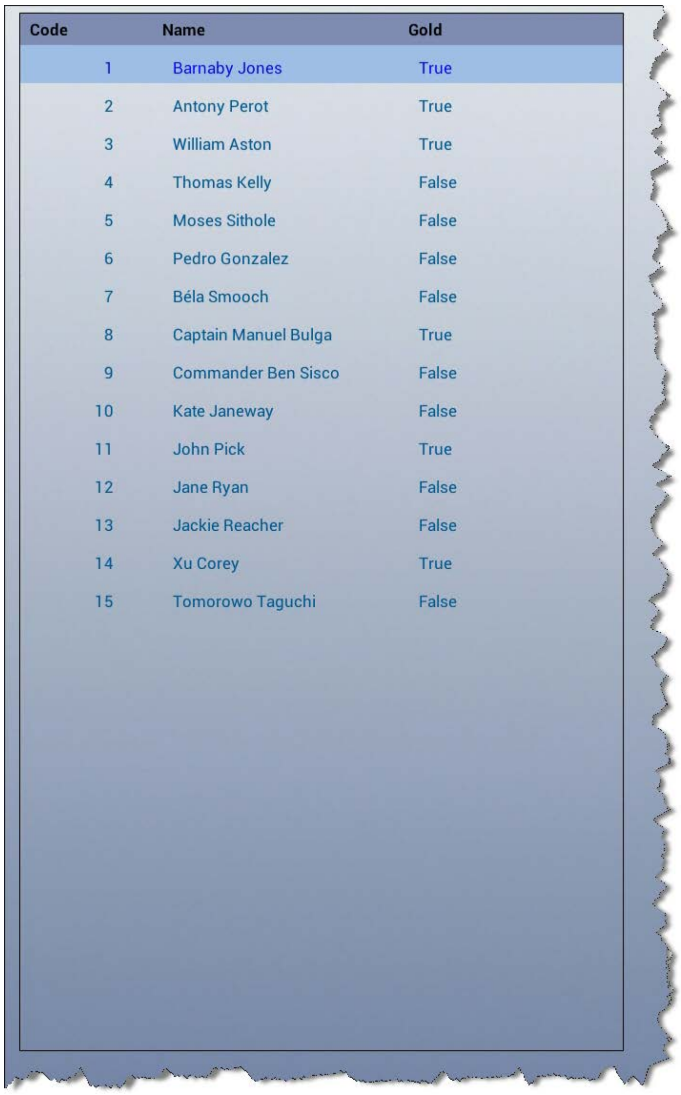
As you can see, you now have a similar look and feel for both environments.

Lesson 7 – Viewing Data Source Content

Increasing the table size as the form increases:

1. Zoom into **Customers - Line Mode** program and zoom into the Form Designer.
2. Park on the Table control and open the control property sheet.
3. Click on the **Placement** property and then click the Zoom button.
4. Set the value of **100** in the **Height** property. Click **OK**.
5. The **Placement** property will show: **0, 0,0,100**.

When you run the program, you will now see all of the records:



Code	Name	Gold
1	Barnaby Jones	True
2	Antony Perot	True
3	William Aston	True
4	Thomas Kelly	False
5	Moses Sithole	False
6	Pedro Gonzalez	False
7	Béla Smooch	False
8	Captain Manuel Bulga	True
9	Commander Ben Sisco	False
10	Kate Janeway	False
11	John Pick	True
12	Jane Ryan	False
13	Jackie Reacher	False
14	Xu Corey	True
15	Tomorowo Taguchi	False

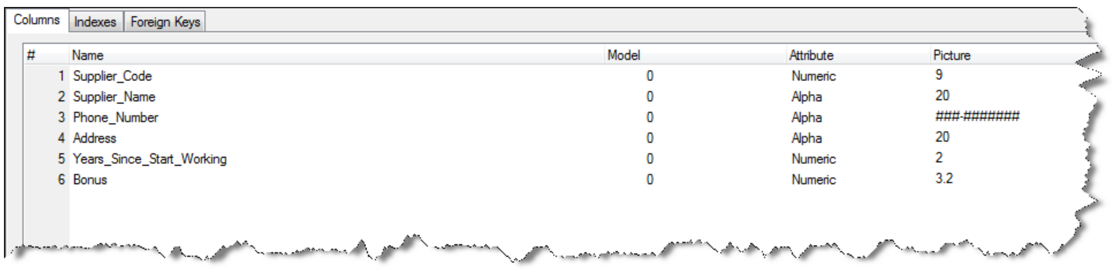
Suppliers Program

Now, you will define the **Suppliers** data source.

1. From the **Project** menu, select **Data** (**Shift+F2**) to open the Data repository.
2. Create a line.
3. In the **Name** column, type **Suppliers**.
4. In the Data source name column, type Suppliers.
5. From the **Database** column, zoom to the **Database list**, and select the **Getting Started** entry.

Defining columns:

6. Click the **Columns** tab in the lower pane.
7. Define column entries for the items **exactly** as shown in the image below:



#	Name	Model	Attribute	Picture
1	Supplier_Code	0	Numeric	9
2	Supplier_Name	0	Alpha	20
3	Phone_Number	0	Alpha	###-#######
4	Address	0	Alpha	20
5	Years_Since_Start_Working	0	Numeric	2
6	Bonus	0	Numeric	3.2

8. Define a unique index called **Supplier_Code** with a segment for the **Supplier_Code** column.

Defining the Program

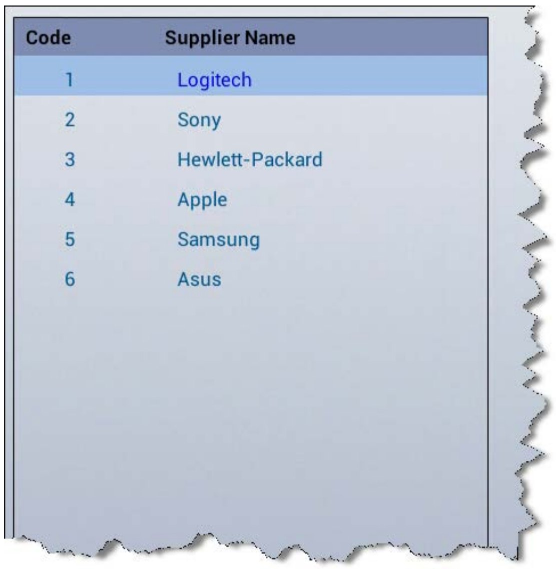
Now you will define the program that will display the suppliers in a table:

1. Create a program and name it: **Suppliers - Line Mode**.
2. Use the **Suppliers** data source as the program's Main Source.
3. Select all of the **Suppliers** data source columns.

Define the **Suppliers - Line Mode** form as follows:

4. Provide wallpaper for the form. Set the **Wallpaper** property to: **%WorkingDir%env\Wallpaper.jpg**.
5. Add a Table control to the form. Set the following properties for the table:
    a. Set the **Set Table Color** property to **Table**.
    b. Set the **Color** property to **Text** color.
    c. Set the **Row Highlight Color** property to the **Row Highlight** color.
6. Attach the **Supplier_Code** and **Supplier_Name** variables to the Table control.
7. Set the **Color** property to **Edit control**.
8. Click on the column header of the **Supplier_Code** column and zoom into the control property sheet.
9. Set the **Column title** property to **Code**.
10. Park on the **Color** property and select the **Text Caption** color.
11. Park on the **Font** property and select the **Text Caption** font.
12. Click on the column header of the **Supplier_Name** column and zoom into the control property sheet.
13. Park on the **Color** property and select the **Text Caption** color.
14. Park on the **Font** property and select the **Text Caption** font.

When you run the program, it will look similar to the image below:



Code	Supplier Name
1	Logitech
2	Sony
3	Hewlett-Packard
4	Apple
5	Samsung
6	Asus

Lesson 8 – Models – Object Definition Centralization

Defining Models

Now you will create models for the **Customers** and **Suppliers** data sources.

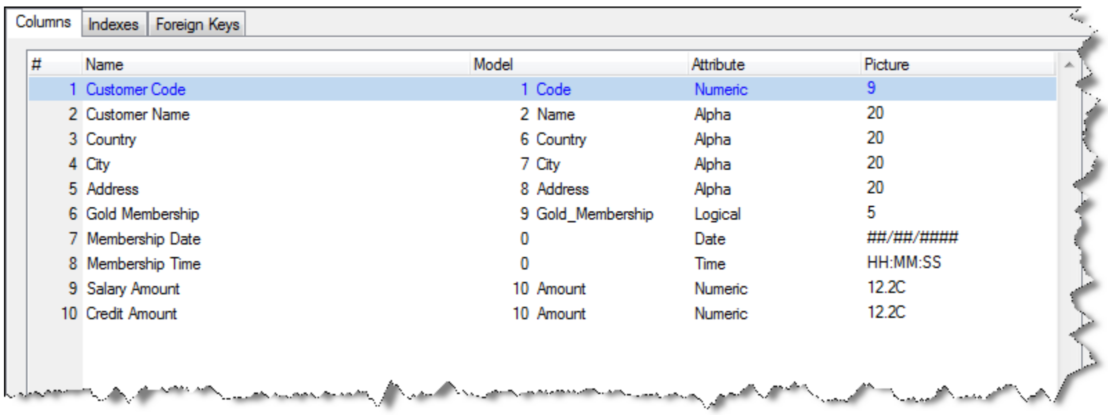
- Add the following models for the data sources and set the parameters as shown in the table below.

Name	Class	Attribute	Model Properties
Country	Field	Alpha	Picture: 20
City	Field	Alpha	Picture: 20
Address	Field	Alpha	Picture: 20
Gold_Membership	Field	Logical	A Logical field attribute has a default picture (5 digits).
Amount	Field	Numeric	Picture: 12.2C
Phone_Number	Field	Alpha	Picture: ###-#######
Years_Since_Start_Working	Field	Numeric	Picture: 2
Bonus	Field	Numeric	Picture:3.2

Assigning a Field Model to a Column

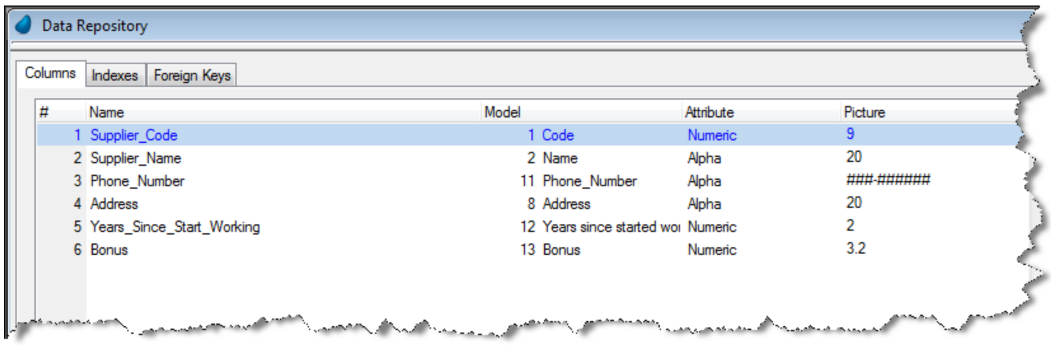
1. Review the **Assigning a Field Model to a Column** section in this lesson.
2. Assign models to all of the remaining **Customers** and **Suppliers** data source columns.
3. Park on a column and open its properties (**Alt+Enter**).
4. From the **Model** property, zoom and select the model shown in the image below.
5. From the **Picture** property, click the **Inherit** button.
6. Verify that the button displays after you click the button.

Customers data source:



#	Name	Model	Attribute	Picture
1	Customer Code	1 Code	Numeric	9
2	Customer Name	2 Name	Alpha	20
3	Country	6 Country	Alpha	20
4	City	7 City	Alpha	20
5	Address	8 Address	Alpha	20
6	Gold Membership	9 Gold_Membership	Logical	5
7	Membership Date	0	Date	##/##/####
8	Membership Time	0	Time	HH:MM:SS
9	Salary Amount	10 Amount	Numeric	12.2C
10	Credit Amount	10 Amount	Numeric	12.2C

Suppliers data source:



#	Name	Model	Attribute	Picture
1	Supplier_Code	1 Code	Numeric	9
2	Supplier_Name	2 Name	Alpha	20
3	Phone_Number	11 Phone_Number	Alpha	###-######
4	Address	8 Address	Alpha	20
5	Years_Since_Start_Working	12 Years since started woı	Numeric	2
6	Bonus	13 Bonus	Numeric	3.2

Assigning a Model to a Control

Follow the example from the **Assigning a Model to a Control** section.

1. Open the Customers - Screen Mode task's form.
2. Assign the **Edit control** model to all of the Edit controls.
3. Park on the **Color** property for each of the Edit controls and inherit the model's properties.

Defining the Display Only Model

You will now define the model for a **Display Only** Edit control:

1. From the **Project** menu, select **Models** (Shift+F1).
2. Create a line.
3. In the **Name** column, type: **Display Only**.
4. From the **Class** column, select **Rich Client Display**.
5. From the **Attribute** column, select **Edit**.
6. Open the control properties sheet.
7. In the **Appearance** section, set the **Border** property to **No**.
8. Zoom from the **Color** property and set the color to **Text Caption**.

Defining Models for Tables

You will now define the models for the various sections of a table:

1. From the **Project** menu, select **Models** (Shift+F1).
2. Create a line.
3. In the **Name** column, type: **Table**.
4. From the **Class** column, select **Rich Client Display**.
5. From the **Attribute** column, select **Table**.
6. Open the control properties sheet.
7. In the **Appearance** section, set the **Set Table Color** property to **by Table**. You will see that the **Color** property above it is now accessible.
8. Zoom from the **Color** property and select the **Text** color.
9. Zoom from the **Row Highlight Color** and select the color you defined previously, **Row Highlight**.
10. Click on the **Placement** property and then click the Zoom button.
11. Set the value of **100** in the **Height** property. Click **OK**.

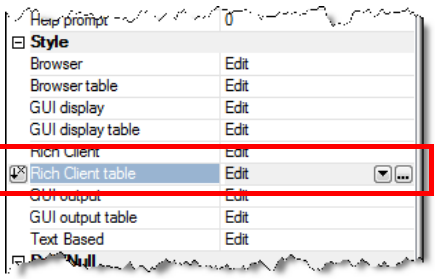
    The **Placement** property will show: **0, 0,0,100**.

Defining the column header:

12. Create a line.
13. In the **Name** column, type: **Column Heading**.
14. From the **Class** column, select **Rich Client Display**.
15. From the **Attribute** column, select **Column**.
16. Set the **Color** property to **Text Caption**.
17. Set the **Font** property to **Text Caption**.

Now you need to set the name and code models so that when they are placed in a Rich Client table, they have specific properties:

18. In the Model repository, park on the **Code** model.
19. Zoom into the properties.
20. In the **Style** section, park on the **Rich Client table** property and click the Zoom button. This opens a property sheet for an Edit control.
21. Set the **Border** property to **No**.
22. Set the **Color** property to **Edit Control**.
23. Repeat steps 18 to 22 for the **Name** model.



Help prompt	0
⊟ Style	
Browser	Edit
Browser table	Edit
GUI display	Edit
GUI display table	Edit
Rich Client	Edit
Rich Client table	Edit
GUI output	Edit
GUI output table	Edit
Text Based	Edit
Null	

Now you need to apply the models:

1. Zoom into the Program repository.
2. Zoom into the **Customers - Line Mode** and zoom into the Form Designer.
3. Park on the table and zoom into the control properties.
4. In the **Model** property, zoom into the models and select **Table**.
5. Inherit the properties for Set Table Color, Color, Row Highlight Color and Placement.
6. Open the **Customer Code** column properties.
7. In the **Model** property, zoom into the models and select **Column**.
8. Inherit the properties for **Color** and **Font**.
9. Repeat steps 6 to 8 for the **Customer_Name** and the **Gold_Membership** columns.
10. Park on the **Customer_Code** Edit control and inherit the property for **Border** and **Color**.
11. Park on the **Customer_Name** Edit control and inherit the property for **Border** and **Color**.

Lesson 10 – Events and Handlers

During the lesson you raised the **Set Gold Membership** event but there was no handler for it:

1. Zoom into the **Customers - Screen Mode** program.
2. Zoom into the Logic Editor.
3. Create a header line (**Ctrl+H**) and select **Event**.
4. In the **Event** dialog box, select **User** and then select **Set Gold Membership** from the list of User events.
5. Add a detail line and **Update** the **Gold_Membership** variable with **True**.
6. Add a detail line and **Update** the **Credit_Amount** variable with the value **Credit_Amount*1.2**.

You were asked to enable the end user to delete a customer.

1. Zoom into the **Customers - Line Mode** program.
2. Zoom into the Form Designer.
3. Place a Button control at the top of the form.
4. Zoom into the control properties.
5. Park on the **Format** property and remove the text which is initially **Button**.
6. Park on the **Button style** property and select **Image Button** from the combo box.
7. Park on the **Image List file name** property and type **%WorkingDir%\images\Delete.png**.
8. From the **Event Type** property, select **Internal**.
9. From the **Event** property, select **Delete Line**.
10. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
11. From the toolbar, click the **Fit Control Size** icon.

Now you will create the handler for this:

1. Zoom into the Logic Editor.
2. Create a header line (**Ctrl+H**) and select **Event**.
3. In the **Event** dialog box, select **Internal** and then select **Delete Line**.
4. Set the **Propagate** property to **Yes**.
5. Add a detail line and select **Raise Event**.
6. In the **Event** dialog box, select **Internal** and then select **Close** from the list.

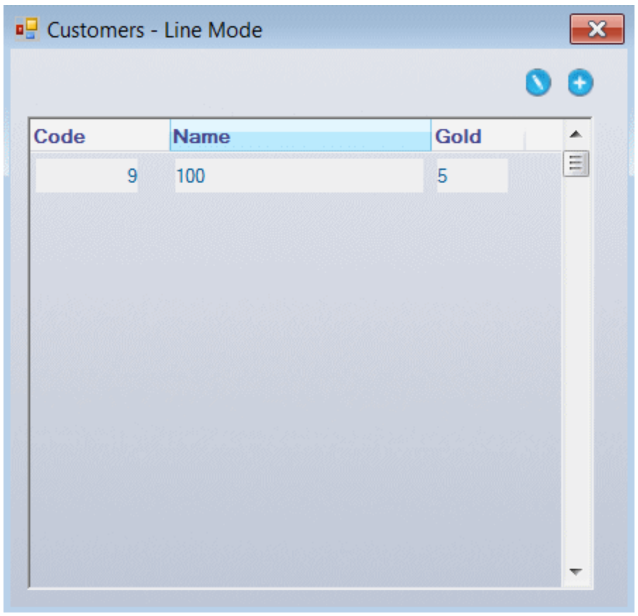
This means that the program will close when you delete a line. However, you were only asked to do this if the program was called from the **Customers - Line Mode** program. In this case you know that the **P.Customer Code** parameter has a value:

7. Zoom from the **Cnd** property and enter the following expression:
   P.Customer Code > 0

You were asked to add a button to allow the user to add a customer. As with the modify record situation, this involves both the calling program and the called program.

1. Zoom into the **Customers - Line Mode** program.
2. Zoom into the User Events repository by pressing **CTRL+U**.
3. Add a line and set the name to **Add Customer**.
4. Set the **Trigger type** to **None**.
5. Zoom into the Form Designer.
6. Place a **Button** control at the top of the form.
7. Zoom into the control properties.
8. Park on the **Format** property and remove the text, which is initially **Button**.
9. Park on the **Button style** property and select **Image Button** from the combo box.
10. Park on the **Image List file name** property and type **%WorkingDir%\images\Add.png**.
11. From the **Event Type** property, select **User**.
12. From the **Event** property, select **Add Customer**.
13. Set the **Color** to **Text Caption**.
14. Click the **Fit Control Size** icon.

Your form will look similar to the following image.



Code	Name	Gold
9	100	5

15. Zoom into the Logic Editor.
16. Create a header line (**Ctrl+H**) and select **Event**.
17. In the **Event** dialog box, select **User** and then select **Add Customer**.
18. Create a detail line for the **Call Program** operation.
19. Select the **Customers - Screen Mode** program from the list.

Since you are adding a new customer, you do not need to send the customer code as a parameter to the **Customers - Screen Mode** program. However, that program expects parameters and if you use syntax checking, **F8**, on the **Customers - Line Mode** program, you will get an error.

20. From the **Arguments** field, zoom to the Arguments repository.
21. Create a line and check the **Skip** box. In this case, when the customer code is passed to the called program, it will be passed as zero.

Now you need to handle the "create" situation in the **Customers - Screen Mode** program. Remember that the program is opened in Modify mode and in the current scenario, the **P.Customer Code** value is zero. You need to open the task in Create mode. How do you open the same task in Modify mode in one scenario and in Create mode in another scenario? By using a parameter, of course.

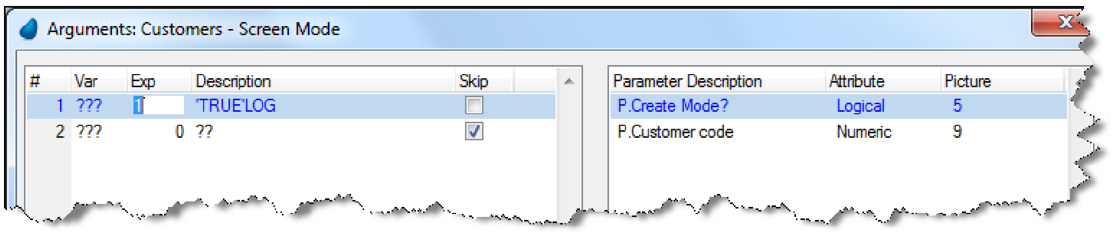
22. Zoom into the **Customers - Screen Mode** program.
23. Park on the first line, the **Main Source** definition, and create a line.
24. From the drop-drown list, select **Parameter**.
25. You are now parked in a field displaying **??**. Type in **P.Create Mode?**.
26. Set the **Picture** to **Logical**.
27. Open the **Task Properties** dialog box by pressing **CTRL+P**.
28. In the **Initial mode** property, select **By Exp**.
29. Zoom from the **Exp:** box to the right of the **Initial Mode**.
30. In the Expression Editor, add an expression: **IF (P.Create Mode?, 'C', 'M')**
    This means that if **True** is passed in the **P.Create Mode** parameter, the task will be in Create mode and if not it will be in Modify mode.

A better method of doing this would be to use the **MODE** literal. Your expression will then be: **IF (P.Create Mode?, 'C'MODE, 'M'MODE).**

Now you need to call the program with the new parameter:

1. Zoom into the **Customers - Line Mode** program.
2. Zoom into the **Add Customer** logic unit.
3. Zoom into the **Call Program** detail line.
4. From the **Arguments** field, zoom to the Arguments repository.
5. Create a line as the first line, meaning before the line where you checked the **Skip** box.

6. Zoom from the **Exp** column and add the following expression: **'TRUE'LOG**.


Arguments: Customers - Screen Mode

#	Var	Exp	Description	Skip
1	???	1	'TRUE'LOG	☐
2	???	0	??	☑

Parameter Description	Attribute	Picture
P.Create Mode?	Logical	5
P.Customer code	Numeric	9

7. Add a detail line after the **Call Program** operation.
8. Select the **Raise Event** operation from the drop-down list.
9. In the **Event** dialog box, from the **Event Type** drop-down list, select **Internal**.
10. From the **Event** field, select the **View Refresh** event and click **OK**.

You added a parameter as the first parameter in the **Customers - Screen Mode** program. However, you called this same program from another handler, the **Modify Records** handler. You need to make a change there.

11. Zoom into the **Modify Records** handler.
12. Zoom into the **Call Program** detail line.
13. From the **Arguments** field, zoom to the Arguments repository.
14. Create a line as the first line, meaning before the line where you passed the Customer Code to the called program.

In this case, you can either zoom from the **Exp** column and add an expression, **'FALSE'LOG**, or you can check the **Skip** box.

15. Check the **Skip** box.

You can now execute the application.

Lesson 11 – Conditioning a Block of Operations

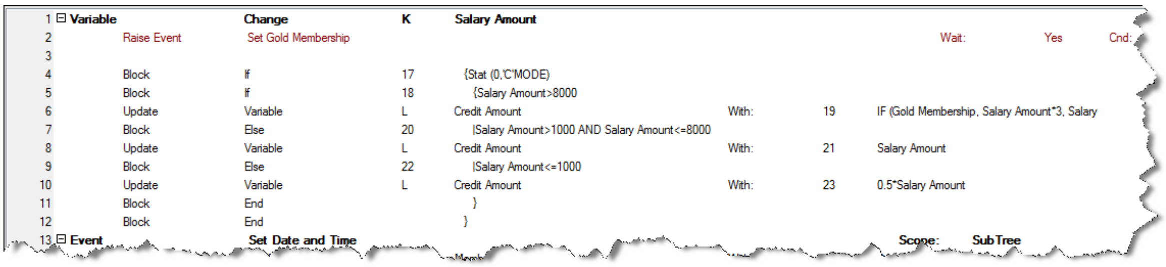
You were asked to update the **Credit_Amount** value according to the following logic when adding a new customer:

- For all customers whose **Salary_Amount** is more than 8000:
    - For customers who do not have **Gold Membership**, the **Credit_Amount** value will be updated with the **Salary_Amount*2**.
    - For customers who have **Gold Membership**, the **Credit_Amount** value will be updated with the **Salary_Amount*3**.
- If the **Salary_Amount** is more than 1000, but less than 8000, update the **Credit_Amount** with the **Salary_Amount**.
- If the **Salary_Amount** is less than 1000, update the **Credit_Amount** with 50% of the **Salary_Amount**.

In this solution, you will do this in a **Variable Change** logic unit for the **Salary_Amount.** You can do this in a **Record Suffix** logical unit with no changes to the actual code. The advantage to the user for doing this in the **Variable Change** logic unit is that the user can see the change to the **Credit_Amount** and make manual changes as well.

1. Zoom into the **Customers - Screen Mode** program and zoom into the Logic Editor.
2. Zoom into the **Variable Change** logic unit for the **Salary_Amount**.
3. Add a detail line after the **Raise Event** for **Set Gold Membership**.
4. Add a **Block If** operation and set the condition to: **Stat (0,'C'MODE)**.
   This will return True if the program is in Create mode.
5. Add a nested **Block If** operation and set the condition to: **Salary_Amount > 8000**.
6. Add a detail line and set an **Update** operation for the **Credit_Amount** variable.
7. Zoom from the **With** property and add an expression:
   IF (Gold_Membership, Salary_Amount *3, Salary_Amount *2)
   Remember that when dealing with a logical data type, you only need to use the name of the variable; you do not need to add **Gold_Membership = 'True'LOG**.
8. Add a Block Else operation and set the condition to:
   Salary_Amount > 1000 AND Salary_Amount <= 8000

9. Add a detail line and set an **Update** operation for the **Credit_Amount** variable.
10. Zoom from the **With** property and add an expression for **Salary_Amount**.
11. Add another **Block Else** operation and set the condition to:
    Salary_Amount <= 1000
12. Add a detail line and set an **Update** operation for the **Credit_Amount** variable.
13. Zoom from the **With** property and add the following expression:
    Salary_Amount * 0.5.



```
1  ⊟ Variable        Change              K    Salary Amount
2       Raise Event   Set Gold Membership                                                                Wait:      Yes     Cnd:
3
4       Block         If                  17     {Stat (0,'C'MODE)
5       Block         If                  18       {Salary Amount>8000
6       Update        Variable            L    Credit Amount                         With:     19     IF (Gold Membership, Salary Amount*3, Salary
7       Block         Else                20       |Salary Amount>1000 AND Salary Amount<=8000
8       Update        Variable            L    Credit Amount                         With:     21     Salary Amount
9       Block         Else                22       |Salary Amount<=1000
10      Update        Variable            L    Credit Amount                         With:     23     0.5*Salary Amount
11      Block         End                          }
12      Block         End                        }
13 ⊟ Event            Set Date and Time                                                          Scope:     SubTree
```

Lesson 12 – One-to-One Data Relationships

Orders Scenario

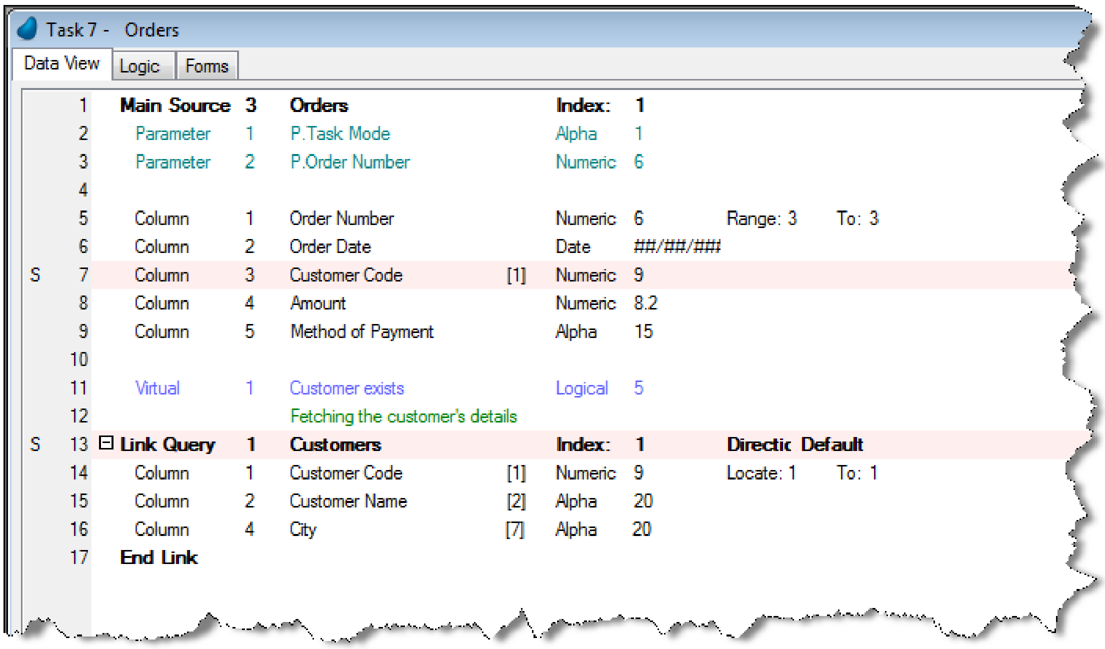
First you will make the changes to the Orders scenario.

1. Zoom into the **Orders** program.
2. Add two parameters:

a. **P.Task Mode**, which will be an Alpha parameter, one character in length. This can hold Q for Query, C for Create or M for Modify.

b. **P.Order Number**, a numeric parameter of size 6.

3. Park on the **Order Number** column of the **Orders** data source and zoom into the **Range From** property.
4. Set the following expression:
   CndRange (P.Order Number > 0, P.Order Number)
   If you execute the **Orders** program directly, the first order will always be displayed.
5. Enter the same expression for the **Range To** property.



	1	**Main Source**	**3**	**Orders**		**Index:**	**1**		
	2	Parameter	1	P.Task Mode		Alpha	1		
	3	Parameter	2	P.Order Number		Numeric	6		
	4								
	5	Column	1	Order Number		Numeric	6	Range: 3	To: 3
	6	Column	2	Order Date		Date	##/##/###		
S	7	Column	3	Customer Code	[1]	Numeric	9		
	8	Column	4	Amount		Numeric	8.2		
	9	Column	5	Method of Payment		Alpha	15		
	10								
	11	Virtual	1	Customer exists		Logical	5		
	12			Fetching the customer's details					
S	13	⊟ **Link Query**	**1**	**Customers**		**Index:**	**1**	**Directic**	**Default**
	14	Column	1	Customer Code	[1]	Numeric	9	Locate: 1	To: 1
	15	Column	2	Customer Name	[2]	Alpha	20		
	16	Column	4	City	[7]	Alpha	20		
	17	**End Link**							

6. Open the **Task Properties** dialog box and from the **Initial mode** property, select **By Exp**.
7. Zoom into the **Exp** field and define an expression.

There are a few ways to define the expression for the Initial mode.

- Simply use the parameter. Since **P.Task Mode** will contain C, for example, you can simply use that. This is problematic when using a multilingual system because Q is Query in English but in German it may be A for Abfrage.
- Use the **IF** function. You can do this as follows:
  IF (P.Task Mode = 'C','C'MODE, IF (P.Task Mode = 'M','M'MODE, 'Q'MODE))
- Use the **CASE** function. You have not learned this function in this course. You can read about it in the Magic xpa Help. When using the CASE function, the function will be:
  CASE (A,'C','C'MODE,'M','M'MODE, 'Q'MODE)

You were also asked to set the current date as the initial date when you are creating a new order. This is a simple request. You use the **Init** property. For a **Column** variable, the **Init** property is evaluated only when the program is in Create mode:

8. Park on the **Order_Date** variable. Zoom into the **Init** property and set the following expression: **Date ()**.

You were asked to add a button so that the user could delete the order if the program is in Modify mode.

9. Zoom into the Form Designer.
10. Place a **Button** control at the top of the form.
11. Zoom into the control properties.
12. Park on the **Format** property and remove the text, which is initially **Button**.
13. Park on the **Button style** property and select **Image Button** from the combo box.
14. Park on the Image List file name property and type:
    %WorkingDir%\images\Delete.png
15. From the **Event Type** property, select **Internal**.
16. From the **Event** property, select **Delete Line**.
17. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
18. Park on the **Visible** property and set the following condition: **Stat (0,'M'MODE)**.
19. From the toolbar, click the **Fit Conrol Size** ▣ icon.

Now you will create the handler for this:

1. Zoom into the Logic Editor.
2. Create a header line (**Ctrl+H**) and select **Event**.
3. From the **Event** dialog box, select **Internal** and then select **Delete Line**.
4. Set the **Propagate** property to **Yes**.
5. Add a detail line and select **Raise Event**.
6. From the **Event** dialog box, select **Internal** and then select **Close** from the list.

This means that the program will close when you delete a line.

Now you need to create the **List of Orders** program that displays the list of orders:

1. Create a program and name it **List of Orders**. Remember to set the **Public name** to **RunMe** and to check the **External** box.
2. Zoom to the **List of Orders** program.
3. From the **Task Properties** dialog box, set the **Task type** to **Rich Client**.
4. Set the **Initial Mode** to **Query**.
5. Open the Data View Editor of the **Orders** program.
6. Create a **Main Source** definition for the **Orders** data source and use the **Order_Number** index.
7. Add the following columns from the **Orders** data source:
   - Order_Number
   - Order_Date
   - Customer_Code
8. Create a **header line** and from the comb box, select **Link Query**.
9. Select the **Customers** data source.
10. Zoom from the **Index** property and select the first index. The **Customer_Code** is added automatically.
11. In the column's **Locate from** and **Locate to** properties, zoom and select the **Customer_Code** field from the **Orders** data source.
12. Add a line and select the **Customer_Name** column.

Since the **List of Orders** program is a display only program, there is no need to add a link success indication.

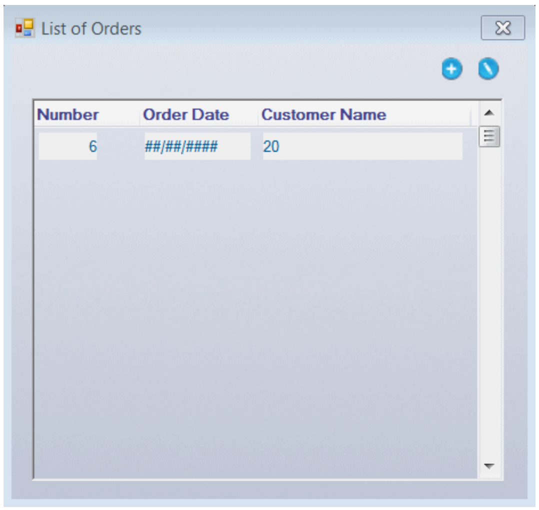
Define the **Suppliers - Line Mode** form as follows:

1. Zoom into the form properties and select the **Table Display Form** model.
2. Zoom into the Form Designer.
3. Drop a Table control onto the form. Zoom into the control properties and select the **Table mode**.
4. Attach the **Order Number** to the Table control. Zoom into the control properties and select the **Edit Control** model.
5. Open the **Order_Number** column properties.
6. Select the **Column Header** model.
7. Set the **Column title** property to **Number**.
8. Attach the **Order Date** to the Table control. Zoom into the control properties and select the **Edit Control** model.
9. Open the **Order_Date** column properties.
10. Select the **Column Header** model.
11. Attach the **Customer Name** to the Table control. This is already connected to the **Edit Control** property.
12. Open the **Customer Name** column properties.
13. Select the **Column Header** model.
14. Increase the width of the table so that all three columns are visible.
15. Increase the height of the table.

You now need to add the buttons for Add and Modify. To do this you need to add User events:

1. Zoom into the User Events repository by pressing **CTRL+U**.
2. Add a line and set the name to **New Order**.
3. Set the **Trigger type** to **None**.
4. Add a line and set the name to **Change Order**.
5. Set the **Trigger** type to **None**.
6. Zoom into the Form Designer.
7. Place a Button control at the top of the form.
8. Zoom into the control properties.
9. Park on the **Format** property and remove the text, which is initially **Button**.
10. Park on the **Button style** property and select **Image Button** from the combo box.
11. Park on the **Image List file name** property and type **%WorkingDir%\images\Add.png**.
12. From the **Event Type** property, select **User**.
13. From the **Event** property,select **New Order**.
14. Set the **Color** to **Text Caption**.
15. Click the **Fit to Size** icon.

16. Place a Button control at the top of the form, to the right of the Add button.
17. Zoom into the control properties.
18. Park on the **Format** property and remove the text, which is initially **Button**.
19. Park on the **Button style** property and select **Image Button** from the combo box.
20. Park on the **Image List file name** property and type **%WorkingDir%\images\Edit.png**.
21. From the **Event Type** property, select **User**.



22. From the **Event**property, select **Change Order**.
23. Set the **Color** to **Text Caption**.
24. Click the **Fit Control Size** icon.

Now you need to handle the events:

1. Zoom into the Logic Editor.
2. Add a header line and select the **New Order** User event.
3. Create a detail line for the **Call Program** operation.
4. Select the **Orders** program from the list.
5. Zoom from the **Arguments** property.
6. In the **Arguments** dialog box:

a. Add a line with the expression: '**C**'. You are entering in Create mode.
   b. Add another line and check the **Skip** box. Since this is Create mode, there is no need to pass an order.

7. Add a detail line and raise the **View Refresh** internal event.
8. Add a header line and select the **Change Order** User event.
9. Create a detail line for the **Call Program** operation.
10. Select the **Orders** program from the list.
11. Zoom from the **Arguments** property.
12. In the **Arguments** dialog box:

a. Add a line with the expression: '**M**'. You are entering in Modify mode.
b. Add another line, zoom from the Variable list and select the **Order Number**.

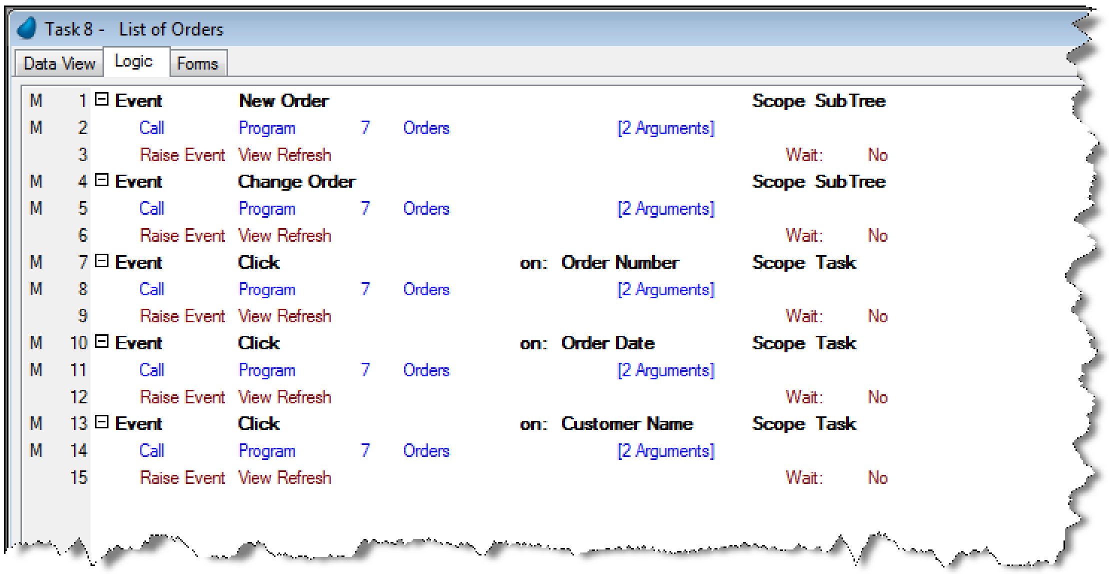
13. Add a detail line and raise the **View Refresh** internal event.

Now you will handle the case where the user taps an order in the list. This raises the internal event **Click**.

14. Add a header line and select the **Click** internal event.
15. Create a detail line for the **Call Program** operation.
16. Select the **Orders** program from the list.
17. Zoom from the **Arguments** property.
18. In the **Arguments** dialog box:
    c. Add a line with the expression: '**Q**'. You are entering in Query mode.
    d. Add another line, zoom from the Variable list and select the **Order Number**.
19. Add a detail line and raise the **View Refresh** internal event.

If you execute the application and then tap the Add push button, Magic xpa will raise the **New Orde**r event and will display the **Orders** program in Create mode, but it will also raise the **Click** event. This is because tapping raises the **Click** event. You need to handle this event only when the tap is made on a control on the table.

20. Park on the header line for the **Click** internal event.
21. Zoom from the **on:** property and select the **Order Number** control.
22. Repeat steps 14 to 21 for the **Order Date** control.
23. Repeat steps 14 to 21 for the **Customer Name** control.



M	1	⊟ Event	New Order					Scope	SubTree
M	2	Call	Program	7	Orders		[2 Arguments]		
	3	Raise Event	View Refresh					Wait:	No
M	4	⊟ Event	Change Order					Scope	SubTree
M	5	Call	Program	7	Orders		[2 Arguments]		
	6	Raise Event	View Refresh					Wait:	No
M	7	⊟ Event	Click			on:	Order Number	Scope	Task
M	8	Call	Program	7	Orders		[2 Arguments]		
	9	Raise Event	View Refresh					Wait:	No
M	10	⊟ Event	Click			on:	Order Date	Scope	Task
M	11	Call	Program	7	Orders		[2 Arguments]		
	12	Raise Event	View Refresh					Wait:	No
M	13	⊟ Event	Click			on:	Customer Name	Scope	Task
M	14	Call	Program	7	Orders		[2 Arguments]		
	15	Raise Event	View Refresh					Wait:	No

Products Scenario

In the following exercise you will define the **Products** data source. Then, you will create the **Products** program to display the products. You will use the **Link** operation to extend the data view and display the supplier details. Finally, you will use the Link **Success indication** property to check the validity of the **Supplier_Code** value.

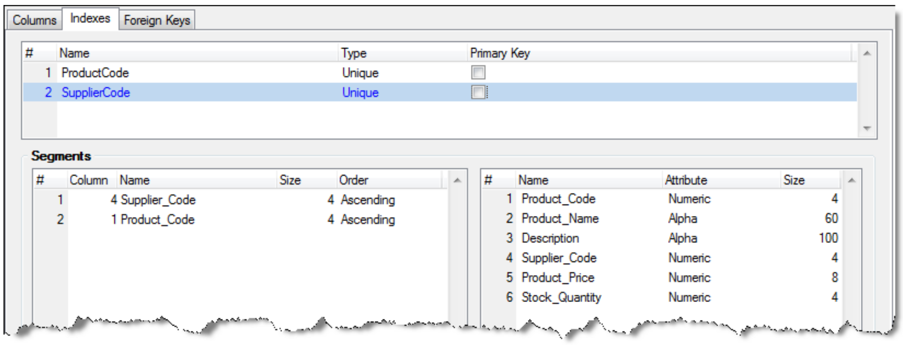
Defining the Products Data Source

1. In the Data repository, create a line.
2. In both the **Name** and **Data source name** column, type: **Products**.
3. From the **Database** column, zoom and select **Getting Started**.
4. Click on the **Columns** tab.
5. Press **F4** for each of the columns and define the following columns:

Name	Model	Attribute	Picture
Product_Code	Code	Numeric	5
Product_Name	0	Alpha	60
Description	0	Alpha	100
Supplier_Code	Code	Numeric	9
Product_Price	0	Numeric	6.2
Stock_Quantity	0	Numeric	6

6. Click the **Indexes** tab.
7. Create a line (press **F4**).
8. In the **Name** column, type **ProductCode**. Check that the **Type** is set to **Unique**.
9. In the **Name** column, type **SupplierCode**. Check that the **Type** is set to **Unique**.
10. From the **ProductCode** index, zoom (**F5**) to the Segment repository.
11. Create a line and press **F5**.
12. Select the **ProductCode**. Check that the number **1** now appears in the **Column** column.
13. Define two index segments for the **SupplierCode** index.

a. Press **F4**, and then press **F5** to select the **Supplier_Code**.
b. Press **F4**, and then press **F5** to select the **Product_Code**.



#	Name	Type	Primary Key
1	ProductCode	Unique	☐
2	SupplierCode	Unique	☐

#	Column	Name	Size	Order
1	4	Supplier_Code	4	Ascending
2	1	Product_Code	4	Ascending

#	Name	Attribute	Size
1	Product_Code	Numeric	4
2	Product_Name	Alpha	60
3	Description	Alpha	100
4	Supplier_Code	Numeric	4
5	Product_Price	Numeric	8
6	Stock_Quantity	Numeric	4

Before continuing, you need to add data to the **Products** data source. The explanation for this is in the **Exercise** section of the lesson.

Creating the Products Program

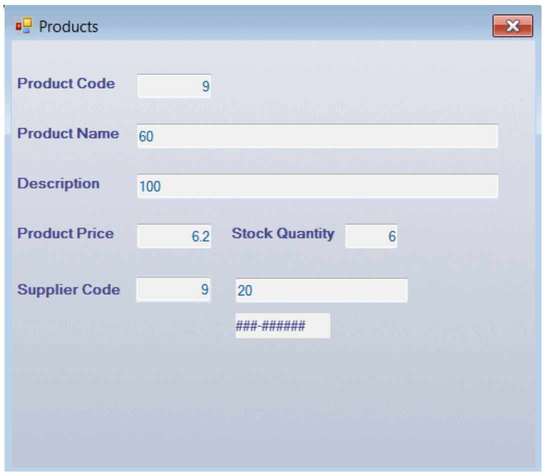
Create the **Products** program according to the following steps:

1. Set the **Task type** to **Rich Client**.
2. Set the **Initial Mode** to **Query**.
3. Define the **Products** data source as the program's Main Source.
4. Select all of the **Products** data source columns.
5. Add a **Link Query** operation to the **Suppliers** data source using the **Supplier_Code** index.
6. Select the following additional columns: **Supplier_Name** and **Phone_Number**.
7. From the **Supplier_Code** column, set the **Supplier_Code** (from the **Products** data source) as the Locate **From** and Locate **To** expressions.

Defining the Products Form

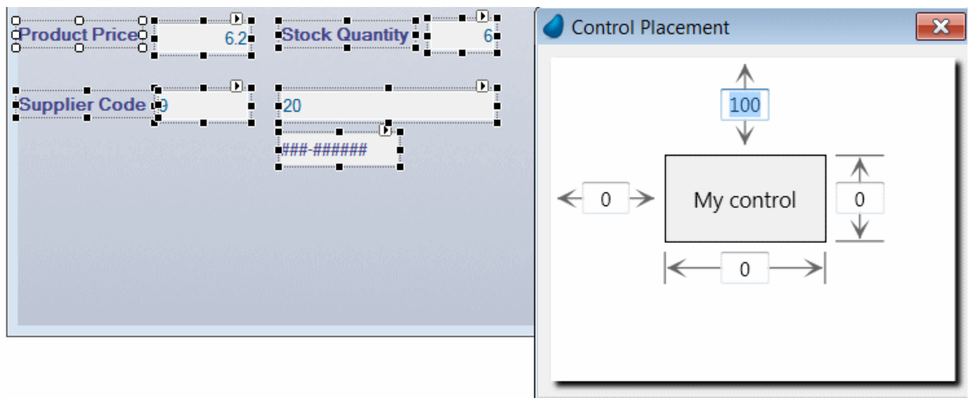
1. Set the form's model to **Table Display Form**.
2. Place all of the **Products** data source columns on the form, as well as the **Supplier_Name** (to the right of the Supplier Code) and the **Phone_Number** from the **Suppliers** data source.
3. Attach all Label controls to the **Text Caption** model.
   You can mark all of them by continuously pressing the **Ctrl** button and clicking on each Label control. Then, go to the **Model** property and select the **Text Caption** model.
4. Right-click on the **Font** property for each of the Label controls and select **Inherit**.
5. Fit the size of the Label controls to the displayed text.
6. Move the Edit controls to a position in which the Label controls are fully displayed.
7. Attach all Edit controls from the **Products** data source to the **Edit Control** model.

8. Attach all Edit controls from the **Suppliers** data source to the **Display only** model.



Now you need to define placement for the variables.

9. Select the following controls, both Label and Edit: **Product Price**, **Stock Quantity**, and **Supplier Code**. Select also the **Supplier Name** and the **Supplier Number**.
10. Zoom into the control properties. Zoom into the **Placement** property and enter **100** in the **Y** property.



The Placement value will be: **0,0,100,0**.

The controls' **Product Name** and **Description** both need to increase in height and width. Because the controls are displayed one underneath the other, they both have to share the increase in height. They both will then increase only 50%.

11. Select the **Product_Name** Edit control and zoom into the control properties.

12. Zoom into the **Placement** property and enter **100** in the **Width** property and enter **50** in the **Height** property. The Placement value will be: **0,100,0,50**.
13. Select the **Description** Label control and zoom into the control properties.
14. Zoom into the **Placement** property. As the **Product_Name** increases in height by 50%, the **Description** must move downwards by 50%. Enter **50** in the **Y** property. The Placement value will be: **0,0,50, 0**.
15. Select the **Description** Edit control and zoom into the control properties.

This must move downwards in the same way as its Label control moved. It also increases in height by 50%.

16. Zoom into the **Placement** property and enter **100** in the **Width** property and enter **50** in the **Height** property.
17. Enter **50** in the **Y** property. The Placement value will be: **0,100,50,50**.

This program will be called from a program that you will create soon; therefore you need to create a parameter.

18. Zoom into the Data View Editor.
19. Add a parameter named **P.Product Code**, which will be based on the model for **Code**.
20. Park on the **Product code** column of the **Products** data source and zoom into the **Range From** property. Set the following expression: **CndRange (P.Product code > 0, P.Product Code)**.
21. Enter the same expression for the **Range To** property.

You will now add the button enabling you to edit the product:

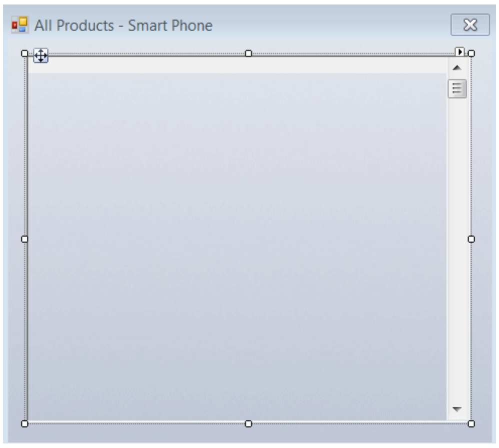
1. Place a Button control at the top of the form.
2. Zoom into the control properties.
3. Park on the **Format** property and remove the text, which is initially **Button**.
4. Park on the **Button style** property and select **Image Button** from the combo box.
5. Park on the **Image List file name** property and type **%WorkingDir%\images\Edit.png**.
6. From the **Event Type** property, select **Internal**.
7. From the **Event** property, select **Modify Records**.
8. Set the **Color** to **Text Caption**. This will make the button transparent for images that have a transparent color.
9. Click the **Fit Conrol Size** icon.

Now you need to create the **All Products** program. This is a simple program and you have created many similar programs.

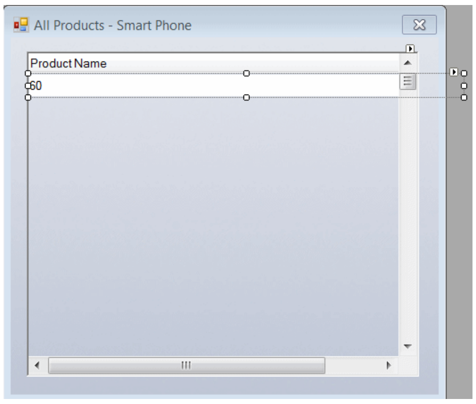
1. Create the **All Products** program.
2. Zoom into the **All Products** program and in the **Task Properties** dialog box, set the **Task type** to **Rich Client**.
3. Set the **Initial Mode** to **Query**.
4. Define the **Products** data source as the program's Main Source.
5. Select the **Product_Code** and the **Product_Name** columns.

Now you will define the form:

1. Zoom into the Form Designer.
2. Set the form's model to **Table Display Form**.
3. Drop a Table control onto the form and set the model to **Table**.
4. Increase the size of the table so that it fills the default form:



5. Drop the **Product_Name** variable onto the table. The variable is too big to fit on the table:



6. Decrease the width of the Edit control so that it fits in the table. There will still be a horizontal scrollbar.
7. Set the **Model** to **Edit control**.
8. Open the Column properties.
9. Set the **Model** to **Column Header**.
10. You need to manually decrease the **Width** property of the column so that it fits in the table. Enter the same width as the Edit control, approximately 68 units.

The Table control is defined so that it increases only in height. You need to increase it horizontally as well:

11. Park on the Table control and zoom into the control properties.
12. Zoom into the **Placement** property and enter **100** in the **Width** property.

Now, when the user taps a row in the table, you need to call the **Products** program. Remember that a tap raises the **Click** internal event.

1. Zoom into the Logic Editor.
2. Add a header line and select the **Click** internal event.
3. Zoom from the **on:** property and select the **Product Name** control.
4. Create a Details line for the **Call Program** operation.
5. Select the **Products** program from the list.
6. Zoom from the **Arguments** property.
7. In the **Arguments** dialog box add a line, zoom from the **Variable List** and select the **Product Code** variable.

Lesson 13 – Selecting Data from a List

The following exercise summarizes what you learned in this lesson.

First you will create the two models:

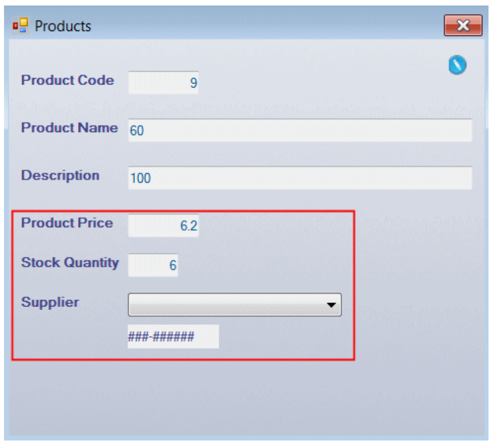
1. Zoom into the Model repository.
2. Create a line and in the **Name** column, type: **Browse button**.
3. From the **Class** column, select **Rich Client Display**.
4. From the **Attribute** column, select **Push button**.
5. Zoom into the Model properties.
6. Park on the **Format** style and type … (three dots).
7. Zoom into the **Raise Event** property and the **Event** dialog box opens.
8. In the **Event type** combo box, select **Internal**
9. Zoom into the **Event List** and select the **Zoom** event.
10. Create a line and in the **Name** column, type: **Image button**.
11. From the **Class** column, select **Rich Client Display**.
12. From the **Attribute** column, select **Push button**.
13. Park on the **Format** property and remove the text, which is initially **Button**.
14. Park on the **Button style** property and select **Image Button** from the combo box.
15. Park on the **Image List file name** property and type **%WorkingDir%\images\Edit.png**.
16. Zoom into the **Raise Event** property and the **Event** dialog box opens.
17. In the **Event type** combo box, select **Internal**.
18. Zoom into the **Event** list and select **Zoom**.
19. Set the **Color** to **Text Caption**.

This will make the button transparent for images that have a transparent color.

Add an Image to the Products Program

You will now add an image to the **Products** program:

1. Zoom into the **Products** program.
2. Zoom into the Form Designer.
3. Move the **Supplier** details downwards and move the **Stock Quantity** below the **Product Price** to make way for the Image control as shown in the image on the right.


Products

Product Code 9

Product Name 60

Description 100

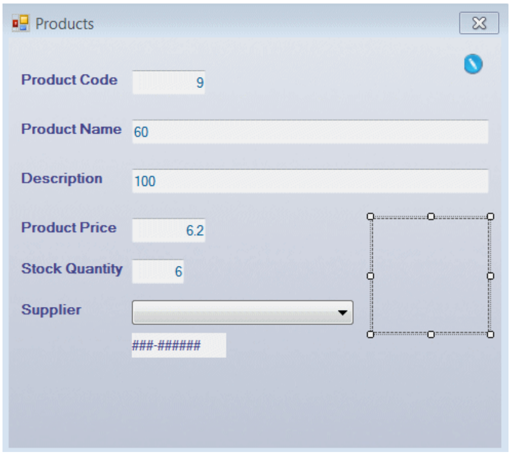
Product Price 6.2

Stock Quantity 6

Supplier

###-######

4. From the Toolbox, select the Image control .
5. Place the Image control on the form next to the **Product_Price**.
6. Open the Image Properties.
7. Expand the **Data** property, zoom from the **Expression** line and create an expression: **'%WorkingDir%'&'products\'&Trim (Product_Name)&'.jpg'**
8. From the **Style** property, select **No Border**.
9. From the **Color** property, select **Text Caption**. This is to ensure that the image is transparent on the background.
10. From the **Image style** property, select **Scaled to Fit**.



Take into account that you previously defined placement for the other controls. In this case you would want the image to move together with the product price.

11. Zoom into the **Placement** property and set **100** in the **Y** property.

What changes would you make if you wanted the width and height of the image to increase along with the form? Hint: You will also need to make changes to other controls as well.

Creating the Products Selection List

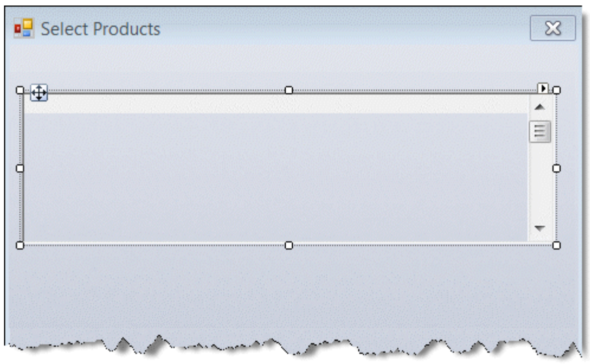
This selection list will be similar to the other selection list but you will also display the image in the table.

1. In the Program repository, create a program named **Select Products**.
2. From the **Task type** property, select **Rich Client**.
3. From the **Initial mode** property, select **Query**.
4. From the **Selection table** property, select **Yes**.
5. In the Data View Editor, set the **Main Source** to **Products**. Use the **Product_Code** index.
6. Add a parameter, **P.Product Code**, and attach it to the **Code** model.
7. Add the **Product_Code** and the **Product_Name** columns from the **Products** data source.
8. Set the **P.Product Code** parameter as the **Locate from** property expression for the **Product_Code**.

9. In the Logic Editor, create a header line and set the logic unit type to **Record**. Set the **Level** to **Suffix**.
10. Add an **Update** operation in which you update the **P.Product Code** parameter with the **Product_Code**.

Now you will define the **Products** selection list's form.

1. Open the Form Editor.
2. Attach the form to the **Table Display Form** model.
3. Zoom to the form.
4. Drop the Table control on the form. Set the **Model** property to **Table**.
5. Increase the width of the table so that it fills the form width as shown in the image below:



6. Select the **Customer_Name** variable and place it on the table. Select the **Edit control** model.
7. Since the width of the control is too wide for the table, you need to decrease it. Set the **Width** property to **62**.
8. Press **Alt+Click** on the **Customer_Name** column and select the **Column Heading** model. Set the **Width** property to **62**.

Now you are going to add the image to the table:

9. From the Toolobx, drop the Image control on the table.
10. Open the Image Properties.
11. Expand the **Data** property, zoom from the **Expression** line and create an expression: **'%WorkingDir%'&'products\'&Trim(Product_Name)&'.jpg'**

Magic xpa searches for images relative to the working directory. Therefore you can also define the expression as: **'\products\'&Trim(Product_Name)&'.jpg'**. During runtime, Magic xpa copies the images to the cache of the device.

12. From the **Style** property, select **No Border**.
13. From the **Color** property, select **Text Caption**.
14. From the **Image style** property, select **Scaled to Fit**.

The image needs to be a thumbnail, so you can decrease the width and height:

15. Set the **Width** property to **5**.
16. Set the **Height** property to **1.5**.
17. Press **Alt+Click** on the **Image** column and select the **Column Heading** model. Set the **Width** property to **5**.
18. Increase the height of the table to the end of the form.

You may find that after adding the image to the form, the form property's row height was changed. If that happens, zoom into the Table properties and set the **Row Height** property to **1.875**.

Adding a Select Button

As was mentioned before, the unique behavior of a Selection List program is achieved when the internal **Select** event is raised.

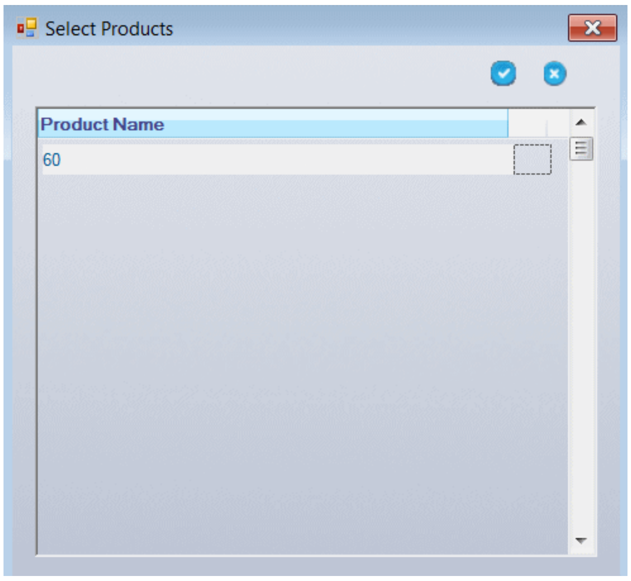
1. Place a Button control at the top of the form.
2. Zoom into the control properties. Set the **Model** to the new **Image button** model that you defined.
3. Park on the **Image List file name** property and change the image to **OK.png**.
4. From the **Event Type** property, select **Internal**.
5. From the **Event** property, select **Select**.
6. Click the **Fit Command Size** icon.

Adding the Exit Button

Add the exit button.

1. Place a Button control at the top of the form.
2. Zoom into the control properties. Set the **Model** to the new **Image button** model that you defined.
3. Park on the **Image List file name** property and change the image to **Exit.png**.
4. From the **Event Type** property, select **Internal**.
5. From the **Event** property, select **Exit**.
6. Click the **Fit Control Size** icon.

The form will look similar to the image below.



Since the image is small, you can decide whether to add functionality so that when the user taps a row, the **Products** program is called.

Lesson 14 – One-to-Many Data Relationships

Countries and **cities** have a one-to-many data relationship. Each **country** can have many cities.

Now you will practice what you learned so far, by creating a program that displays a list of countries and their related cities.

Defining the Country&City Code Model

1. Open the Model repository.
2. Create a line.
3. In the **Name** column, type: **Country&City Code**
4. In the **Attribute** column, select **Numeric**.
5. In the property sheet, set the **Picture** to **6**.
6. In the **Style** node, park on the **Rich Client table** property. This is the property that governs the look and feel of this variable when it is dropped on a table.
7. Click the Zoom … button. This opens a control property sheet.
8. Set the Model to Edit Control.

Defining the Countries Data Source

1. In the Data repository, create a line.
2. In the **Name** and **Data source name** columns, type: **Countries**.
3. In the **Database** column, zoom to select **Getting Started**.
4. From the **Options** menu, select **Get Definition**.

The definition of the **Countries** data source has now been successfully imported.

5. In the Column repository, park on the **Country_Code**.
6. Zoom from the **Model** column and select the **Country&City Code** model.
7. Re-inherit the **Picture** property.

Defining the Cities Data Source

1. In the Data repository, create a line.
2. In the **Name** and **Data source name** columns, type: **Cities**.
3. In the **Database** column, zoom to select **Getting Started**.
4. Use the **Get Definition** utility to import the definition of the **Cities** data source.
5. Set the **Country_Code**'s model property to the **Country&City Code** model and re-inherit the **Picture** property.
6. Set the **City_Code**'s model property to the **Country&City Code** model and re-inherit the **Picture** property.

Creating the Countries Program

1. Open the Program repository.
2. Create a line and name the program: **Countries**.
3. Zoom into the program.
4. Define the **Task type** as **Rich Client**.
5. Set the **Initial mode** to **Query**.
6. Set the **Main Source** to **Countries**.
7. Set the **Main Source index** to **Country_Code**.
8. Create a line and select all of the **Countries** columns (2 columns).

Designing the Countries Form

1. Open the Form Editor. Set the **Model** to **Table Display Form**.
2. Zoom to the **Countries** form.
3. Add a Table control. Set the **Model** to **Table**.
4. Add the **Country_Code** variable to the table.
5. Click on the column and open the control properties.
6. Set the **Model** to **Column Header**.
7. Set the **Column Title** property to **Code**.
8. Set the **Width** to **9.5**.
9. Drop the **Country_Name** variable on to the table.
10. Set the **Model** to **Edit control**.
11. Set the **Width** to **40**.
12. Click on the column and open the control properties.
13. Set the **Model** to **Column Header**.
14. Set the **Column Title** property to **Country**.
15. Set the **Width** to **41**.
16. Increase the width of the table so that there is no horizontal scrollbar.
17. Increase the height of the table to fill the form.

Creating the **Cities** Subtask:

1. From the Navigator pane, park on the **Countries** task and create a subtask.
2. In the **Task name** property, type: **Cities**.
3. Define the **Task type** as **Rich Client**.
4. Set the **Initial Mode** to **Query**.

In the Data View Editor:

1. Create a line.
2. Create a **P.Country_Code** parameter with the **Country&City Code** model.
3. Set the **Main Source** to **Cities**.
4. Set the **Main Source index** to **Country_and_City**.
5. Create a line and select all of the **Cities** columns.
6. Park on the **Country_Code** column (of the **Cities** data source) definition line.
7. In the **Range from** and **To** properties, set the **Country_Code** parameter.
8. Open the Form Editor.
9. Park on the **Cities** form.
10. Open the Form Properties and set the **Model** to **Table Display Form**.
11. Remove the expression in the **Wallpaper** property.
12. Set the **Color** property to **Text caption**.
13. Zoom to the **Cities** form.
14. Add a Table control. Set the **Model** to **Table**.
15. Add the **City_Code** variable to the table.
16. Click on the column and open the control properties.
17. Set the **Model** to **Column Header**.
18. Set the **Column Title** property to **Code**.
19. Set the **Width** to **9.5**.
20. Drop the **City_Name** variable on to the table.
21. Set the Model to Edit control.
22. Set the **Width** to **33**.
23. Click on the column and open the control properties.
24. Set the **Model** to **Column Header**.
25. Set the **Column Title** property to **City**.
26. Set the **Width** to **34**.
27. Increase the width of the table so that there is no horizontal scrollbar.
28. Increase the height of the table to fill the form.

Designing the Countries Form

1. Using the Navigation pane, navigate to the **Countries** task.
2. Open the Form Designer.
3. Add a Subform control to the form.

4. Open the Subform Control Properties.
5. From the **Connect to** property, select **SubTask**.
6. From the **PRG/TSK num** property, select the **Cities** subtask.
7. Zoom from the **Arguments** property and select the **Country_Code** column.
8. Increase the Subform control so that it fits the form.
9. Close the program and save the changes.

Lesson 16 – Reports

You were asked to print a single invoice from the **List of Orders** program.

To print a specific invoice, you need to pass the order number as an argument.

1. Zoom to the **Invoice** program.
2. In the Data View Editor, park on the first line and create a **Parameter** line.
3. Type in the name **P.Order_Number** and set the parameter type to **Numeric** and set the **Picture** to **6**.
4. Park on the **Order_Number** column in the **Orders** data source.
5. From the **Range** entry, zoom to the Expression Editor and create an expression for the **P.Order_Number** parameter.
6. Type the same expression for the **To** entry.
7. Close the program and save the changes.

Creating the Print Event

In the **List of Orders** program, you will add the **Print** handler. The **Print** event handler will call the **Invoice** program and pass the **Order_Number** as a parameter.

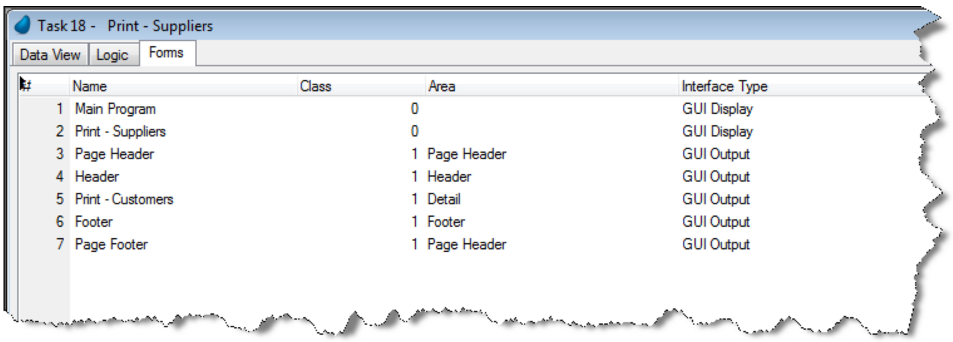
1. Zoom to the **List of Orders** program.
2. Add a Virtual variable named **PDF Name** with an attribute of **Alpha** and a length of **255**.
3. Open the Logic Editor.
4. Park on the last line in the editor.
5. Create an **Event** logic unit for the **Print** event.
6. Zoom to the **Event** dialog box.
7. From the **Event type** property, select **User**.
8. From the **Event** property, zoom and select the **Print** event.
9. Park on the **Event** line and create a line.
10. Set the **Call Program** operation to call the **Invoice** program.
11. From the **Arguments** entry, zoom to the Argument repository and create a line.
12. Zoom from the **Var** column and pass the **Order_Number** column from the Variable list.
13. Add a detail line and update the **PDF Name** variable with the expression: **ServerFileToClient('%TempDir%Invoice.pdf')**
14. Add a detail line and call the **Display PDF** program.
15. Zoom into the Argument repository and add a line.
16. Zoom into the **Var** column and select the **PDF Name** variable.

17. Open the Form Editor and zoom to the form.
    18. From the **Models** pane, drag the **Image button** model and drop it on the table.
19. Open the control property sheet and change the image file name to **Print.png**.
20. From the **Raise Event** property, select **User** from the **Event type** field and select the **Print** user event that you defined in the Main Program.
21. Click the **Fit Control Size** icon.

You can now run the application and view the results. Remember that this is valid for iOS and Desktop.

You were asked to create a Suppliers List report in the same way that you created the Customers List report.

1. In the Program repository, create a program: **Print Suppliers**.
2. Zoom (**F5**) to the program. The **Task Properties** dialog box will open.
3. In the **Task type** property, select **Batch**.
4. Set the **Initial mode** to **Query**.
5. Click **OK**.
6. In the Data View Editor, set the **Suppliers** data source as the task's **Main Source**.
7. Select the **Suppliers** data source's columns: **Supplier_code**, **Supplier_Name**, **Address**, **Phone_Number**, in that order.
8. From the **Task** menu, select **I/O Devices** (Ctrl+I).
9. Add a line and define a printer named **Print Suppliers**.
10. In the **Exp/Var** column, zoom and set the following expression: **'%TempDir%SupplierList.pdf'**
11. Open the Form Editor.
12. Define the forms 3-7 as shown in the image below:

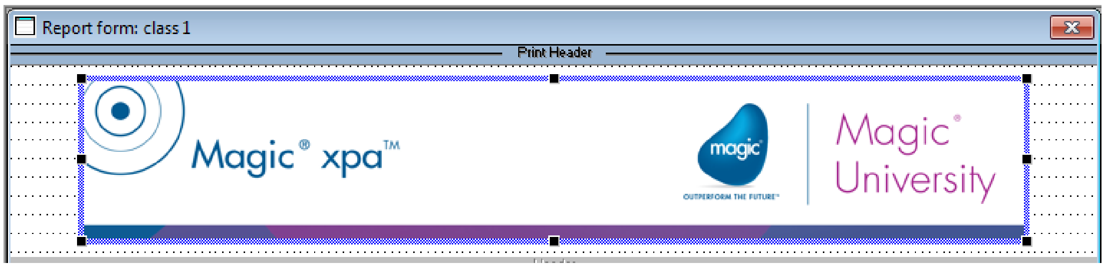


#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	Print - Suppliers	0		GUI Display
3	Page Header	1	Page Header	GUI Output
4	Header	1	Header	GUI Output
5	Print - Customers	1	Detail	GUI Output
6	Footer	1	Footer	GUI Output
7	Page Footer	1	Page Header	GUI Output

13. For each form use the **GUI Print Form** model.

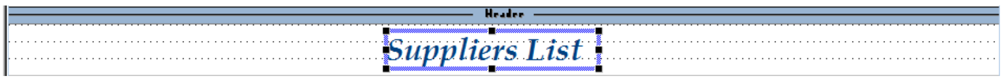
Page Header Form

1. Drop an image on the form and set the name to **%WorkingDir%images\Print_Logo.jpg**.
2. Set the **Image style** to **Scaled to Fit**.
3. Set the **Width** of the image to **17.4**.
4. Set the **Height** to **3**.
5. Remember to use **Center Horizontally** and **Center Vertically**.



Header Form

1. Drop a Text control on the form and set the text to **Suppliers List**.
2. Set the **Font** to **Report Header**.
3. Set the **Color** to **Report Header**.

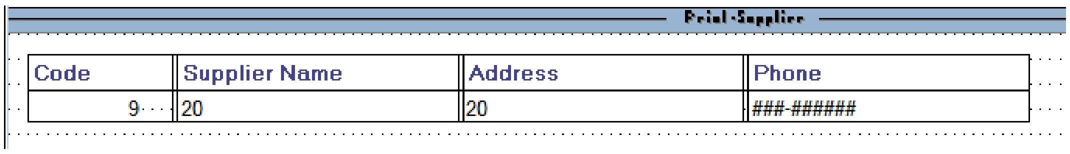


Print Suppliers Form

1. Zoom to the **Print - Supplier** form.
2. From the Control palette, select a Table control and place it on the form.
3. Change the **Height** property to **1**.
4. Select the following variables, one by one, from the Variable palette and place them on the Table control: **Supplier_Code**, **Supplier_Name**, **Address** and **Phone_Number**.
5. For each column:

a. Press **Alt+Click on** the column field.
b. In the Column properties (**Alt+Enter**), from the **Font** property, zoom and select the **Text Caption** font.
c. For the **Supplier_Code** column, change the title to **Code**.
d. For the **Phone_Number** column, change the title to **Phone**.

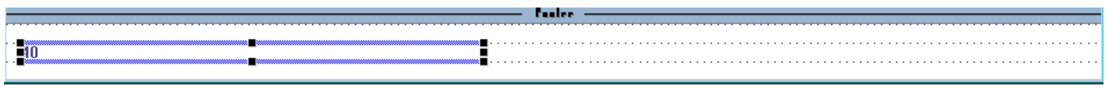
e. In the **Color** property, zoom and select the **Text Caption** color.



Code	Supplier Name	Address	Phone
9	20	20	###-######

Footer Form

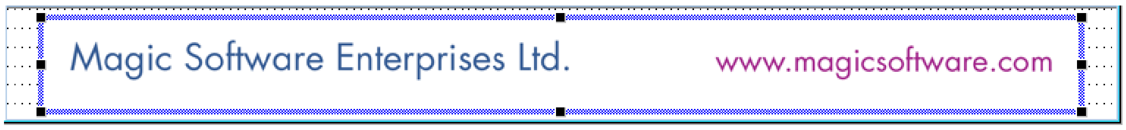
1. From the Control palette, select an Edit control and place it on the form.
2. Open the Edit control properties and from the **Data** property, enter a new expression: **'Total Number of Suppliers: '&Trim(Str(Counter(0),'6'))**
3. In the **Format** property, type: **40**.
4. In the **Font** and **Color** properties, select **Text Caption**.
5. Park on the Edit control and move the control to the left.
6. In the Command palette, click the **Fit Size** ( ) command icon.
7. Click the **Vertical center of form** ( ) command icon. This will center the Edit control on the form



The Page Footer Form

1. Drop an image on the form and set the name to **%WorkingDir%images\Print_Footer.jpg**.
2. Set the **Image style** to **Scaled to Fit**.
3. Set the **Width** of the image to **18.76**.
4. Set the **Height** to **1.67**.

Remember to use **Center Horizontally** and **Vertically**.



Printing the Forms

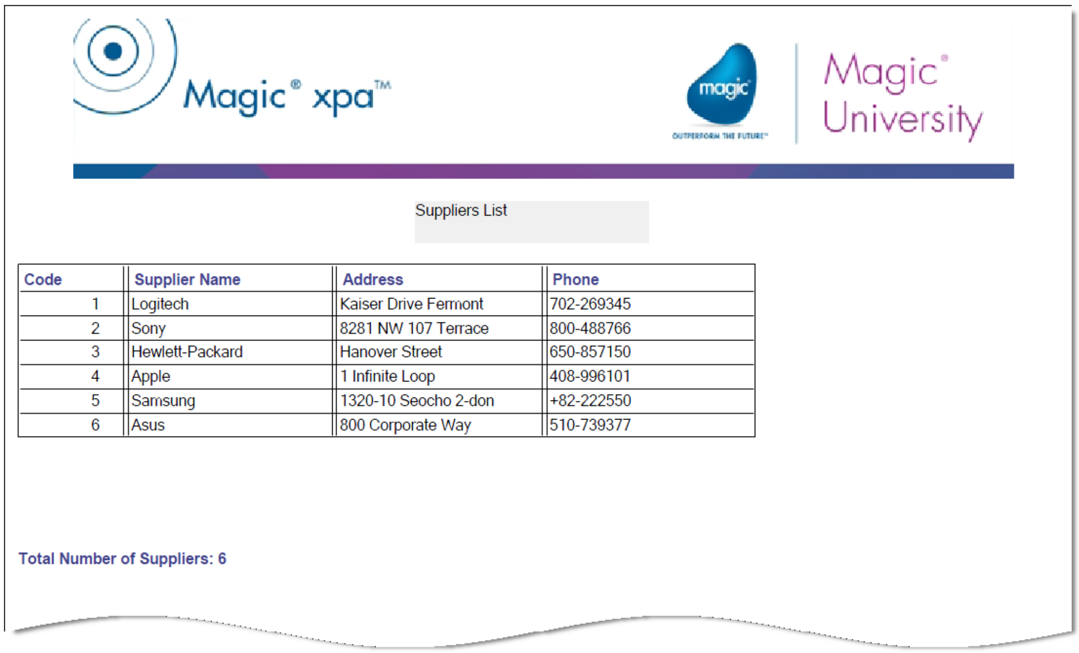
1. In the Logic Editor, from the **Task Prefix** logic unit, print the **Header** form.
2. From the **Record Suffix** logic unit, print the **Print Suppliers** form.
3. From the **Task Suffix** logic unit, print the **Footer** form.
4. From the **Task** menu, select **I/O Devices** (Ctrl+I).
5. Open the **Print Suppliers** I/O properties (**Alt+Enter**).
6. In the **Page Header form** property, set the **Page Header** form (#3).
7. In the **Page Footer form** property, set the **Page Footer** form (#7).
8. In the **PDF** property, select **Yes**.
9. Click **OK**.
10. Close the program and save the changes.

You need to call the **Print Suppliers** program from the **Suppliers - Line Mode** program as you did during the lesson.

In this course you are defining reports for a few programs each with its own Print handler. Instead of defining the handler in each program, you can define it in the Main Program. The definition will then be available for all the programs.

1. Zoom to the **Suppliers - Line Mode** program.
2. Add a **Virtual** variable named **PDF Name** with an attribute of **Alpha** and a length of **255**.
3. Open the Form Editor and zoom to the form.
4. From the Control palette, select the Push Button control and drop it on the form. Select the **Image button** model.
5. Open the control property sheet and change the image file name to **Print.png**.
6. From the **Raise Event** property, select **User** from the **Event type** field and select the **Print** user event that you defined in the Main Program.
7. Use the **Fit to Size** icon on the palette.
8. Zoom into the Logic Editor.
9. Add a header line for the **Print** User event.
10. Add a detail line and call the **Print - Suppliers** program.
11. Add a detail line and update the **PDF Name** variable with the expression: **ServerFileToClient('%TempDir%SupplierList.pdf')**
12. Add a detail line and call the **Display PDF** program.
13. Zoom into the Argument repository and add a line. Zoom into the **Var** column and select the **PDF Name** variable.

You can now run the application and view the results. Remember that this is valid for iOS and Desktop.



Suppliers List

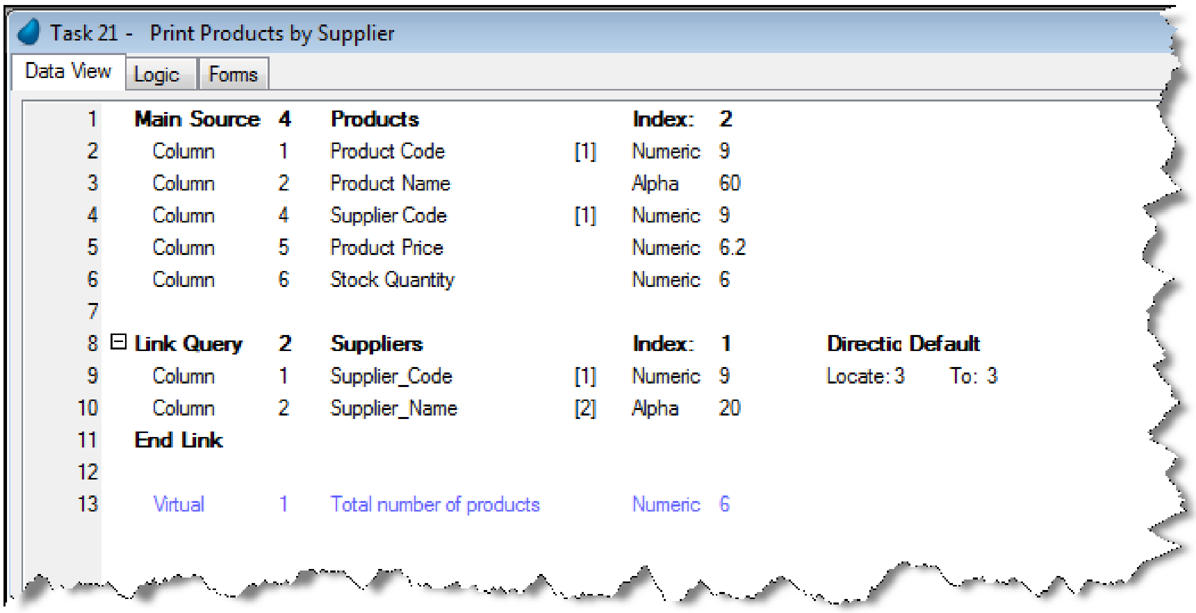
Code	Supplier Name	Address	Phone
1	Logitech	Kaiser Drive Fermont	702-269345
2	Sony	8281 NW 107 Terrace	800-488766
3	Hewlett-Packard	Hanover Street	650-857150
4	Apple	1 Infinite Loop	408-996101
5	Samsung	1320-10 Seocho 2-don	+82-222550
6	Asus	800 Corporate Way	510-739377

Total Number of Suppliers: 6

Lesson 17 – Processing Data by Groups

You were asked to create a report that displayed each supplier and a list of its products. You will use the Group logic unit.

1. Create a program: **Print Products by Supplier**.
2. In the **Task Properties** dialog box, from the **Task type** property, select **Batch**.
3. In the **Initial mode** property, select **Query**.
4. Define the data view in the same way as the image below:



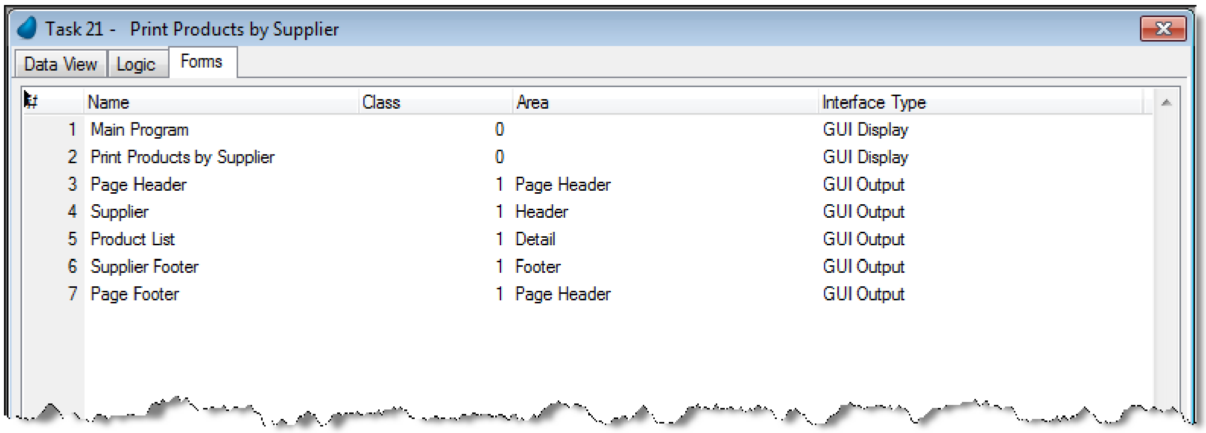
Task 21 - Print Products by Supplier

Data View | Logic | Forms

1	**Main Source**	**4**	**Products**		**Index:**	**2**		
2	Column	1	Product Code	[1]	Numeric	9		
3	Column	2	Product Name		Alpha	60		
4	Column	4	Supplier Code	[1]	Numeric	9		
5	Column	5	Product Price		Numeric	6.2		
6	Column	6	Stock Quantity		Numeric	6		
7								
8	⊟ **Link Query**	**2**	**Suppliers**		**Index:**	**1**	**Directic Default**	
9	Column	1	Supplier_Code	[1]	Numeric	9	Locate: 3	To: 3
10	Column	2	Supplier_Name	[2]	Alpha	20		
11	**End Link**							
12								
13	Virtual	1	Total number of products		Numeric	6		

Creating the Task Forms

Use the image below as a guideline for setting the task forms. You have created similar forms in the last two lessons:


Task 21 - Print Products by Supplier

Data View | Logic | Forms

#	Name	Class	Area	Interface Type
1	Main Program	0		GUI Display
2	Print Products by Supplier	0		GUI Display
3	Page Header	1	Page Header	GUI Output
4	Supplier	1	Header	GUI Output
5	Product List	1	Detail	GUI Output
6	Supplier Footer	1	Footer	GUI Output
7	Page Footer	1	Page Header	GUI Output

1. The **Page Header** form and the **Page Footer** forms are the same as previous reports.
2. Set the **Supplier Header** form to a height of 1cm.
3. In the **Supplier Header** form, place an Edit control with the expression: **'Product list for: '&Trim(Supplier_Name)**
4. Set the **Format** property to **44**.
5. Set the **Font** property to **Report Header**.
6. Set the **Color** property to **Report Header**.
7. Set the **Product List** form to a height of **2.2**cm.
8. Zoom to the **Product List** form.
9. From the Control palette, select a Table control and place it on the form
10. Select the **Product_Code**, **Product_Name**, **Product_Price** and **Stock_Quantity** variables and place them on the Table control.
11. Mark the **Product_Code** column and set the **Color** to **Text Caption** and the **Font** to **Text Caption**.
12. Set the **Column Title** property to **Code**.
13. Mark all of the other columns and set the **Color** to **Text Caption** and the **Font** to **Text Caption**.
14. Mark the **Product_Price** column and set the **Column Title** property to **Price.**
15. Mark the **Stock_Quantity** column and set the **Column Title** property to **Stock.**

16. Set the **Supplier Footer** form to a height of **2**cm.
17. Zoom into the Supplier Footer form and place an Edit control with the expression:
    'Total number of products for '&Trim (Supplier Name)&': '&Trim(Str(J,' Total num_Products'))
18. Set the **Format** to **60**.
19. Set the **Font** property to **Text Caption**.
20. Set the **Color** property to **Text Caption**.

Defining the I/O Device

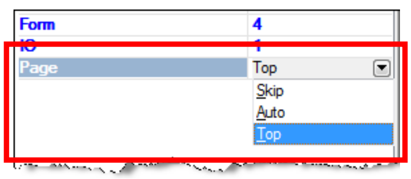
1. From the **Task** menu, select **I/O Devices** (Ctrl+I).
2. Add a line and set the **Name** to **Products by Supplier**.
3. From the **Exp/Var** column, zoom and set the following expression:
   '%TempDir%SupplierProductList.pdf'
4. Open the **I/O Properties** dialog box.
5. From the **Page header form** property, zoom to the Form list and select the **Page Header** form.
6. From the **Page footer form** property, zoom to the Form list and select the **Page Footer** form.
7. Set the **PDF** property to **Yes**.

Defining the Task Logic

1. Open the Logic Editor and create a header line.
2. Set a **Group Prefix** logic unit for the **Supplier_Code** variable (from the **Products** data source).
3. Add a **Form Output** detail operation and print the **Supplier Header** form.

You were asked to print each new supplier on a new page:

4. Zoom into the properties of the **Form** operation and select **Top** from the **Page** property. This will print this form at the top of each page.
5. Add an **Update Variable** detail operation and update the **Total Number of Products** variable with zero.



Form	4
IO	1
Page	Top

Now you will print out the detail form for each product in the group. In addition, you need to increase the counter variable value by one, for each city in the group.

6. Create a **Record Suffix** logic unit.
7. Add a **Form Output** detail operation and print the **Product List** form.

8. Add an **Update Variable** detail operation and update the **Total Number of Products** variable with **Total Number of Products +1**. This increases the counter of the number of products.

Now you need to print the footer showing how many products there are for that supplier. This is performed in the Group Suffix logic unit.

9. Create a **Group Suffix** logic unit for the **Supplier_Code** variable.
10. Add a **Form Output** detail operation and print the **Supplier Footer** form

You can now run the program.

Lesson 20 – Offline Implementation

You are asked to implement the situation where the client updates the server.

1. Zoom into the **Offline Countries** program that you created in this lesson.
2. Add a **Rich Client** subtask and name it **Add Country**. Uncheck the **Offline** box.
3. Set the Initial mode to Create.
4. Select **Local Countries** as the Main source and select all three records.
5. Add a Virtual variable named **Last synchronization timestamp** and link it to the **Timestamp** model. You will use this for the synchronization.
6. Zoom into the Logic Editor and add a **Record Suffix** logic unit.
7. Add an **Update Variable** operation and update the **Last Modified** variable with the following expression: **DStr (Date (),'YYYYMMDD')&TStr (Time(),'HHMMSS')** This will create a string concatenating the date and time.
8. Add a User event named Submit Country to Server. Set the Force Exit property to Post record update.

By using **Post Record Update,** the engine first writes the modified record to the database and then executes the logic units of the triggered event.

9. Zoom into the form and display only the **Country_Code** and the **Country_Name** variables. You can use the **Pop Up** form property to **True**.
10. Add a Button control to the form named **Submit to Server**, which raises the User event named **Submit Country to Server**.

Now you will update the calling program, the **Offline Countries**, so that it will call the **Add Country** subtask.

1. Return to the **Offline Countries** program and add a User event named **Add Country**.
2. Zoom into the Logic Editor and add an **Event** logic unit for the **Add Country** User event.
3. In the **Add Country** logic unit, call the **Add Country** subtask.
4. Raise the **View Refresh** internal event.
5. Zoom into the Form Designer and place a Button control at the top of the form.
6. Zoom into the control properties. Set the **Model** to the **Image button** model.
7. Park on the **Image List file name** property and change the image to **Add.png**.
8. From the **Event Type** property, select **User**.
9. From the **Event** property, select **Add Customer**.
10. Click the **Fit Control Size** icon.

The **Offline Countries** program is an Offline program and needs the program images so that they can be displayed. As you learned in the lesson, these need to be copied to the client:

1. Zoom into the **Start** program.
2. Park on the **Block If** line and add a line after it. Within the block, it does not matter where you add the line.
3. Add an **Evaluate Expression** operation and use the expression: **ServerFileToClient('%WorkingDir%images')**

Synchronization logic

You now need to recreate the logic that you used in the **Start** program. You learned that synchronizing data from the server to the client involved the following steps:

- Fetch the last timestamp that a successful synchronization was performed.
- Use the **DataViewToDataSource** function to synchronize the data according to the timestamp.
- Update the timestamp with the current date and time.

You will now create the program that performs the synchronization:

1. Zoom into the Program repository and create a program named **Sync Countries from client**.
2. Set the **Task type** to **Rich Client**.
3. Uncheck the **Interactive** box to make this a non-interactive task.
4. Set the **End task condition** to **Yes**.
5. Set the **Evaluate condition** to **Before entering record**.
6. Zoom into the task and select **Local Countries** as the **Main source**.
7. Select all of the fields from the **Local Countries** data source.
8. Add a **Parameter** named **P.Last Sync** with a model of **Timestamp**.
9. Park on the **Last modified** column, zoom into the **Range from** property, and set the expression: **P.Last Sync**. This ranges all of the entries since the last synchronization.
10. Zoom into the Logic Editor and add a **Task Suffix** logic unit.
11. Add an **Evaluate Expression** operation: **DataViewToDataSource (0, 'Country_Code,Country_Name,Last Modified','6'DSOURCE, '','')**

    '6'DSOURCE is the **Countries** data source

Now you need to call this program from the Main Program because Offline programs cannot call server programs.

1. Zoom into the Main Program.
2. Zoom into the Logic Editor and then zoom into the **Sync countries** logic unit.
3. Add a line and use a **Call** operation to call the **Sync countries from client** program.
4. Zoom into the **Arguments** property, add a line and select the **P.Current Timestamp** parameter.
5. Set the condition on the line to **Not (P.Sync from Server?)**.

Now you need to retrieve the last timestamp and call the synchronization program:

1. Zoom into the **Offline Countries** program and zoom into the **Add Country** subtask.
2. Add a subtask named **Fetch last sync data** and uncheck the **Interactive** box.
3. Set the **End task condition** to **Yes** and the **Evaluate condition** to **After updating record**.
4. Zoom into the **Data View Editor** and set the Main source to the **Local sync table**.
5. Select all of the columns from this table.
6. Add a **Record Suffix** logic unit, add an **Update** operation and update the **Last synchronization timestamp** variable from the parent task with the **Last Modified Country** column from this subtask.

You will now add the update subtask of the last successful timestamp.

1. Return to the **Add Country** subtask.
2. Add a subtask named **Update last sync data** and uncheck the **Interactive** box.
3. Set the **End task condition** to **Yes** and the **Evaluate condition** to **After updating record**.
4. Zoom into the Data View Editor and add a **Link Write** operation to the **Local sync table**. Remember that with a **Link Write** operation, if no record exists, one will be added. In the course example, there is already a record in the table.
5. Select the **Idx** column and set an expression with the value **1** for the **Locate from** and **Locate to** columns. Use the same expression for the **Init** expression.
6. Add the **Last Modified Country** column as well.
7. Add a **Record Suffix** logic unit, add an **Update** operation and update the **Last Modified Country** column with the **Last synchronization timestamp** variable from the parent task.

You are now ready to use the methodology.

1. Return to the **Add Country** subtask.
2. Zoom into the Logic Editor and add an event logic unit for the **Submit Country to Server** event.
3. Add a **Block If** operation and set the expression to **ServerLastAccessStatus()=0**. This means that if there is a problem with the server, the Block operation will be ignored.
4. Add a line and define a **Call** operation that calls the **Fetch last sync data** subtask.
5. Add a **Raise Event** operation and select the **Sync countries** User event.
6. Zoom into the **Arguments** property, add a line and select the **Last synchronization timestamp** variable to match the **P.Current Timestamp** parameter.
7. Add a second line, zoom into the **Exp** column and create an expression of **False** to match the **P.Sync from Server** parameter. This means that the logic unit in the Main Program will handle the event and call the **Sync Data from Client** program.

If you have a situation in which you have multiple client devices, then the server table may also have been updated. You can also add a call to sync the client data from the server:

8. Add a **Raise Event** operation and select the **Sync countries** User event.
9. Zoom into the **Arguments** property, add a line and select the **Last synchronization timestamp** variable to match the **P.Current Timestamp** parameter.
10. Add a second line, zoom into the **Exp** column and create an expression of **TRUE** to match the **P.Sync from Server** parameter.
11. Add an **Update Variable** operation and update the **Last synchronization timestamp** variable with the following expression: **DStr (Date (),'YYYYMMDD')&TStr (Time(),'HHMMSS')**
    This will create a string concatenating the date and time.
12. Add a **Call** operation after the **Update** operation that updates the **Last synchronization timestamp** variable and call the **Update last sync data** subtask.
13. After the **Block End** operation, add a line and raise the **Exit** internal event.

Remember that when you clicked the **Submit to Server** button, there may not have been server access and as a result, the server was not updated with the client data. As a result, you could also perform a client to server synchronization in the Start program process.

1. Zoom into the **Start** program.
2. Park on the line with the **Raise Event** operation for the **Sync countries** and add a line after it.
3. Add a **Raise Event** operation and select the **Sync countries** User event.
4. Zoom into the **Arguments** property, add a line and select the **Last Country timestamp** variable to match the **P.Current Timestamp** parameter.
5. Add a second line, zoom into the **Exp** column and create an expression for **FALSE** to match the **P.Sync from Server** parameter.

You can now execute the **Start** program and from the **Offline Countries** program, add **Mexico** as country number **25**.