OUTPERFORM THE FUTURE™

Magic® University

Getting Started with Magic xpa 3.x

Self-Paced Tutorial

Created Date: June 1,2016
Updated Date: August 09,2016
Source: Getting Started with Magic xpa 3.X and Mobile(Self-Paced Tutorial)

HRD Singapore PTE. LTD

Magic®
University

www.magicsoftware.com

Contents

Introduction 1
About Magic xpa 1
About the Course 2
Course Prerequisites 2
How to Use this Guide 3
Exercises 3
Course Materials 4
Entity Relations Diagram (ERD) 4

Magic xpa Studio Interface 5
Creating a New Project 6
The Navigator Pane 8
The Checker Result Pane 10
The Comment Pane 10
Summary 12

Creating Your First Program 13
Opening the Getting Started Project 14
The Project Source Files 14
The Application Properties 15
Creating a Program 16
What Is a Task? 19
Checking the Program 28
Running the Program 28
Executing on Your Mobile Device 30
About Magic xpa Attributes 31
About the Magic xpa Picture 33
Exercise 34
Summary 35

Architecture Overview 37
Mobile Deployment 38
Summary 40

Data Manipulation and Validation 41
Numeric Data Manipulation 42
Logic Editor 43
Alphanumeric Data Manipulation 51
Magic xpa Internal Data Validation 55
Developer Validation 57
Data Consistency 59
Data Consistency – Short Summary 63
Parking Condition 64
Exercise 66
Summary 67

Setting Initial Values 69
Update in Task Prefix 70
Variable Initialization 71
Update in Control Prefix 73
Exercise 75
Summary 76

Setting the Form's Appearance 77
Adding Colors to the Color Repository 78
Adding Fonts to the Font Repository 81
Wallpaper 85
Placement 89
Exercise 94
Summary 95

Viewing Data Source Content 97
Defining the Database 98
Defining a Data Source 102
Automatic Program Generation 105
Manually Creating the Customers Program 107
Short Summary 112
Viewing Several Records 113
Task Mode 113
Creating a Line Mode Program 114
Exercise – Suppliers Line Mode Program 120
Summary 121

Models 123
What Is a Model? 124
Advantages to Defining Models 124
The Inheritance Mechanism 125
Field Class Models 126
Display Class Model 127
Assigning Models to Objects 128
Using a Field Model in a Task 129
Exercises 131
Summary 132

The Application Engine Concept 133
Event-Driven Development Concept 134
The Task 134
Task Execution Stages 135
The Task Execution Logic Units 136
Execution Rules 139
Summary 143

Events and Handlers 145
Events and Handlers Concept 146
Types of Events 146
Raising Internal Events 147

User-Defined Events....149
Handlers....152
Event Checking....154
Having More than One Handler for the Same Event....155
Handling Internal Events....157
Exercise....162
Summary....163

Conditioning a Block of Operations....165
What Is a Block Operation?....166
Block If – Conditioning Operations....166
Advantages of the Block Operation....170
The Block While Operation....171
Exercise....173
Summary....174

One-to-One Data Relationships....175
One-to-One vs. One-to-Many Data Relationships....176
Linking to Other Data Sources....177
Link Header Line....177
Link Operation Usage....178
Link Types....179
Using Link Query....180
Short Summary....188
Link Recompute Mechanism....188
Link Success Indication....188
Exercise....191
Summary....194

Selecting Data from a List....195
Selection List....196
Data Control....201
Exercises....204
Summary....205

One-to-Many Data Relationships....207
One-to-Many Data Relationship Preface....208
Defining the Many Data Source....211
Establishing the One-to-Many Data Relationship....212
Subform Control....217
More About the Subform Control....223
Incremental Update....225
Exercise....229
Summary....230

Non-Interactive Processing....231
Data View Editor and Rich Client....232
Client-Side vs. Server-Side Operations....234
The Task Life Cycle....235
Identifying Client and Server Activity....236

Rich Client Operation Colors 237
Batch Programming 238
Rich Client Task vs. Batch Task 238
Engine Flow for a Batch Task 239
Batch Task Behavior 241
Batch Delete 242
Summary 245

Reports 247
Using the Program Generator Utility 248
Manually Creating a Report 249
Print - Customers Form 251
Designed Report 254
The Browser Control 261
Displaying a PDF on an Android Tablet 264
Printing a PDF on a Mobile Device 265
Complex Report Concept 265
Exercise 273
Summary 274

Processing Data by Groups 275
About the Group Logic Unit 276
Group Logic Unit Execution Order 276
Engine Flow for a Batch Task 277
Sorting the Data 278
Exercise 283
Summary 284

Menus 285
The Menu Repository 286
Context Menu 287
Menu Properties 289
Application Menus 290
Summary 292

Using the Device Functionality 293
Fetching the Device Orientation 294
Accessing the Camera 297
Telephone, SMS, Mail and HTTP 298
Using the GPS 299
Multiple Forms 302
Summary 305

Offline Implementation 307
Concept 308
How Does It Work? 309
Local (Offline) Storage 310
Synchronization Programs 311
Offline Programs 313
Synchronization Issues 315

Server Access Failure ... 319
Deleting a Record ... 322
Offline Images ... 322
Exercise ... 323
Summary ... 323

Best Practices ... 325
Models ... 326
Code Reuse ... 326
Logical Names ... 327
One-to-Many Forms ... 328
Defining Different Forms ... 328
Manager Program ... 328
Performance Enhancements ... 329
Summary ... 332

Customization and Installation ... 333
Creating a Cabinet File ... 334
Setting Up the Server ... 335
Setting Up the Web Server ... 336
Rich Client Folders ... 336
Customizing the Application ... 336
Directly Executing on an Android Device ... 342
Using Native OS Code in Mobile Apps ... 343
Fonts ... 346
Summary ... 349

Solutions ... 351
Lesson 4 – Data Manipulation ... 353
Lesson 5 – Initializing a Variable ... 355
Lesson 6 – Setting the Form's Appearance ... 357
Lesson 7 – Viewing Data Source Content ... 360
Lesson 8 – Models – Object Definition Centralization ... 363
Lesson 10 – Events and Handlers ... 367
Lesson 11 – Conditioning a Block of Operations ... 371
Lesson 12 – One-to-One Data Relationships ... 373
Lesson 13 – Selecting Data from a List ... 385
Lesson 14 – One-to-Many Data Relationships ... 390
Lesson 16 – Reports ... 394
Lesson 17 – Processing Data by Groups ... 400
Lesson 20 – Offline Implementation ... 404

Introduction

About Magic xpa

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**

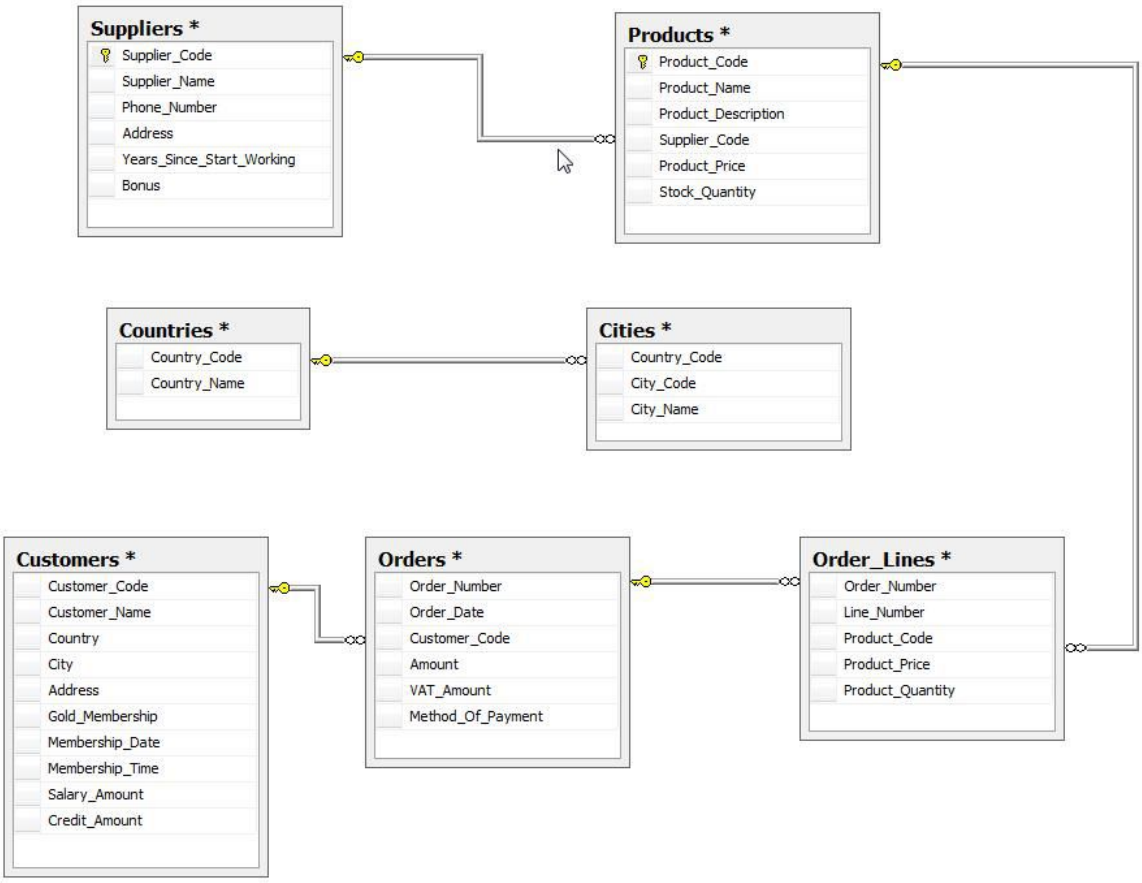
About the Course

In this course you will:

- Learn the fundamentals of Magic xpa and how to get the best out of Magic xpa.
- Become familiar with the Magic xpa Studio interface.
- Get to know the Magic xpa wizards and utilities.
- Understand the Magic xpa concepts and standards.
- 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.

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

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.

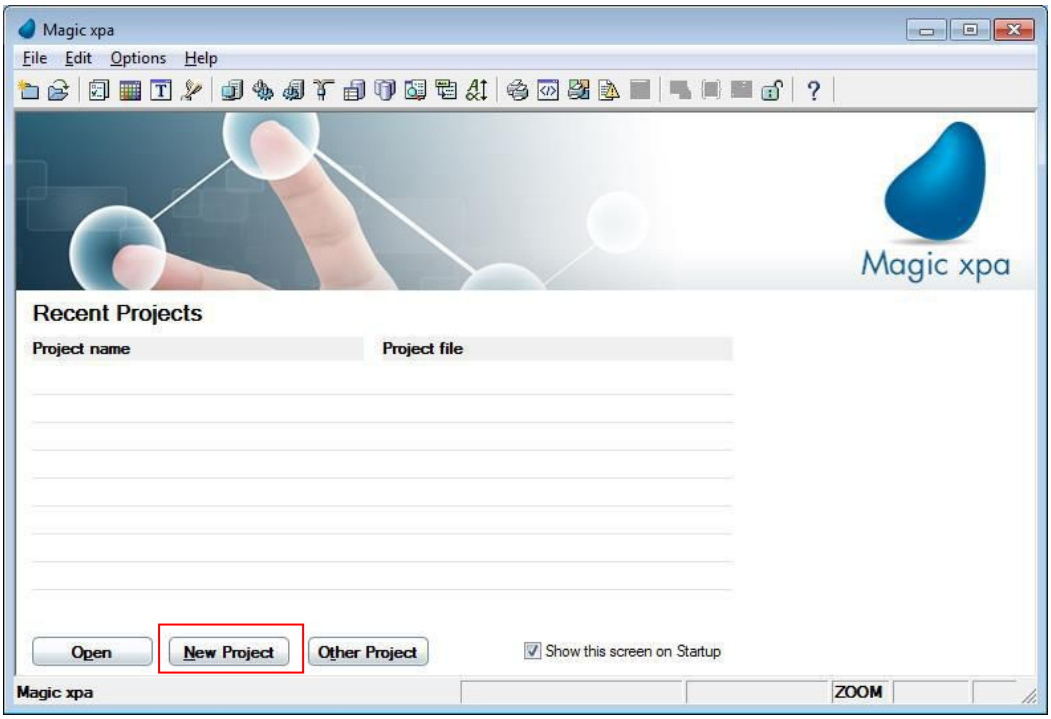
By the end of this lesson you will:

- 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.

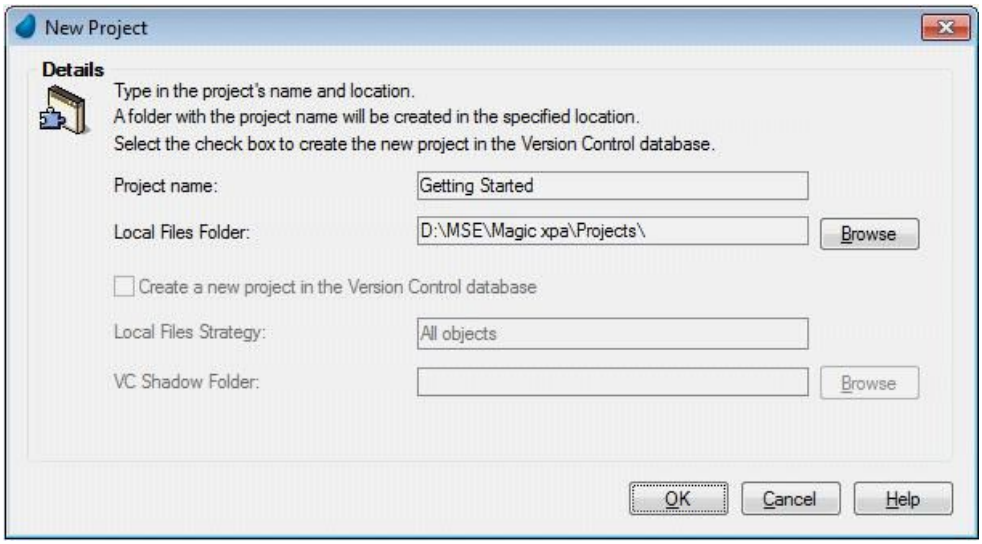
Creating a New Project

Now, you will create the **Getting Started** project..

1. Open the Magic xpa Studio.
   Notes: On your desktop, use the shortcut: "Magic xpa 3.0 For Training" Or use this shortcut:"C:¥XPA¥Magic xpa 3.0 For Training"
2. In the Startup screen, click the **New Project** button.



The **New Project** dialog box opens.



3. In the **Project name** entry, type **Getting Started**.
4. The **Local Files Folder** entry has a default value of:

   [Magic xpa path]\Projects

   Notes : Specifically, this is"C:¥Program Files¥MSE¥Magic xpa 3.0 Single User Edition¥Projects".

   Use the Browse button if you want to specify a different location.

   Notes: In HRD, use this folder."C:¥XPA¥Apps¥"
5. Click **OK**.

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

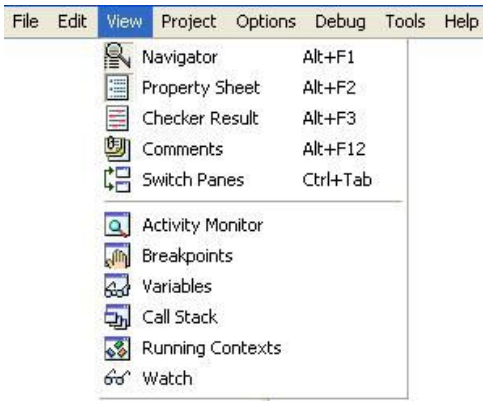
The Navigator Pane

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.

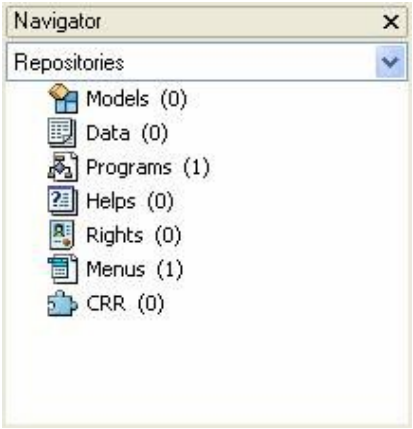


The first entry in the Navigator pane is a selection box, which enables you to display one of five categories.

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.



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

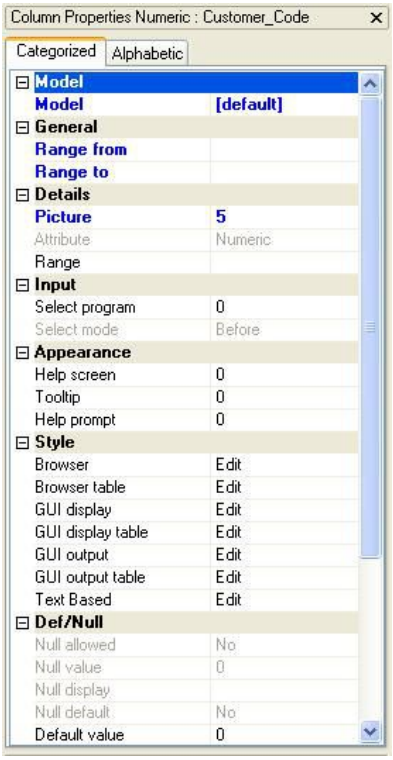
The Property Sheet

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.

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).**

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.

Notes:

Before the next lesson,close the project.
From the toolbar,select "Close the current project".

Magic University

Lesson 2

Creating Your First Program

In this lesson you will learn more about the project concept and you will create your first program.

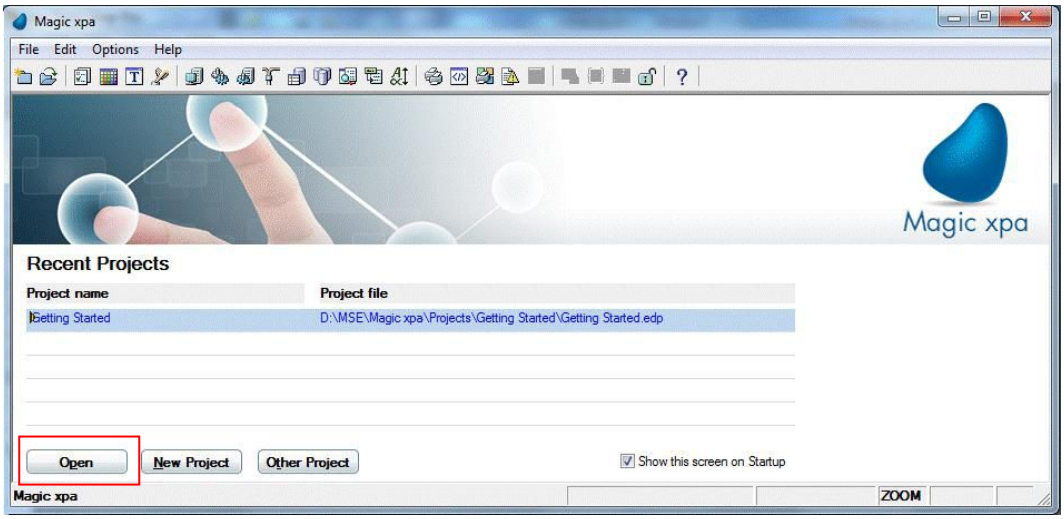
By the end of this lesson you will be:

- 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.

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, Select the **"Getting Started"** project then click the **Open** button.



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.

Creating a Program

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

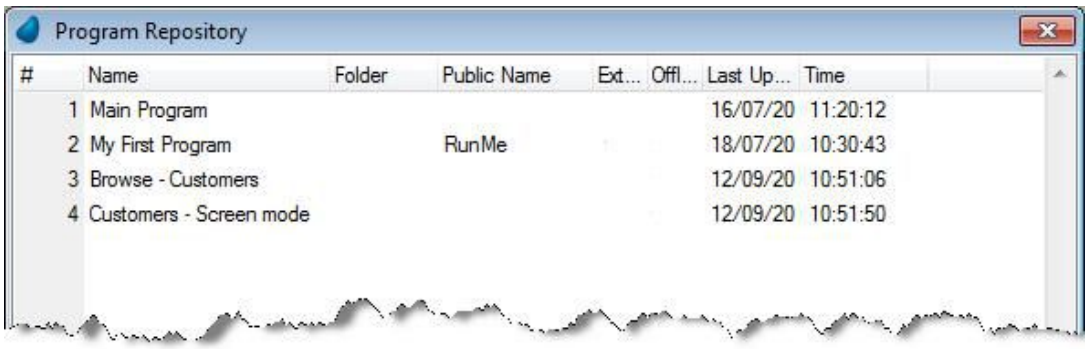
The program creation process consists of the following steps:

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

The Program Repository(Shift + F3)

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.

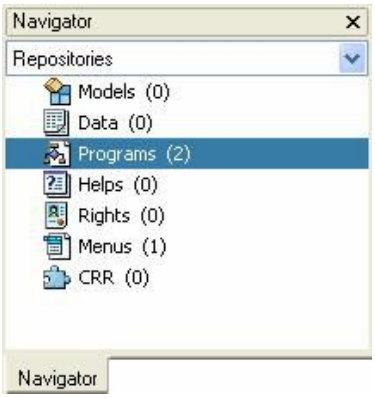


The following table describes each column in the Program repository.

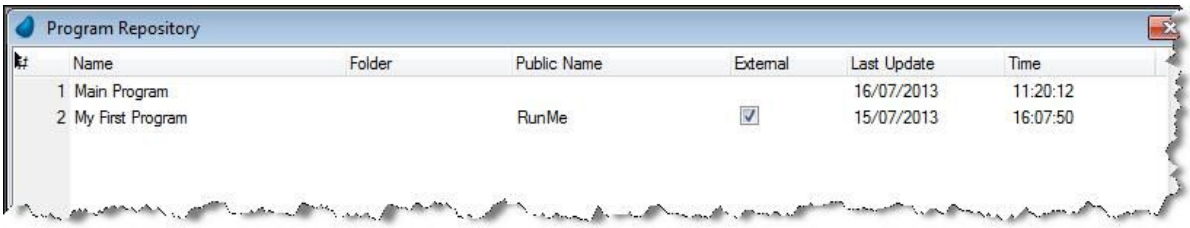
Column	Description
#	A program identifier number that is generated automatically by Magic xpa. When you edit the Program repository by adding, deleting rows or moving a program from one place to another, Magic xpa automatically recalculates the number of the programs and updates their identifier numbers. 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.
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**.



What Is a Task?

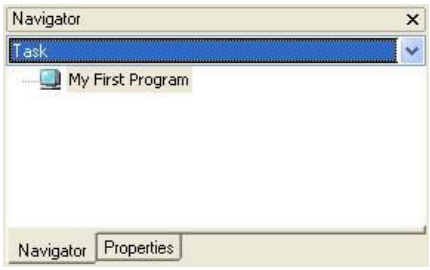
Both of the terms **program** and **task** are used to describe an application task in Magic xpa.

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.

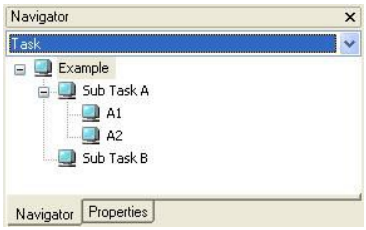


In the image to the right, you can see an example of another task tree:

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.

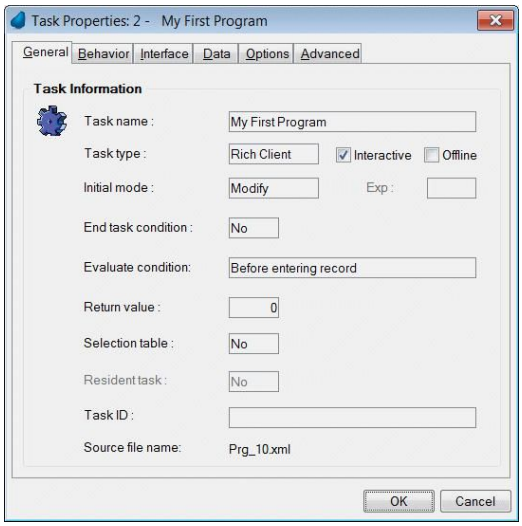


The Task Properties

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(F5)** 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).**



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.

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

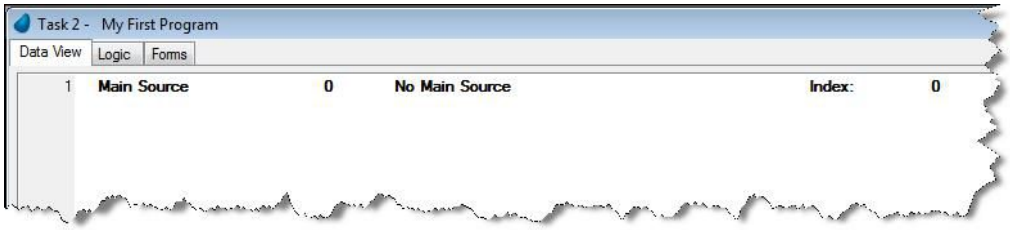
5. Click **OK** to close the **Task Properties** dialog box.

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



- 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:

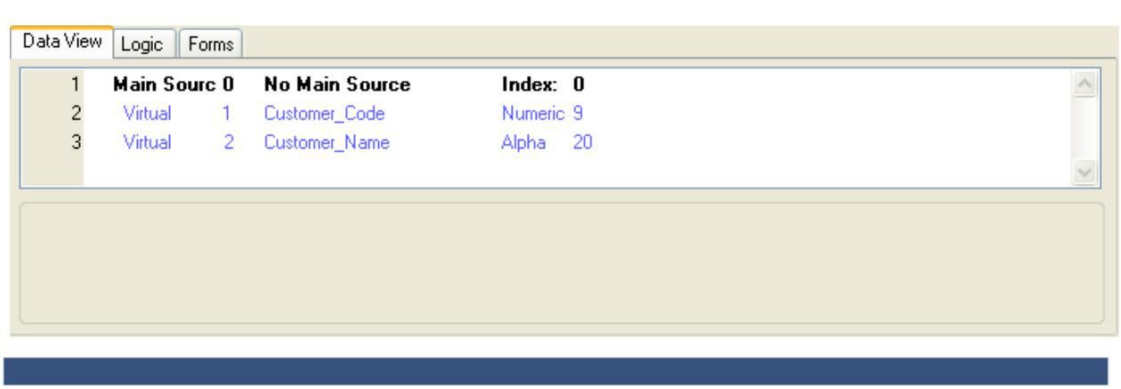
- **Parameter** – Variables which receive values from a different program
- **Column** – Data object columns defined within a database table

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 (**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	TV:Customer_Code	Numeric	9
Virtual	TV:Customer_Name	Alpha	20



The Task Forms

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.

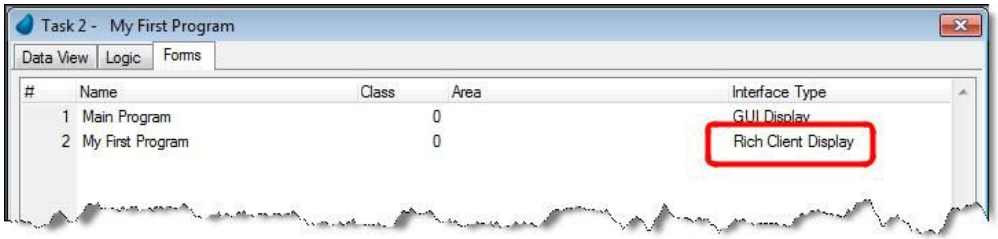
The task's main form cannot be deleted or moved to another position in the repository.

Main Form

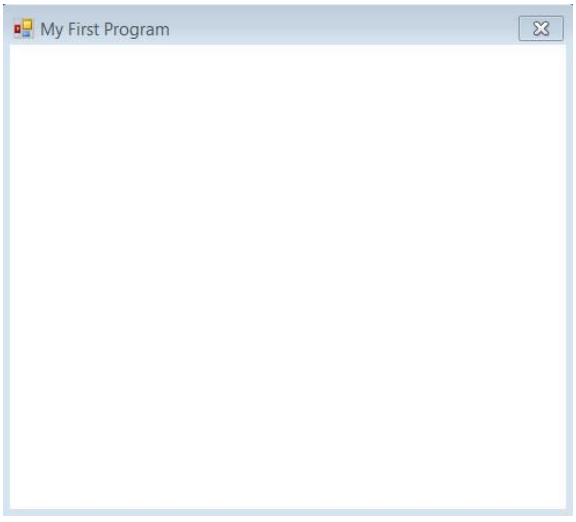
Now you will define the task's main form.

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

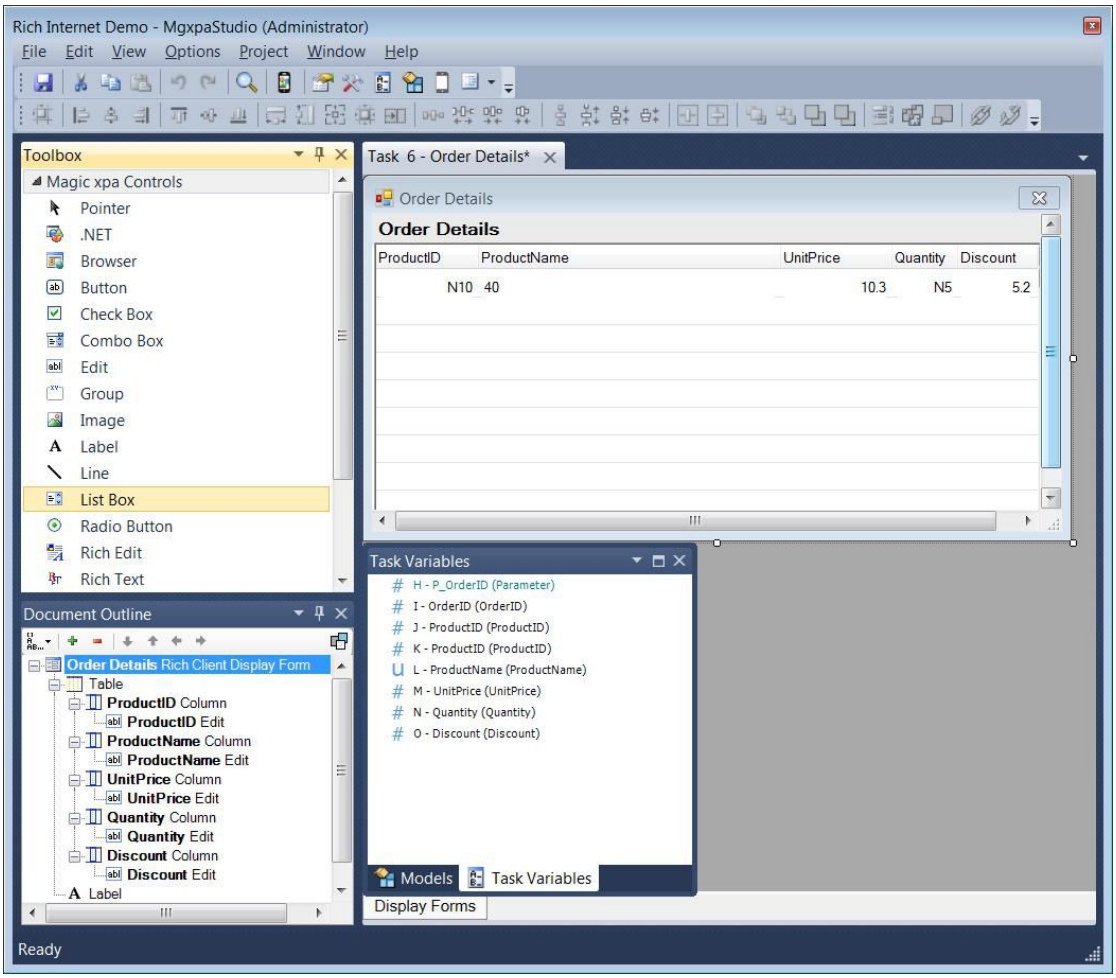
11. Park on the **My First Program** form entry and **zoom** (**F5**) or double click.



My First Program Form will appear.



The Form Designer



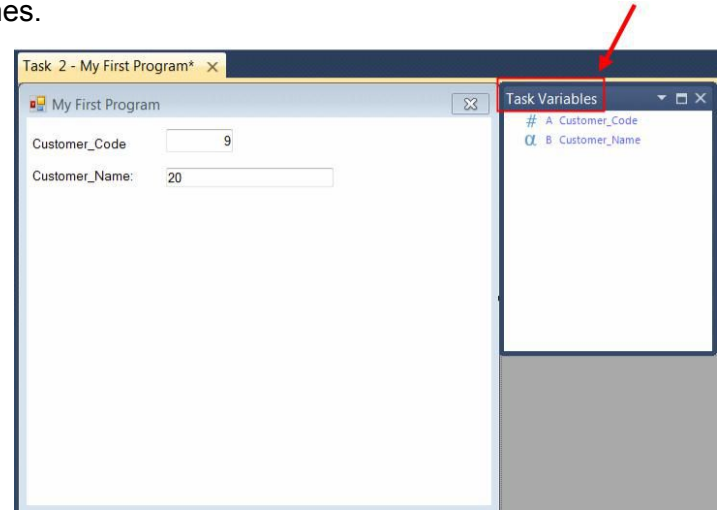
The Form Designer includes:

- 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.
- The **Task Variable pane** and **Model pane**. This shows you all of the variables and models in your data view.

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.

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



Saving the Program

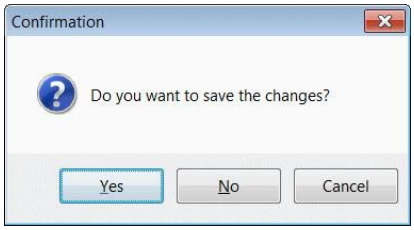
You have created a new program and defined its data view and form.

Now you will save and exit your first program:

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

15. 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.

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.

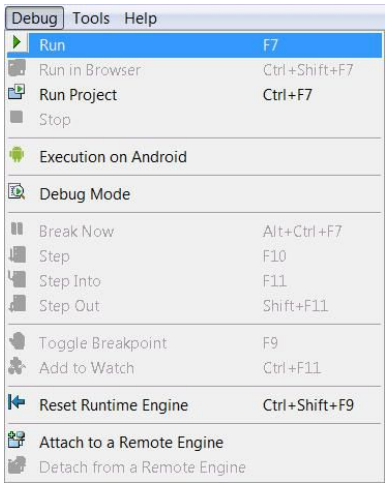
If there are mistakes, the errors are listed in the **Checker Result pane**.

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.

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



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

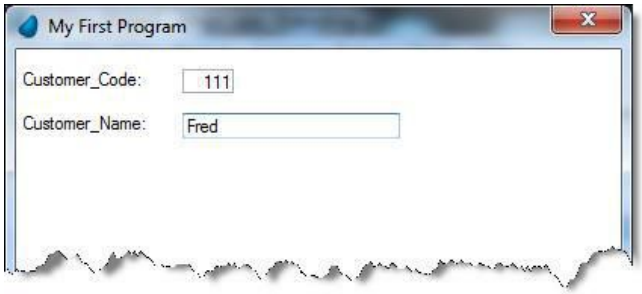
To execute a program, you can press **F7** or select the Run icon from the toolbar:

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

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

You have just typed in values in the **TV:Customer_Code** and **TV: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.



Internal Data Validation – Numeric Variable Example

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

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

For example if you type the word **Barney** in the **TV:Customer_Code** variable you will not see any text because the input was rejected by the engine.

Picture Limitation Example

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

In this example, the **TV: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 **TV: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 **TV:Customer_Code** variable.
   Magic xpa will accept the first 9 digits and ignore the rest.
2. Type more than 20 characters in the **TV:Customer_Name** variable.
   Magic xpa will accept the first 20 characters and ignore the rest.

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 and Time	In Magic xpa, a Date value and a Time value are not saved in one variable. Not like in SQL Server we can a lot 1 column for date and time. 1 Column for date and 1 column for time.

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

Data Manipulation and Validation

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

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.

Operations

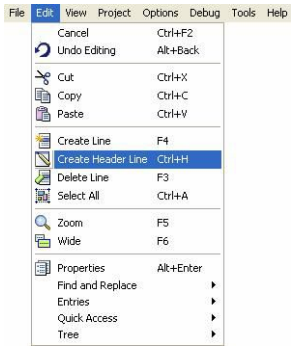
The operations are listed below:

- **Remark** (default) – Enables you to enter a description of the section ofcode.
- **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.

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(Ctrl+H)**.
4. Define a **Control Suffix or (Press C)** operation.
5. Zoom from the **of:** field and the Control list opens.
6. Park on the **Salary_Amount** control.
7. Click the **Select** button.

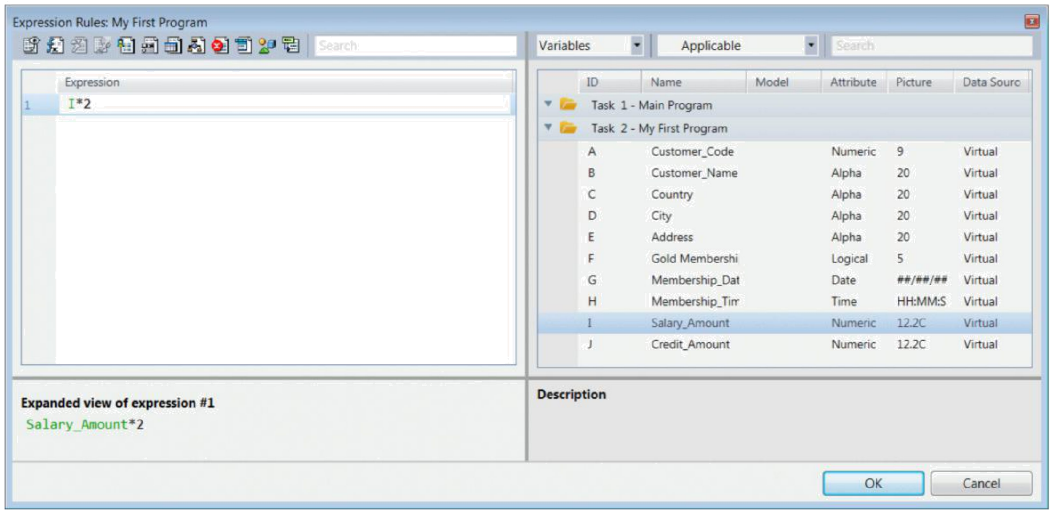
Expression Editor

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.
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



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.

10. **Save** and **close** the program.

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

11. Check if no error by pressing F8 then execute the project by pressing **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**.

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**.

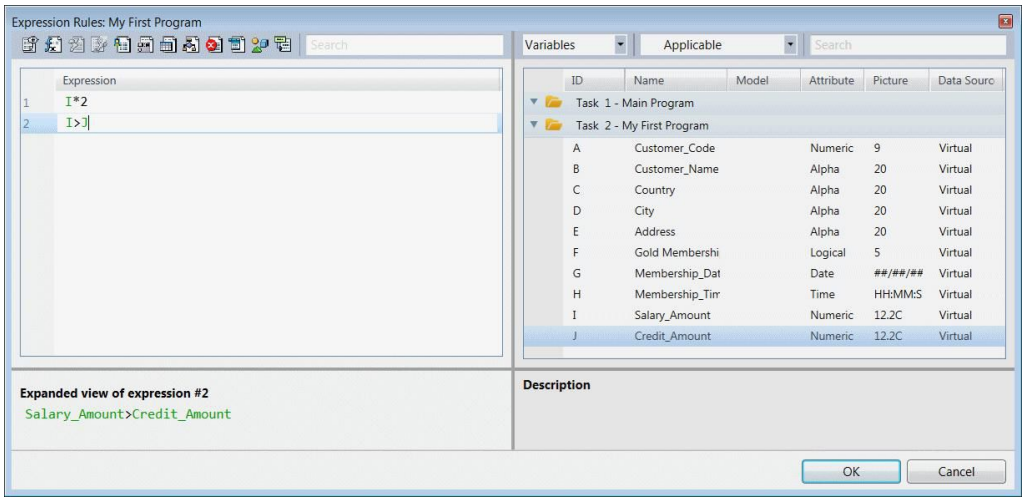
In the same way, you can use all of the other arithmetic operators that were mentioned in the beginning of this lesson.

Conditional Calculations

The **condition(Cnd) i**s 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**).



7. Save and close the program.
8. Execute the project by pressing **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 Internal "IF" Function

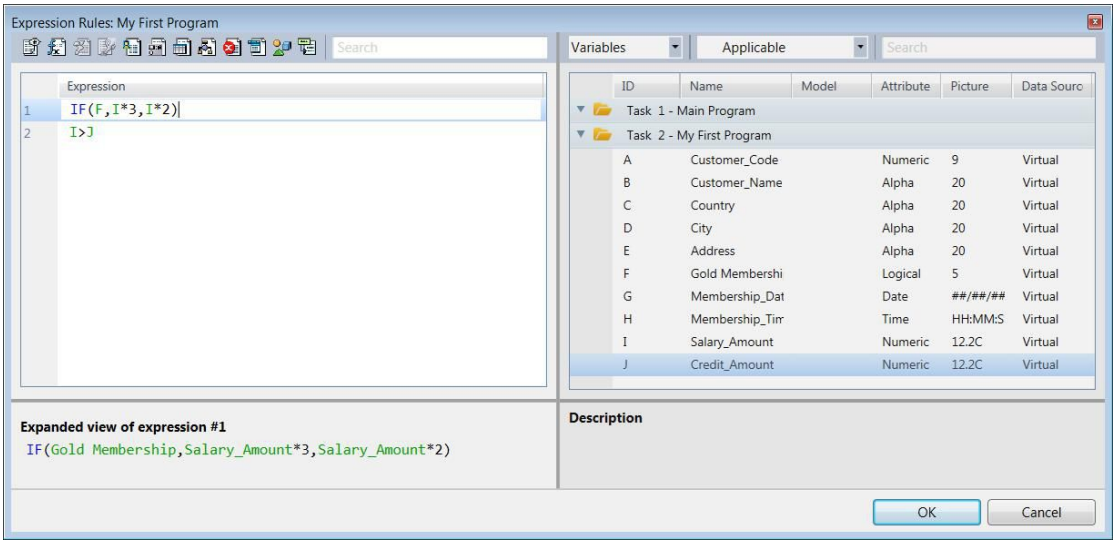
When you need to condition an operation, you can use the **IF** function. This function evaluates a Boolean expression and returns one value if **True** and another value if **False**. In the following example you will use the **IF** function to control the multiplication factor (from the last example), 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).

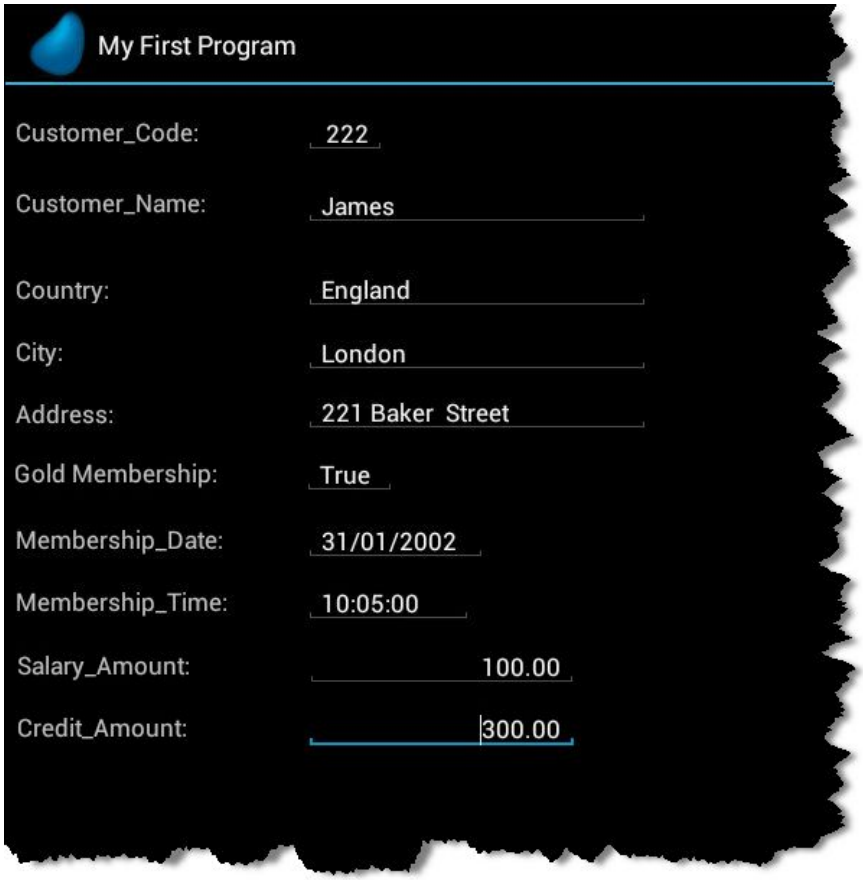


6. **Save** and **close** the program.
7. Check if no error by pressing F8 then execute the project by pressing F7 and then open the Magic xpa client on your device

8 .In the **Gold_Membership** control, type: **True**.

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**.



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**.

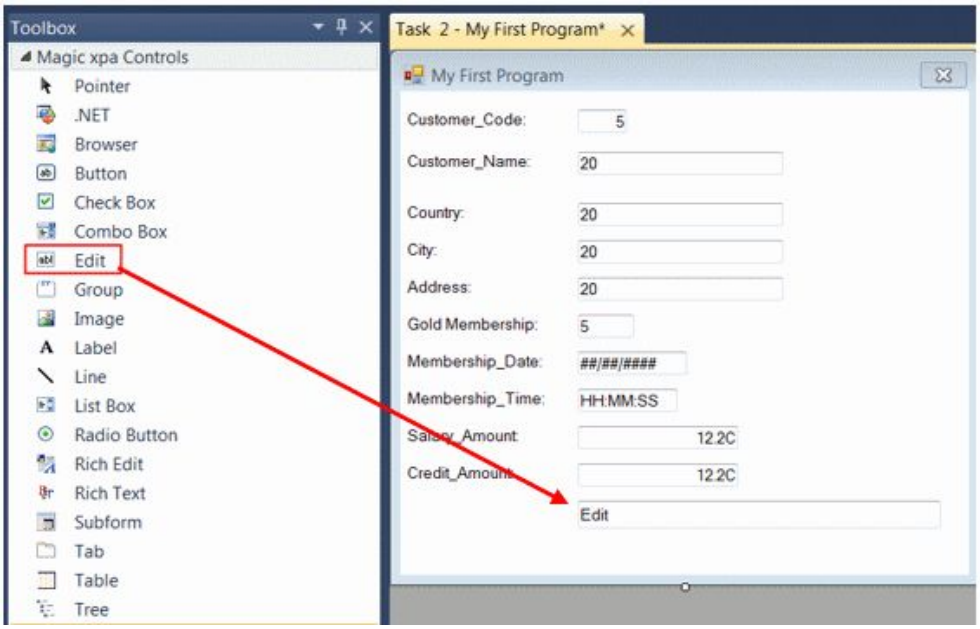
Alphanumeric Data Manipulation

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

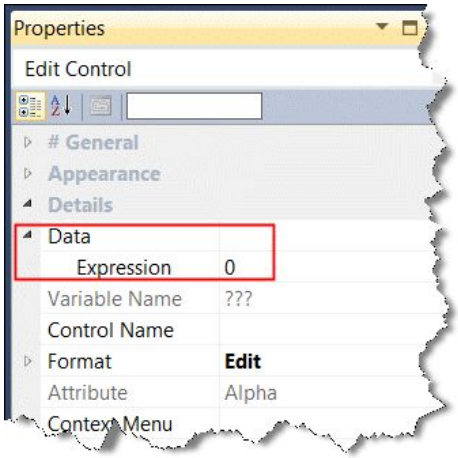
1. Zoom to **My First Program**.
2. Switch to the Form Editor and zoom to the **My First Program** form.

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.



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



6. The Expression Editor opens.

7. Create an expression line.
8. Type the following expression: **Trim(E)&', '&Trim(D)&', '&Trim(C)**

   Where:

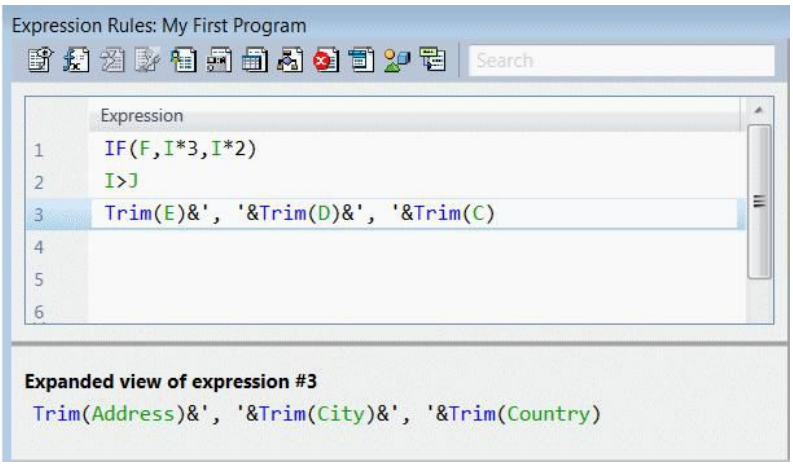
   E is the **Address** variable.

   D is the **City** variable.

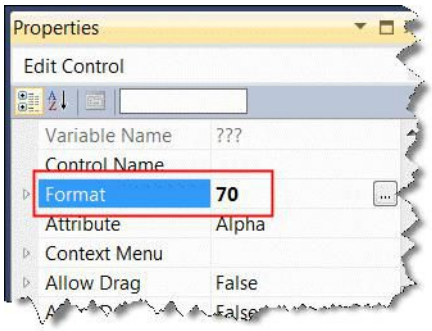
   C is the **Country** variable.

   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.

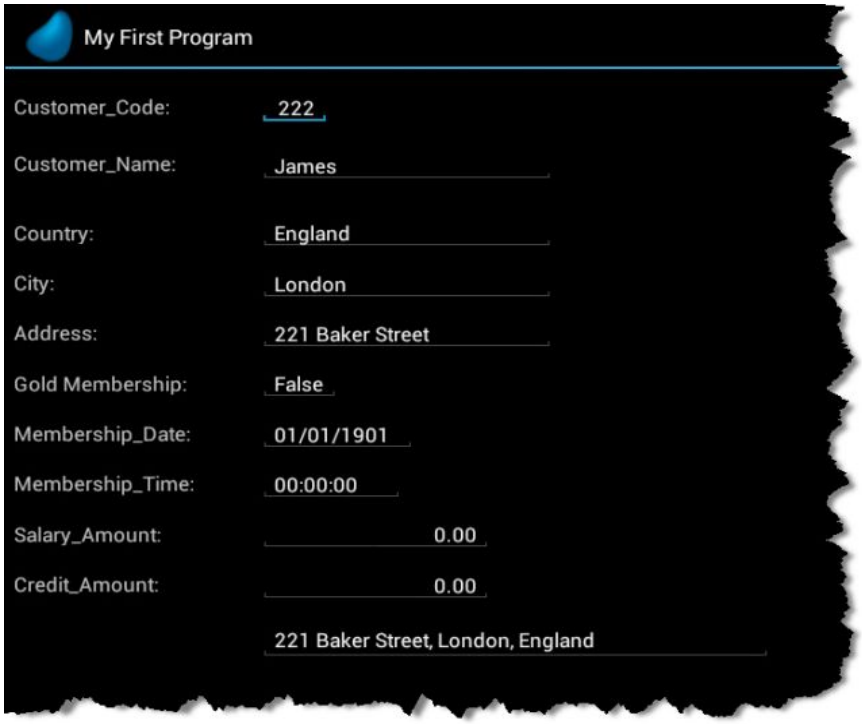


9. Click **OK** to select the expression for the **Data** property.
10. In the Edit Control properties' **Format** property, type **70**.



11. Close the program and save the changes.
12. Check if no error by pressing **F8** then execute the project by pressing **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.



Magic xpa Internal Data Validation

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. **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.
Logical	5	"5" : "False" or "True" on the other hand "1" : "F" or "T"
Time	HH:MM:SS	This is the default picture for defining a time variable.

Numeric Attribute Validation

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**.

Alpha Attribute Validation

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

The variable picture also defines the length of the variable.

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.

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 **not** later than the current date.

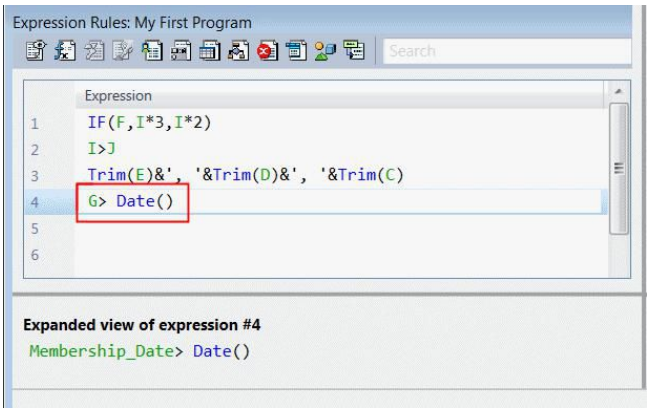
1. Zoom to **My First Program**.
2. Switch to the Logic Editor.
3. **Create Header Line** (**Ctrl+H**) to create a Header line for the **Control Suffix** logic unit. Define this Header line for the **Membership_Date** control.

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.

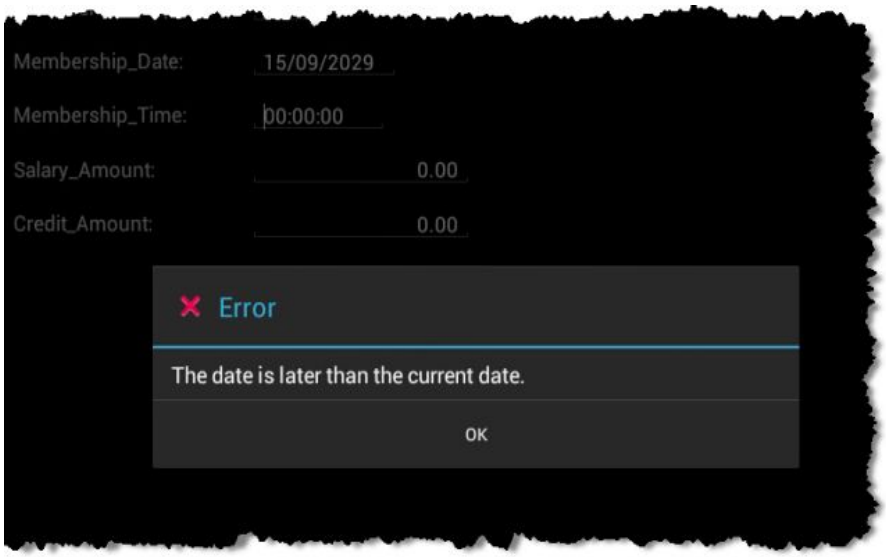
4. Park on the **Control Suffix** logic unit line and create a Detail line.
5. Define a **Verify Error** operation.
6. In the property sheet, go to the **Text** property and type: **The date is later than the current date**.
7. Park on the **Cnd** property and zoom to the Expression Editor by pressing **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**.



Now you will execute the program to view the results.

12. Check if no error by pressing **F8** then execute the project by pressing **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.



Data Consistency

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.

Parameters

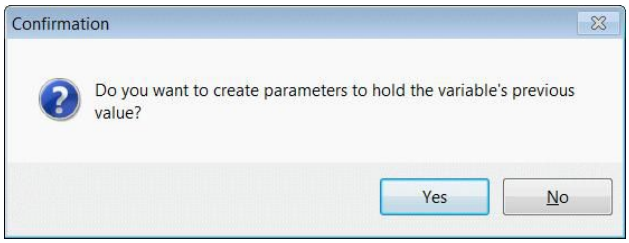
When you create a Variable Change logic unit, you are prompted to approve the creation of the following parameters. You will learn more about parameters later in the course.

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.

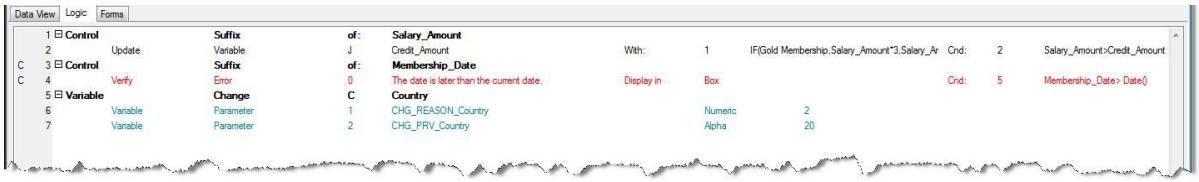
1. Zoom to **My First Program** and switch to the Logic Editor.
2. Create a Header line (**Ctrl+H**) and define a **Variable Change** logic unit for the **Country** variable.

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



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

Your program should now look similar to the image below.



4. In the **Variable Change** logic unit, create a Detail line (**F4**).
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 <>' '**

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

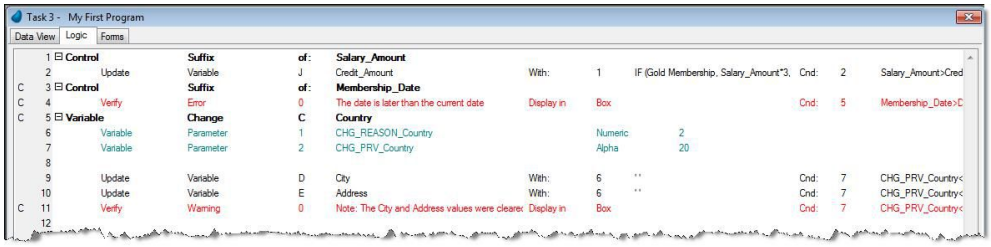
Now you will add a **Verify** operation to alert the end user that the values of the city and address were cleared.

9. In the **Variable Change** logic unit line, create a line and add a **Verify Warning** operation.

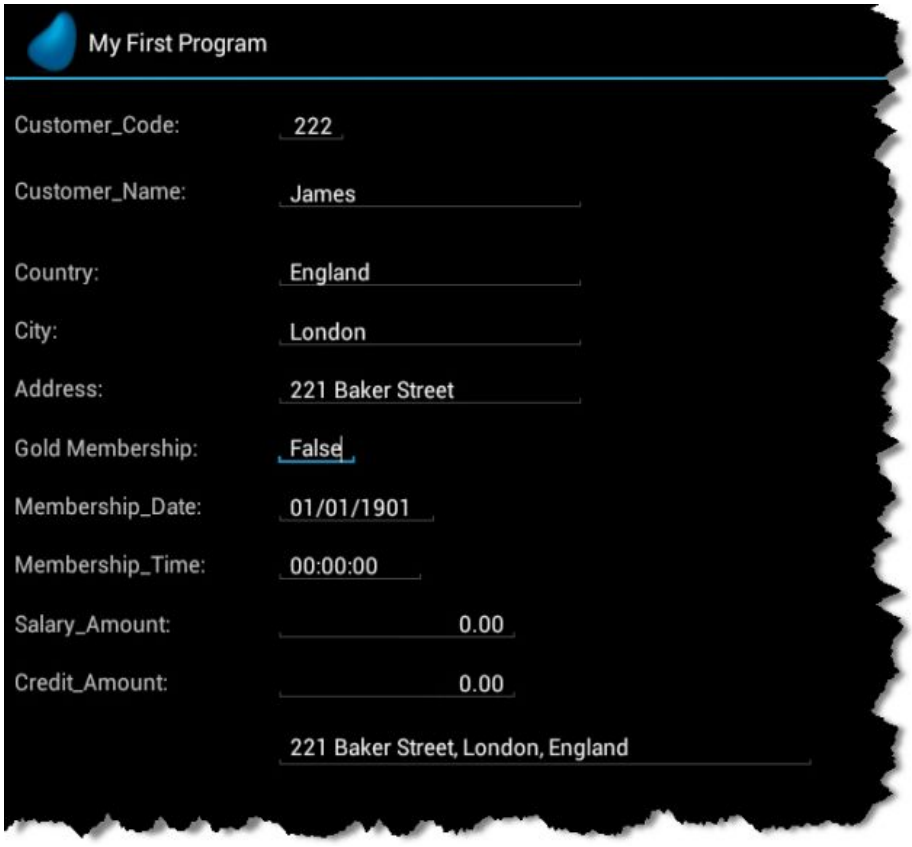
   The warning will be: **Note: The City and Address values were cleared!**

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:



Now you will run the program to view the results.



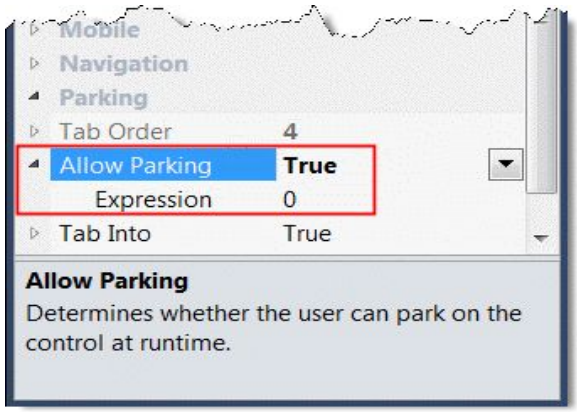
12. Execute the project.
13. Type the same customer details as you did earlier in this lesson.
14. Change the **Country** value to: **Spain.**
15. Tap the **Customer_Name** control.

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.

Parking Condition

Now you will learn another way to maintain data consistency.

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**.
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**.

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 <>"**
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.

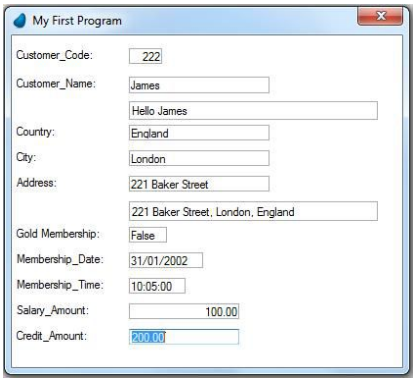
Exercise

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**.

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.



Now you will practice what you learned about validation.

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.

Maintain the data consistency of the **Membership_Date** and **Membership_Time** variables, so that if the date is changed, the time is cleared.

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 **'00/0/0000'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 4

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 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

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.

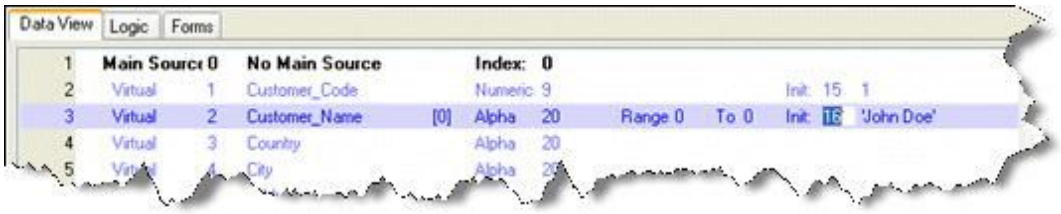
There are some differences between the **Update operation** and **variable initialization.**

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

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**
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 (' ').



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.

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.

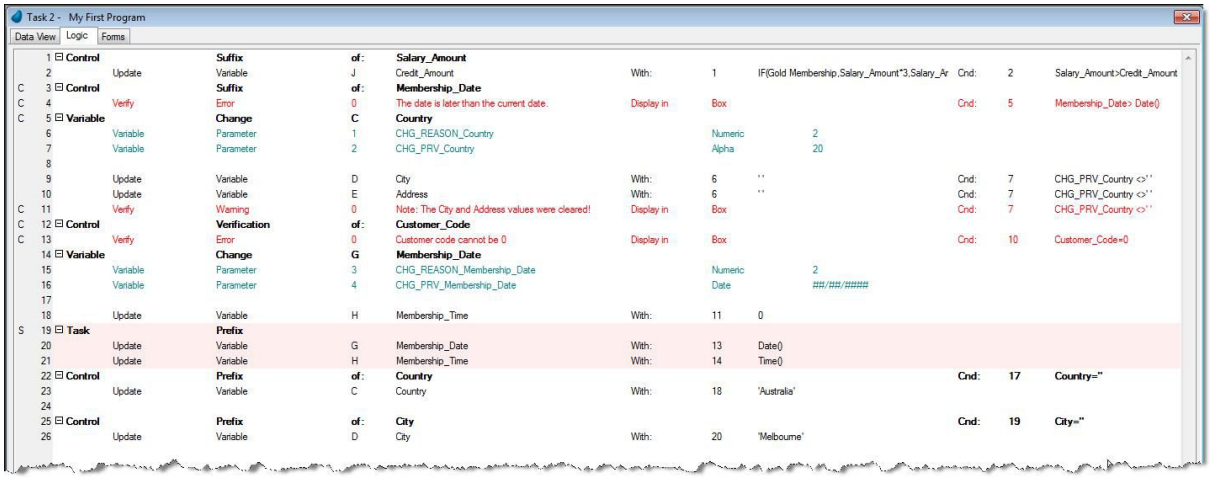
When you tap a control, the **Control Prefix** logic unit operations for that control are executed before the focus is moved to the control.

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.

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'**



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 5

Setting the Form's Appearance

This lesson covers various topics including:

- Color, font, and style
- Wallpaper

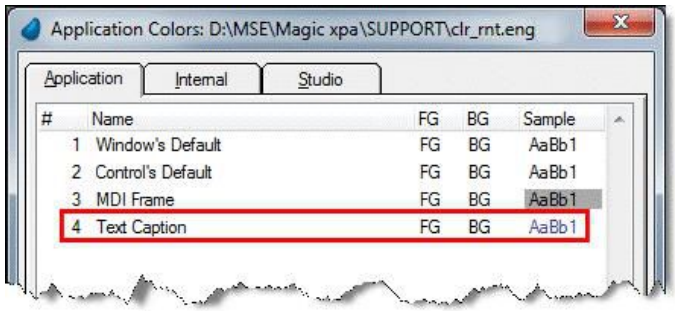
Adding Colors to the Color Repository

Adding a Text Caption Color

Now, we will learn how to add color settings to the Color Repository.
But in REAL WORK, the Color Repository is already defined.
So you should NOT change the Color repository.

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

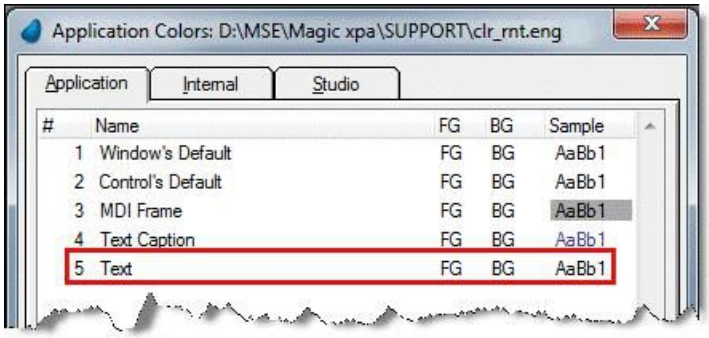


7. Click **OK**.
8. Zoom from the **BG** column.
9. Select the first (empty) entry from the **System** drop-down list.
10. **Check** the **Transparent** check box.
11. Click **OK**.

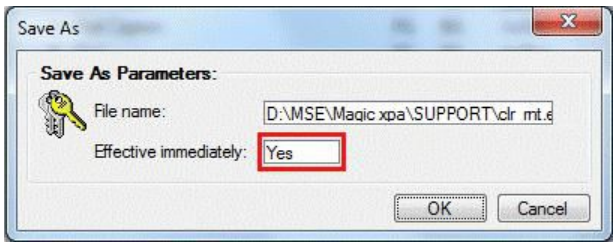
Adding a Text Color

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**.



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**.

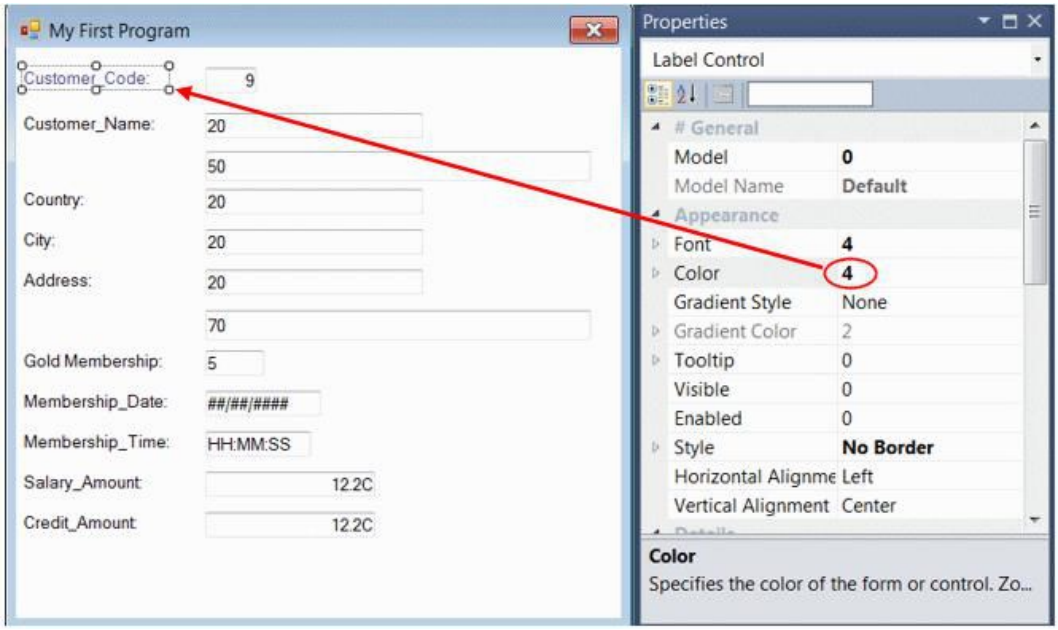


Changing a Control's Color

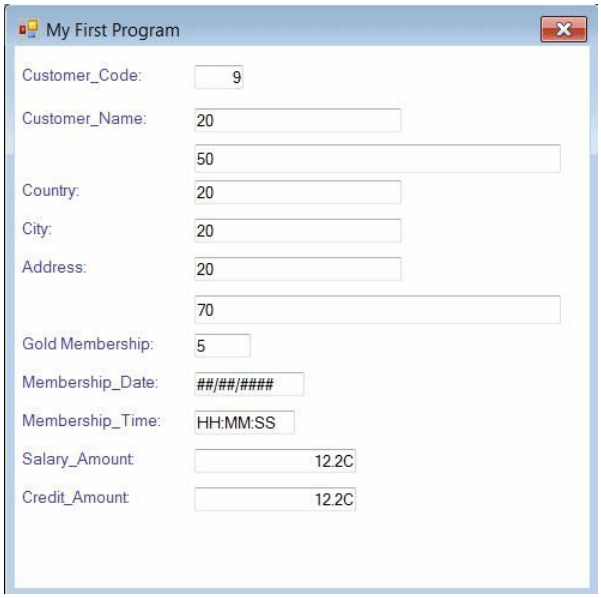
In this section, you will learn how to change the color of a control.

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.



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.



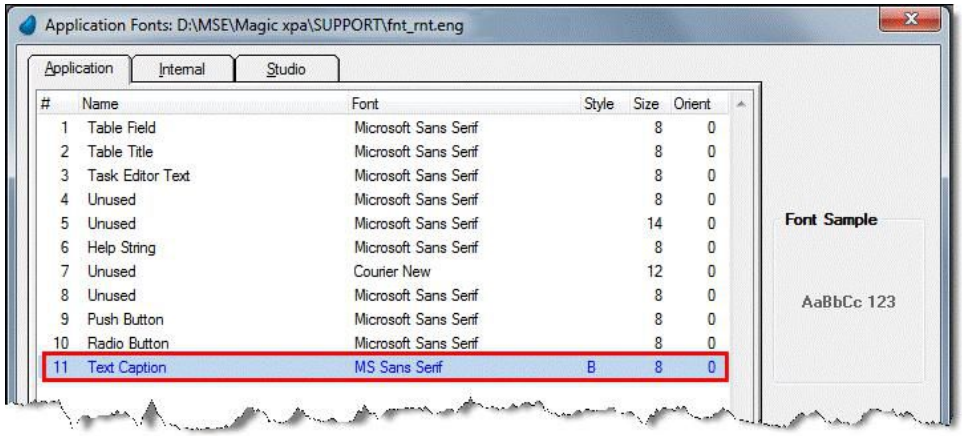
Adding Fonts to the Font Repository

Adding a Text Caption Font

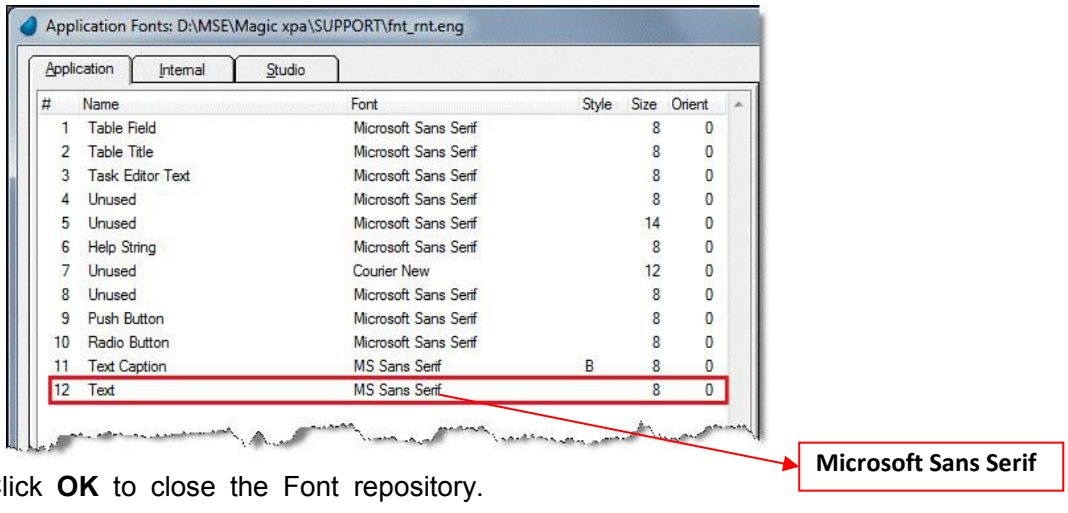
Like the Color Repository,in REAL WORK, the Font Repository is already defined. So you should NOT change the Font repository, neither.

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**.



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

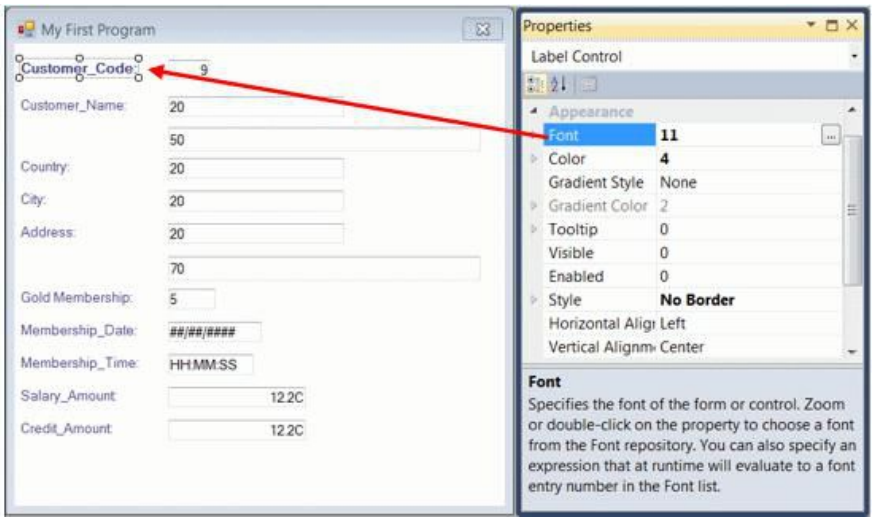


7. Click **OK** to close the Font repository.
   The **Save As** dialog box will appear.
8. From the **Effective immediately** parameter, select **Yes**.
9. Click **OK**.

Changing a Control's Font

In this section, you will learn how to change the control's 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.



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

Move a Control

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.

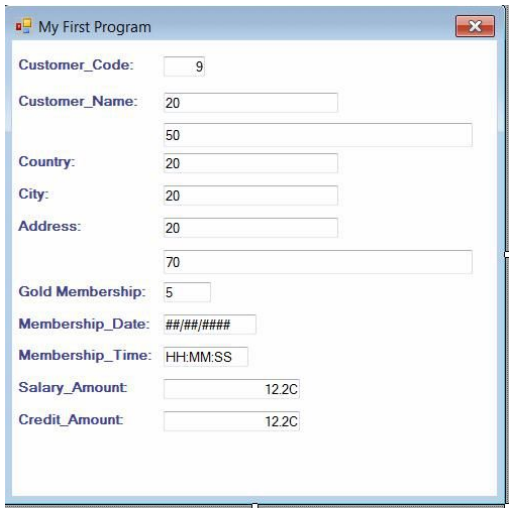
Using the Keyboard

- 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

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.



Wallpaper

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

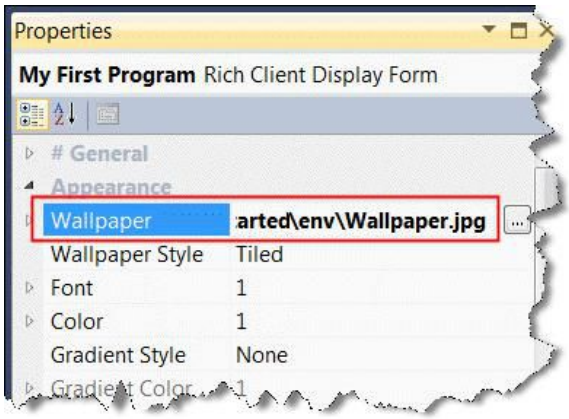
Please copy the "env" the following folders from

¥¥10.169.140.15¥SDShare¥Tools¥MagicXPA_Traning¥MagicXPA_TraningSetUp¥2_Copy Folder¥XPA¥Temp¥env

into your project (**Getting Started**) directory.

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.



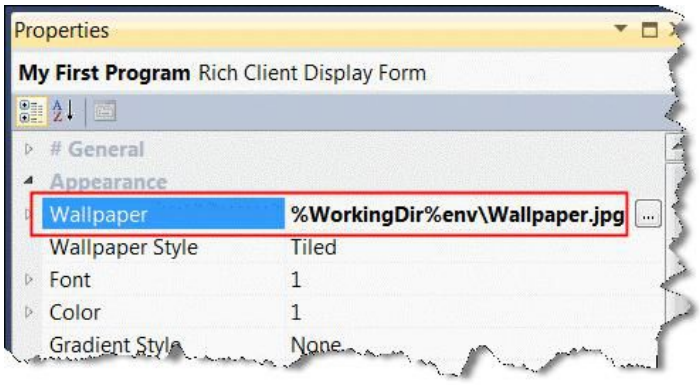
Dynamic Variable

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**.



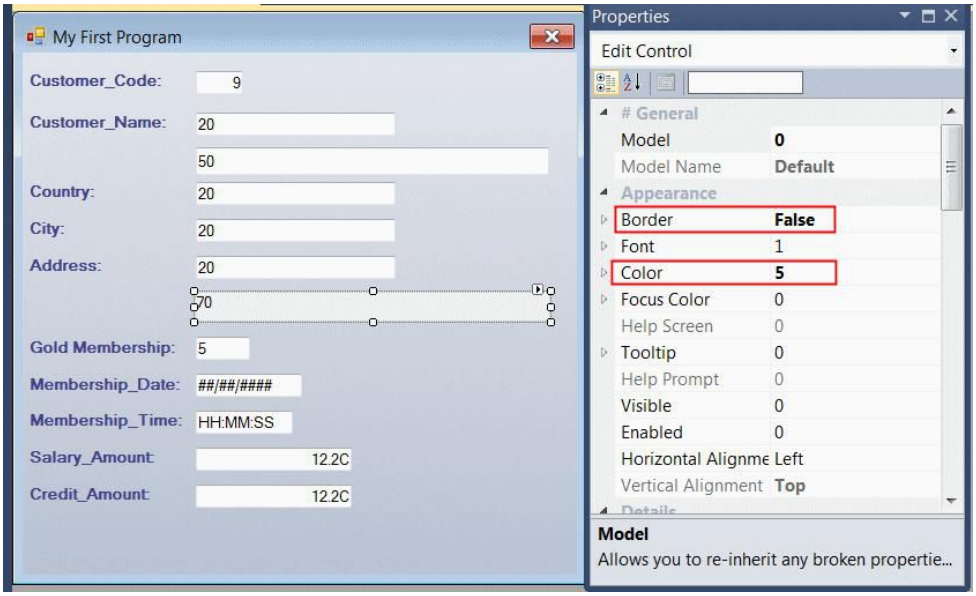
You can open the Form Designer and see that the wallpaper image is still there.

Transparent Color for a Control

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.

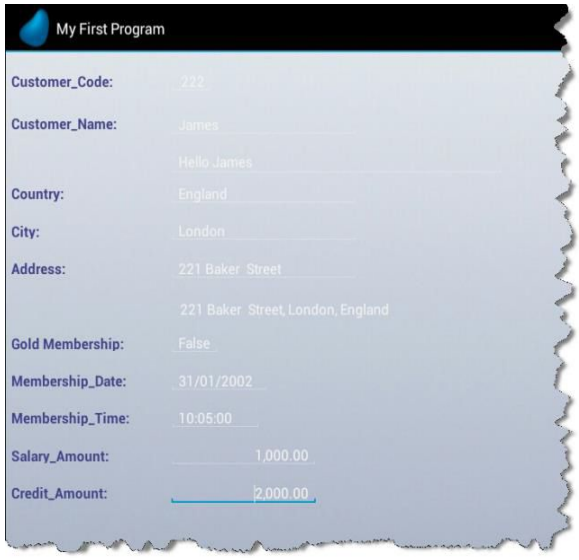


4. Close the program and save the changes.

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 at the right.



Placement

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**.
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. Reset the **Border** property to **True**.

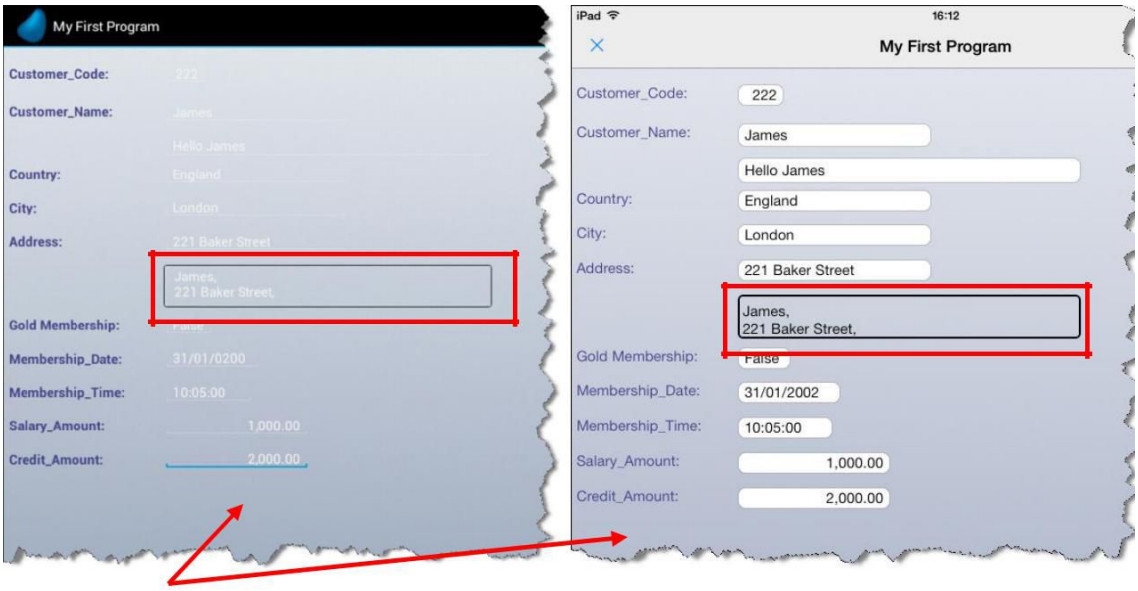
   d. In the **Input** section, set the **Multiline Edit** property to **True**.

   e. In the Appearance section, set the Vertical Alignment property to Top.

   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.

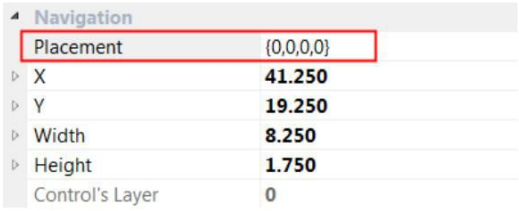
Now you can execute the program and see the result.

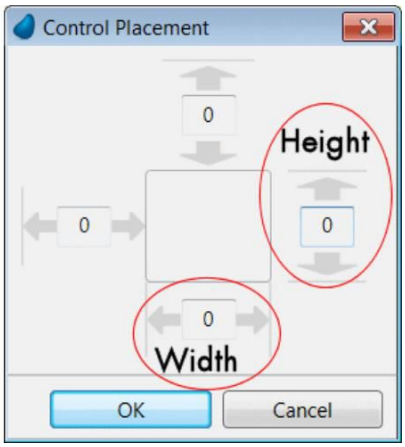


As you can see in the image above, the **Address Label** shows only part of the data.

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.

The placement of a control is defined by four values.





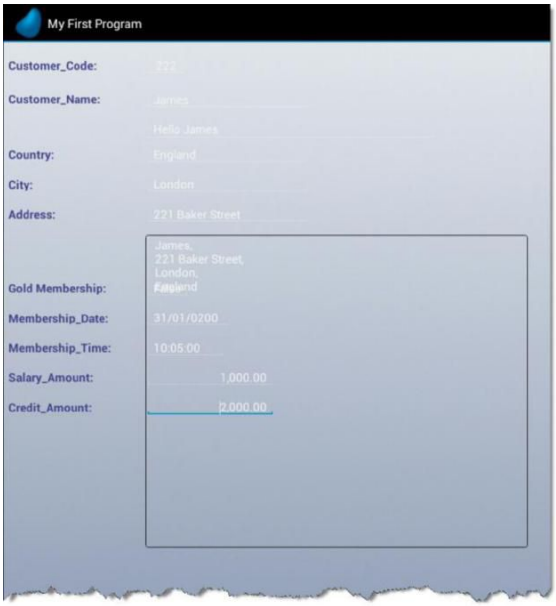
In the **Placement** property, the **Width** and **Height** values determine how the control resizes itself when the form is resized. The value is in percentages. When you set the value as zero, you are defining that the control is not resized. When you set a value of 100, you are saying that the control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

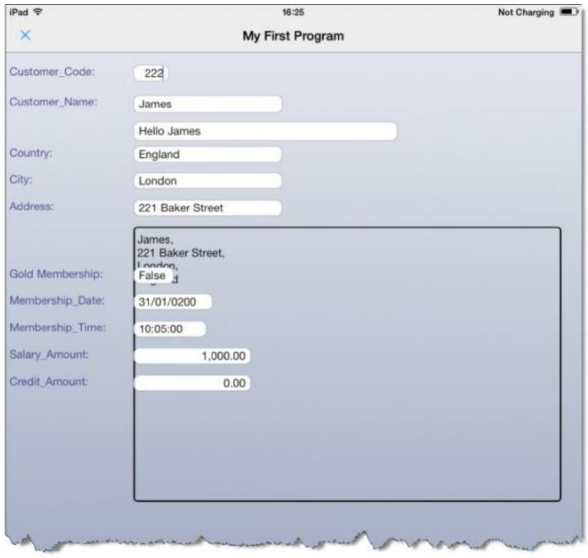
1. Zoom into **My First Program** and zoom into the Form Designer.
2. Park on the **Address** Label control and open the control property sheet.
3. Click on the **Placement** property and then click the Zoom button.
4. Set the **Width** property to **100** in and the **Height** property to **100**.
5. 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.

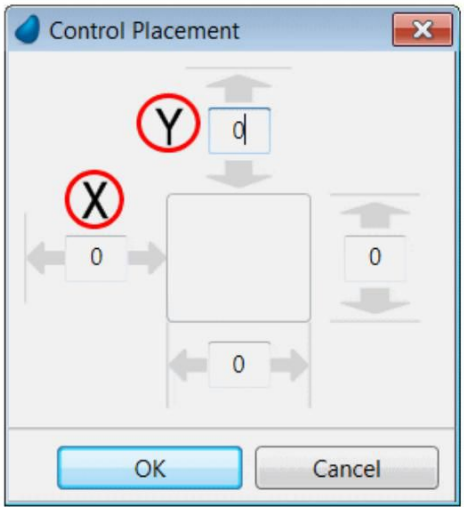
Android



iOS



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**.

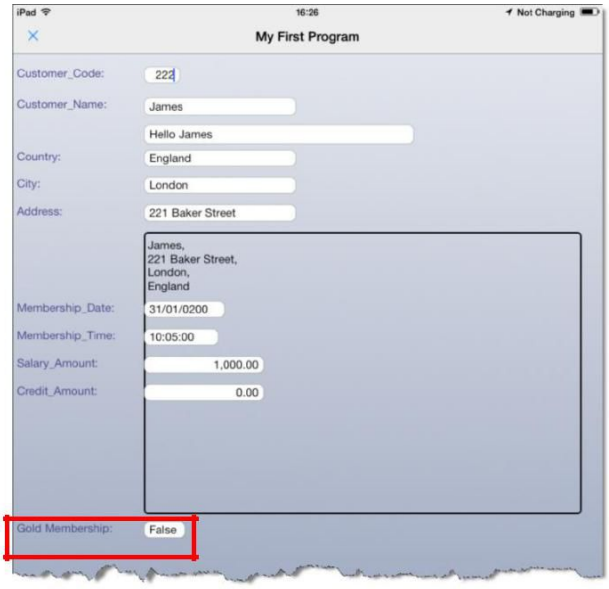
Remember that if you change the Edit control, its label must move with it:

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

iOS



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.

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%.

Magic University

Lesson 6

Viewing Data Source Content

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.

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.

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.

Defining the GettingStarted Database

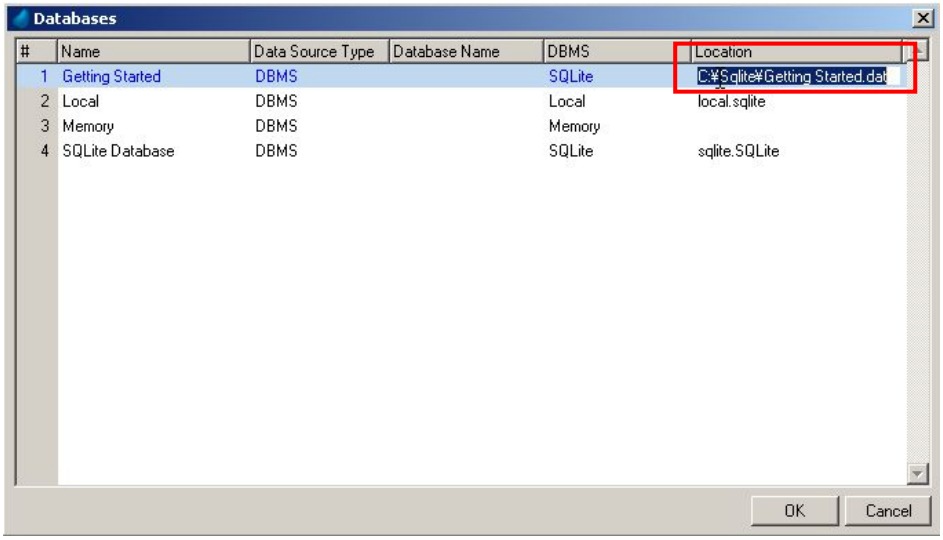
To access those tables, you need to define a connection to the **SQLite** entry in the Database repository.

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.Set the **Location** to:
   C:\Sqlite\Getting Started.dat



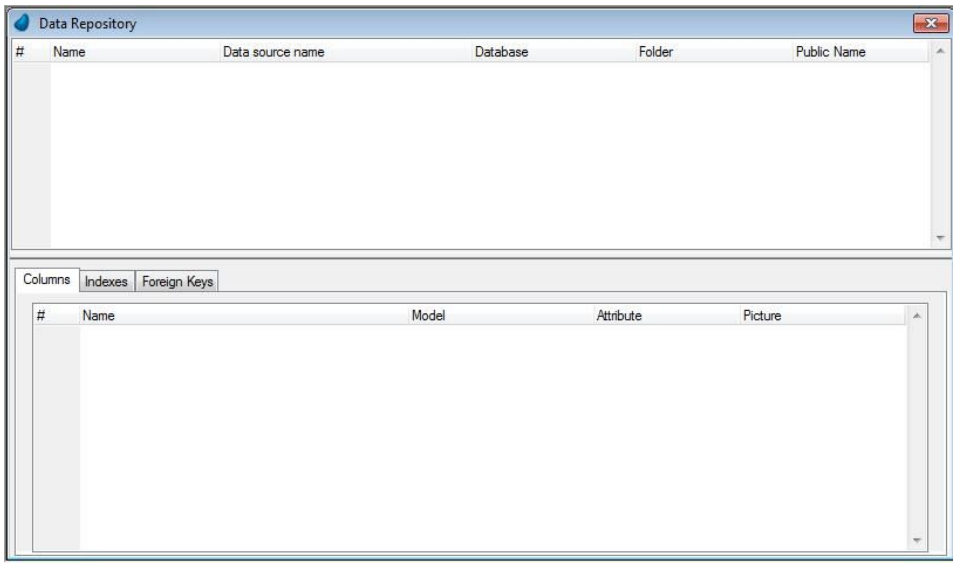
Defining a Data Source

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:

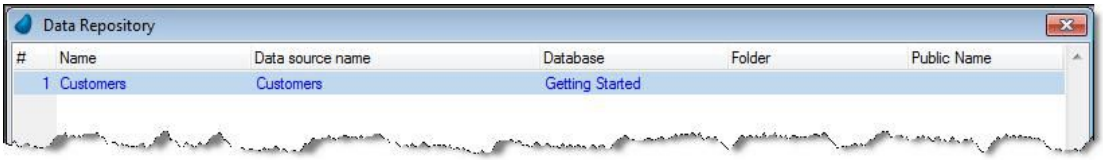
- The **upper pane** contains basic properties of the data source, including its name, data source name, database, folder, and public name.
- The **lower pane** displays three repositories: columns, indexes, and foreign keys. The displayed content in the lower pane changes according to the selected data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



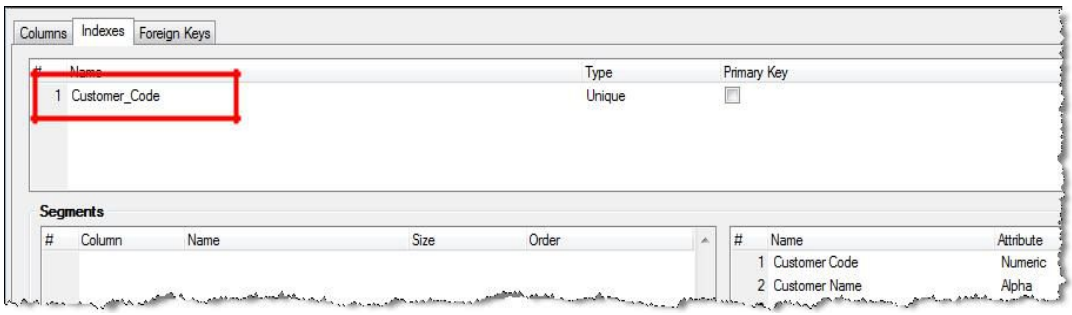
Defining Columns

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:

Defining an Index

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

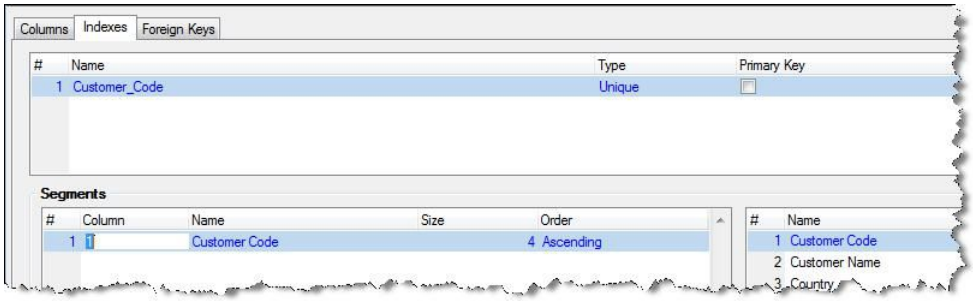
9. Click the **Indexes** tab in the lower pane.
10. Create a line and define an index as shown in the image below:



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.



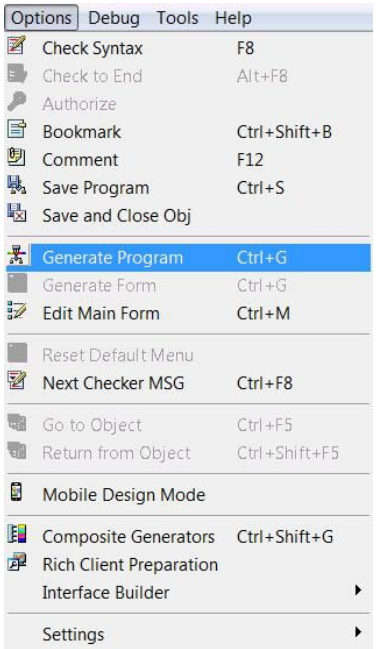
15. Click the upper pane to return to the Data Source definition.
16. In the Confirmation dialog box, select **Yes** to save your work.

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

Automatic Program Generation

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.

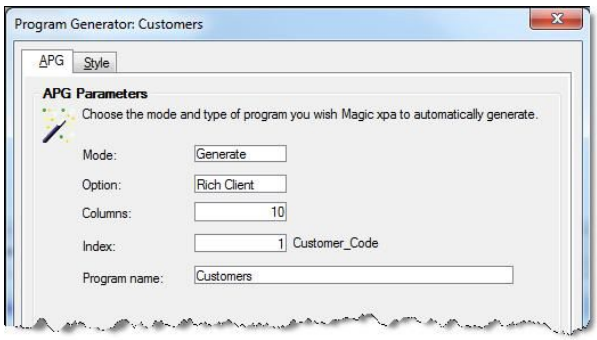
1. In the Data repository, park on the **Customers** data source.
2. From the **Options** menu, select **Generate Program** (**Ctrl+G**).



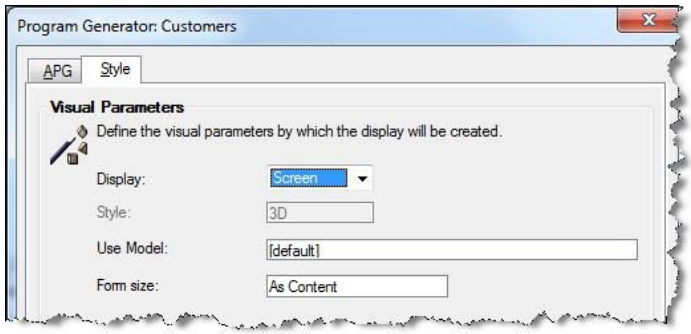
3. In the **Program Generator** dialog box, from the **Mode** property, select **Generate**.

   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.

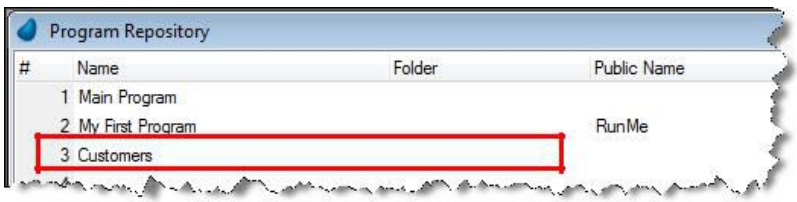


4. From the **Option** property, select **Browse**. This property sets the program's functionality.
5. Set the **Program name** property to **Customers**.
6. Click the **Style** tab.
7. From the **Display** property, select **Screen**.
8. Click **OK**.



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.

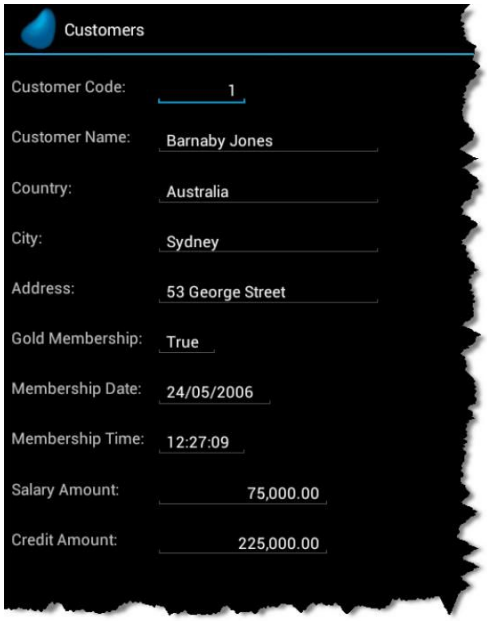


Running the Customers Program

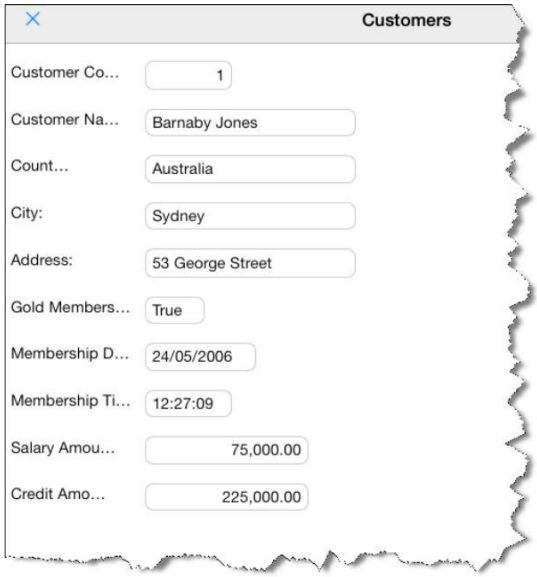
You now have a program that can display the data from the **Customers** data source.

11. Execute the project.

Android



iOS



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.

Manually Creating the Customers Program

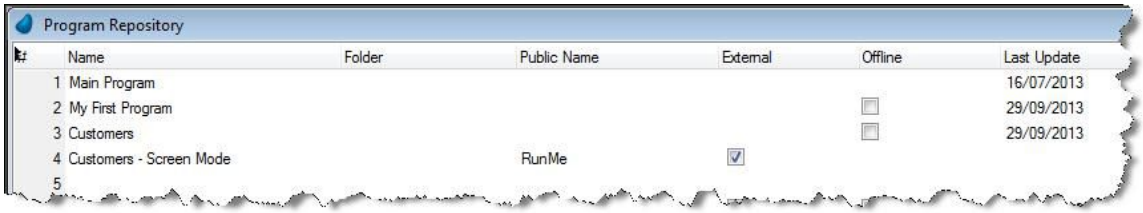
In the previous section you used the Program Generator to create a **Screen** mode program. In this section you will manually create the same program.

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.

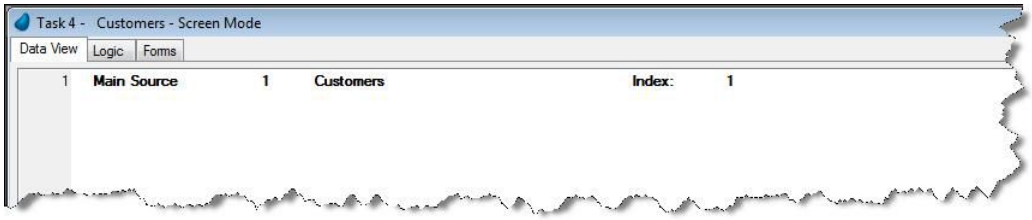
1. Open the Program repository (**Shift+F3**).
2. Create a line and name the program: **Customers - Screen Mode**



Defining a Main Source

3. Zoom into the **Customers - Screen Mode** program.
4. Define the **Task Type** as **Online.**
5. Click **OK**.
6. In the Data View Editor, park on the **Main Source** definition.
7. 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.

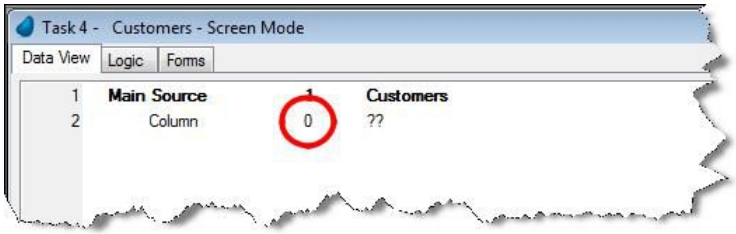
   Note: The first index is automatically selected in the **Index** column.



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

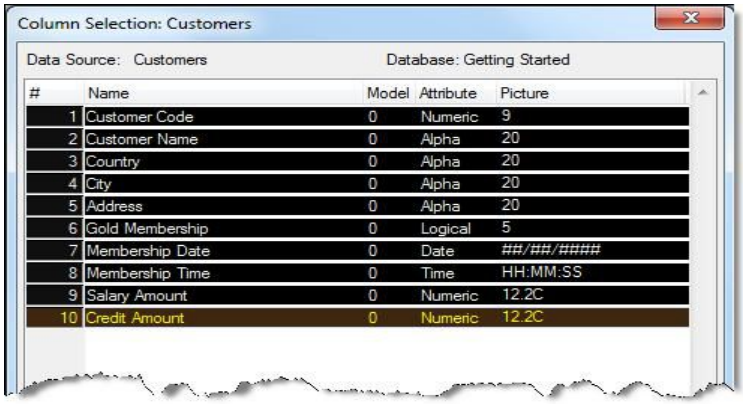
8. Create a line and from the column number (circled in red below), zoom to the

   Column Selection window.



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

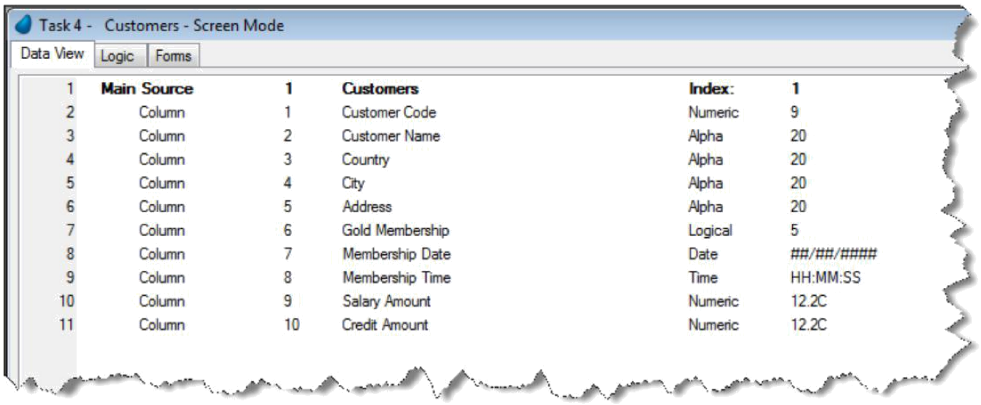
   - Park on the first line, hold down the **Shift** key, and click on the **#** column of the last line.
   - Park on the first line, hold down the **Shift** key, and press the **Down Arrow** key until all 10 rows are marked.



10. 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

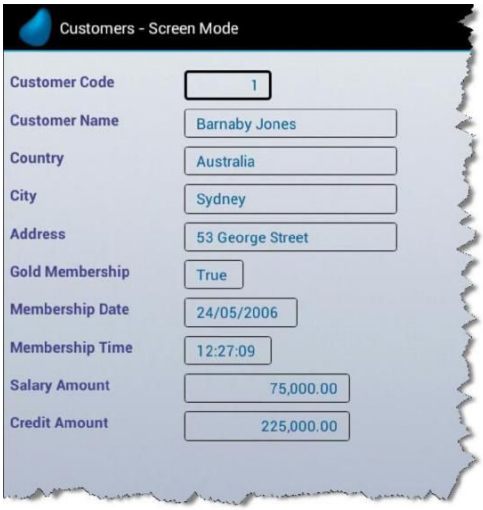
Designing the Form

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**.

12. Close the program and save the changes.
13. Execute the project.

Android



iOS

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.

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

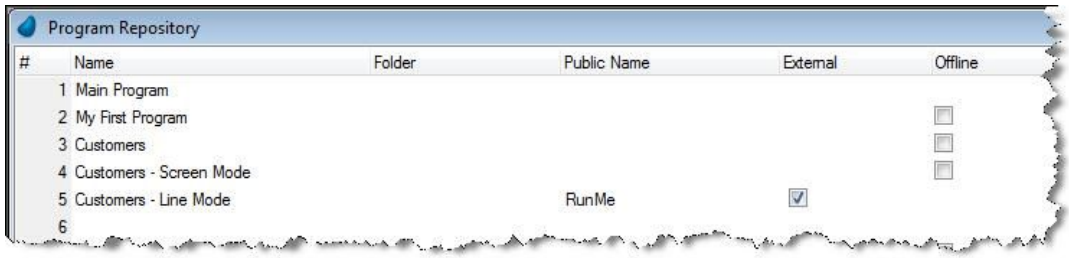
Magic xpa allows four modes of operation:

Mode of Operation	Allowed Operations
Create (INSERT IN SQL)	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 (UPDATE IN SQL)	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 (SELECT IN SQL)	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.

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**.



Defining the Task Mode

Set the Task Mode as follows:

2. Zoom into the **Customers - Line Mode** program.
3. In the **Task type** property, select **Online.**
4. Set the **Initial mode** property to **Query**. Users will only be allowed to scroll through the data without being able to edit it.

Defining a Main Source

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

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

Designing the Form

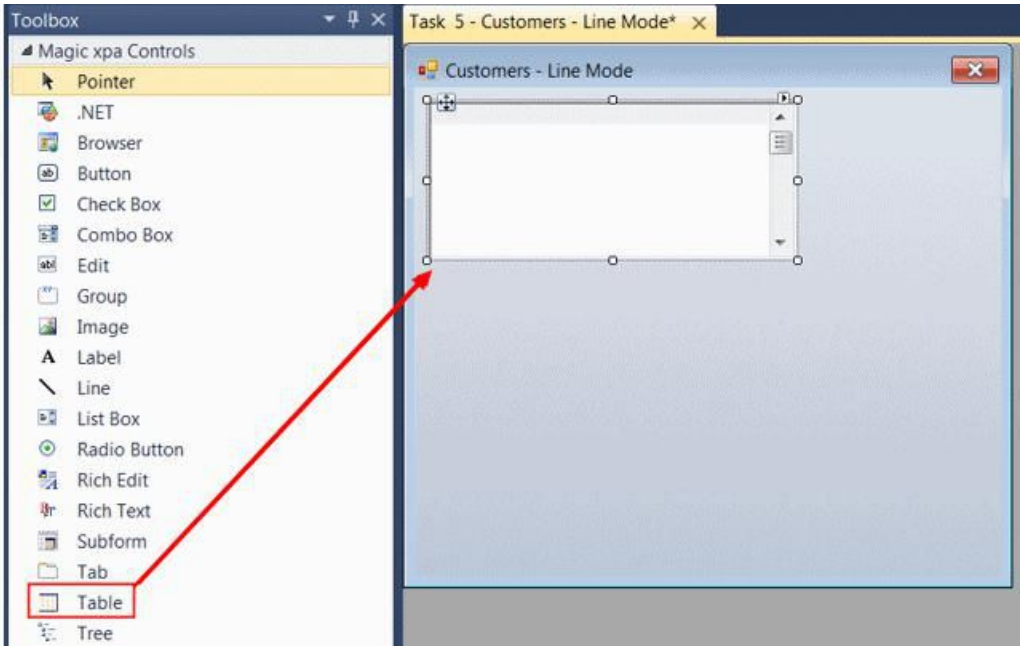
Set the wallpaper for the form as you did before:

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

Placing a Table control on the form

In the Form Designer:

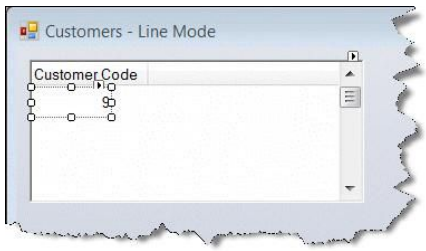
7. Zoom to the **Customers - Line Mode** form.
8. From the Toolbox, click on the Table control.
9. Drop the Table control on the form as shown in the image below.



Note: 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

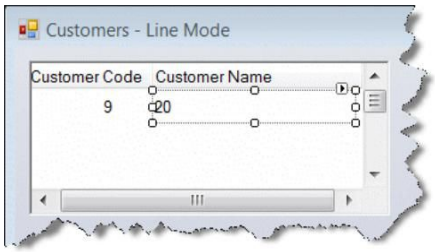
10. From the **Task Variables** pane, drag and drop the **Customer_Code** variable onto the Table control.
11. In the control properties, set the color to the **Edit control** color. This will provide a similar color to the screen mode form.



Now you will add two more variables to the table:

12. Place the following variables on the table:

    - Customer_Name
    - Gold_Membership

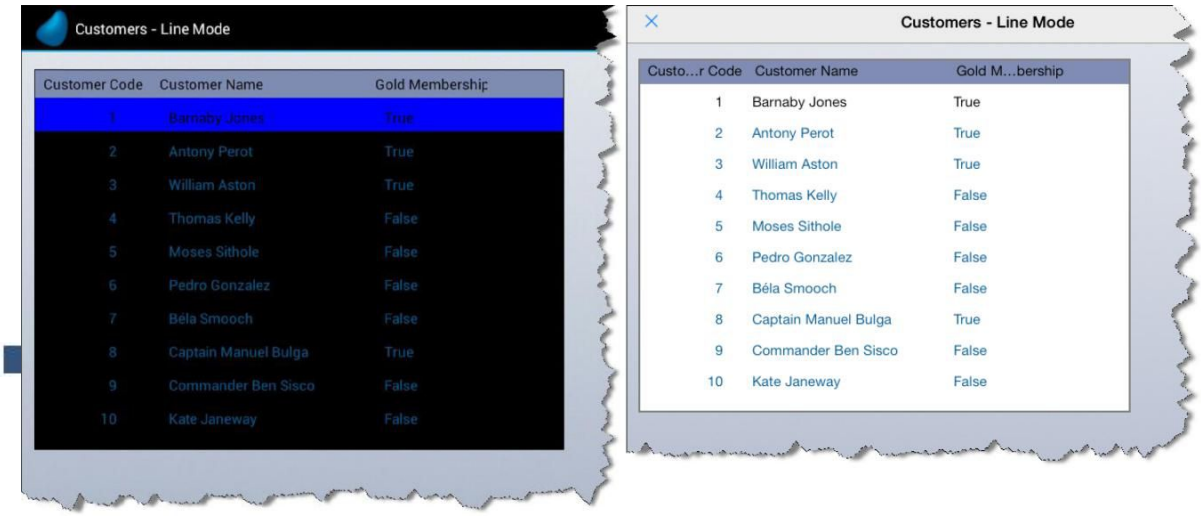


13. In the control properties of all three Edit controls, set the color to the **Edit control** color.

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.

14. Select the Table control and increase the table width and the table height.
15. Close the program and save the changes.
16. Execute the project and display it on your mobile device.



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:

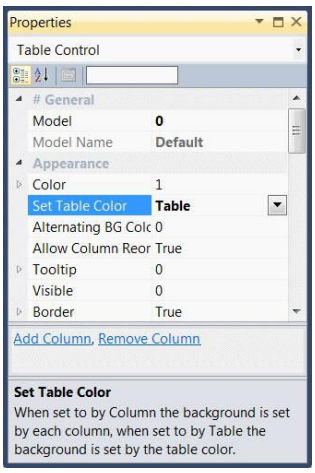
- Change the color of the table.
- Change the color of the row highlighting line.
- Change the text of the column header.

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**.



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.

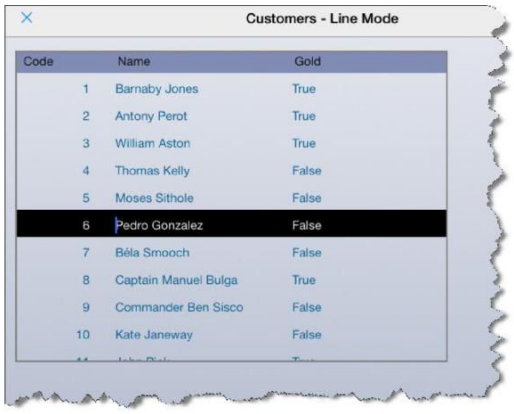
19. Park on the **Font** property and select the **Text Caption** font.

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

iOS



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 7

Models

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

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.

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.

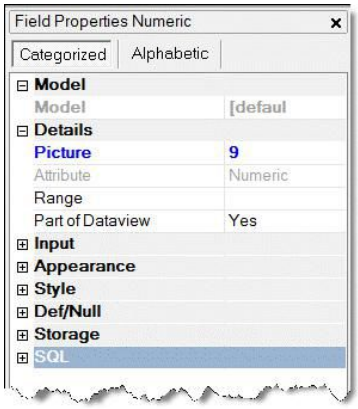
Some advantages to using the Model repository are:

- **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.

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**.



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

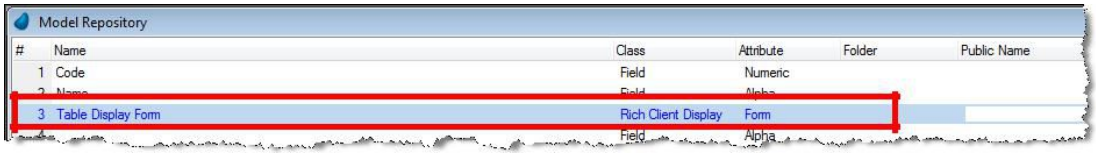
In a similar manner you can define a model for **Name**, which can be used for both the
Customer_Name and the **Supplier_Name**.

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**.



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

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 **Static**.
11. Define the following properties:

    - **Font** – Text caption
    - **Color** – Text caption
    - **Style** – No Border

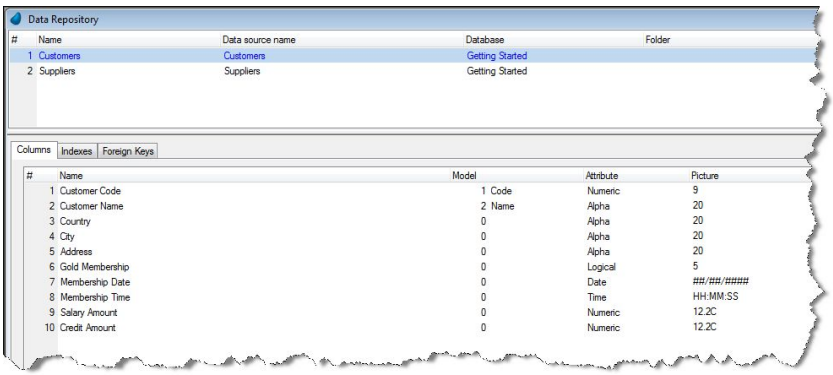
12. Create a model line.
13. In the **Name** column, type: **Edit Control**.
14. From the **Class** column, select **Rich Client Display**.
15. From the **Attribute** column, select **Edit**.
16. Set the **Color** property to the **Edit Control** color.

Assigning Models to Objects

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.
5. Select the **Code** model.
6. Repeat steps 3-4 for the **Customer_Name** column and select the **Name** model.



Inheriting the Model Properties

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**.
7.

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 **Online** 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 8

The Application Engine Concept

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 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

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.

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.

Task Prefix and Task Suffix

It is divided into two logic units:

Task Prefix (equivalent to Form _Open)

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(equivalent for Form_Unload)

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

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 logic unit is cyclical and may be executed several times during the task execution.

Control Prefix, Control Verification, and Control Suffix

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	-

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.

Magic® University

Lesson 9

Events and Handlers

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

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.

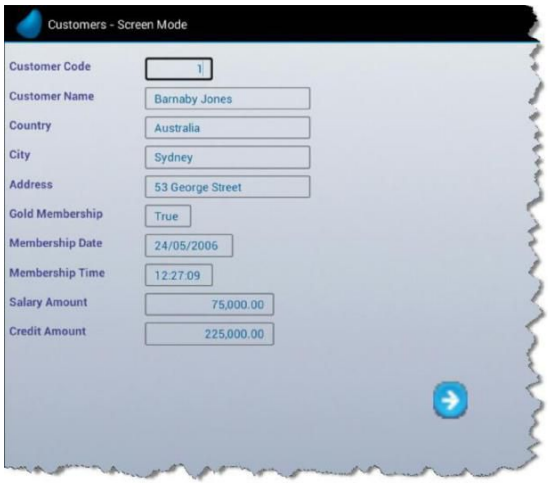
Raising Internal Events

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



iOS

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(Prepare the image file)** – 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**.

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 Date 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.

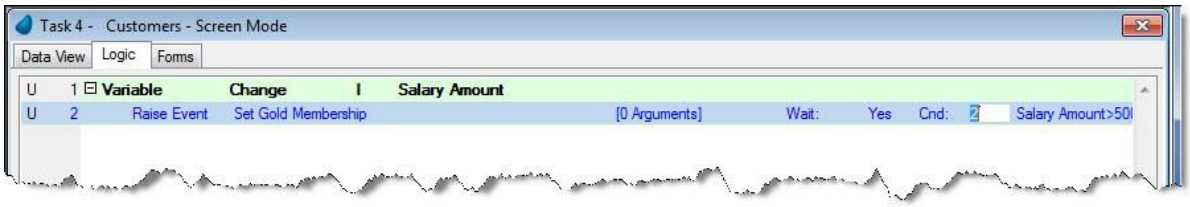
You will now create the logic unit that will raise the **Set Gold Membership** event when the **Salary_Amount** value is changed.

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**.

1

You have just used the Raise Event operation to invoke an event according to the task logic.

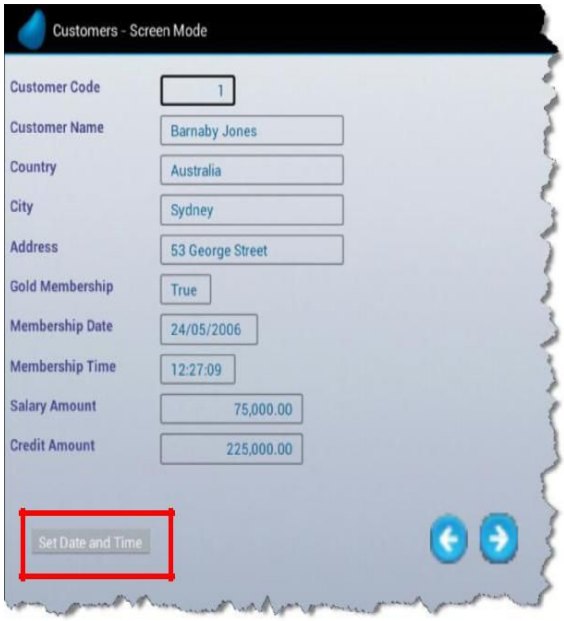


Invoking a User Event Using a Button Control

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

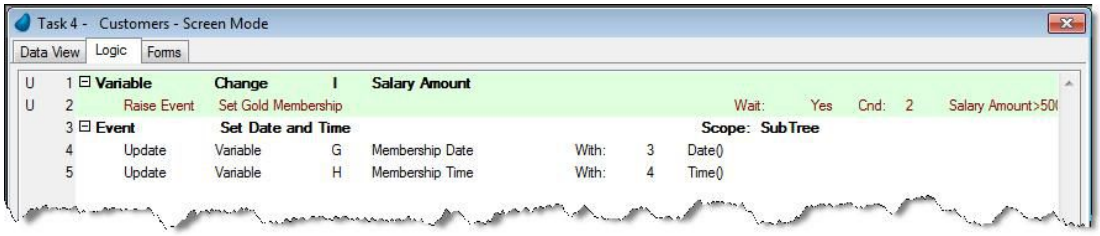


iOS

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**. Now, you will create the handler for the event.

1. In the **Logic Editor**, create a **Header line (Ctrl+H)** 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()**.



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.

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.

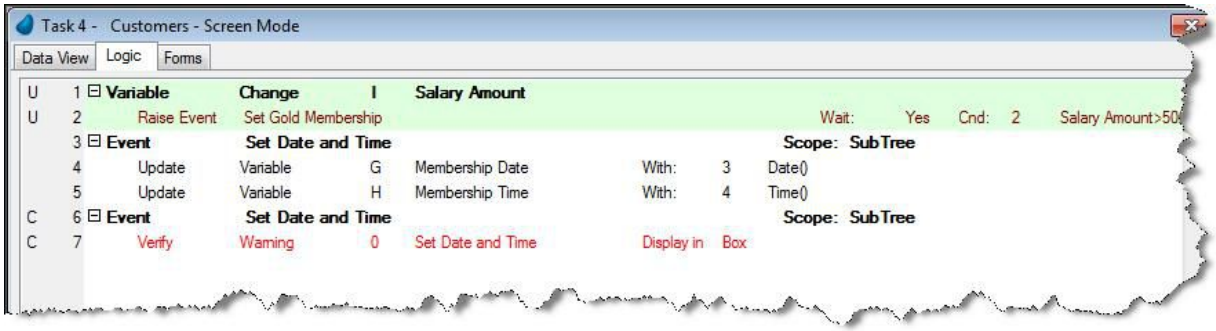
Setting this property to **Yes** is useful when you have a handler for the event in a parent task as well as in the current task. You will learn about this scenario later in this lesson.

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.



3. Execute the program on your computer 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.

Handling Internal Events

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.

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.

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 **PR: Customer code**. You do not need to add the **PR.** 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.

Range Criteria

Range criteria enable you to display only those records that fall between two values, from-to. You will now set the **PR: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 **PR: Customer Code** *parameter*.
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.

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.

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 (PR.Customer Code > 0, PR.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.

22. Set the expression to: **CndRange (PR:Customer Code > 0, PR: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%**.

   This description is not clear.
   Please make programs like below:

   CreditAmount = SalaryAmount * 2;

   if (SalaryAmount > 50000)
      CreditAmount = CreditAmount * 1.2;

   For example:
   SalaryAmount = 10000 then CreditAmount = 20000
   SalaryAmount = 100000 then CreditAmount = 240000

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.

Magic® University

Lesson 10

Conditioning a Block of Operations

.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

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.

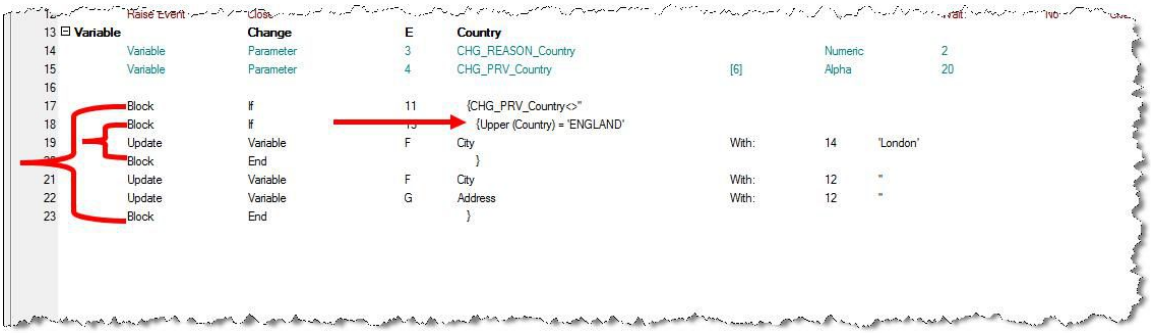
5. Park on the **Block If** line and create a line under it.
6. Update **City** with **''** – remember this is the way to reset an Alpha value.
7. Update **Address** with **''**.

The two Update operations will only execute if the condition of the Block If operation evaluates to True.

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.

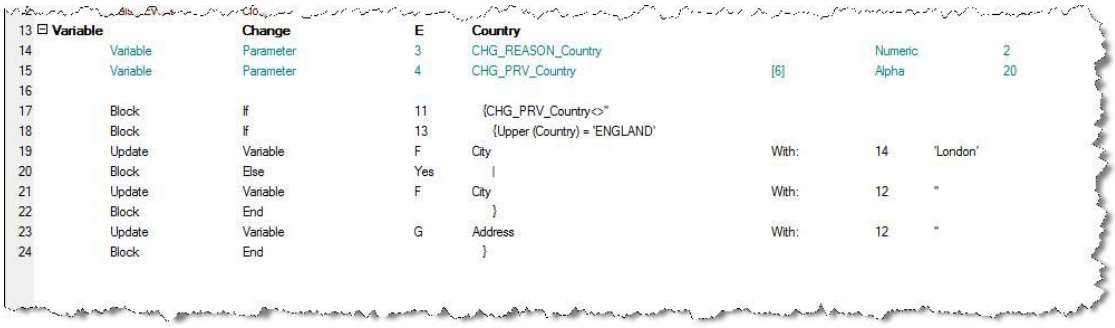


As you can see from the image above, there are **two Block If** operations, one directly after the other. The **second Block If** operation is a nested block. The **second Block** operation is displayed by the Logic Editor as nested.

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		}				

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 r**eturns the number of times the loop has iterated.** This means that you do not have to create a special counter for each loop.

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*)**
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*)**
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.

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.

Magic® University

Lesson 11

One-to-One Data Relationships

This lesson introduces you to one-to-one data relationships and describes how they are implemented in Magic xpa.

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.

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

```
SELECT *
FROM Orders
INNER JOIN Customers
ON Customers.CustomerCode = Orders.CustomerCode;
```

- One-to-many data relationships (You will learn about the one-to-many data relationships in a later lesson.)

```
SELECT *
FROM Orders
LEFT JOIN Items ON Orders.ItemCode = Items.ItemCode;
```

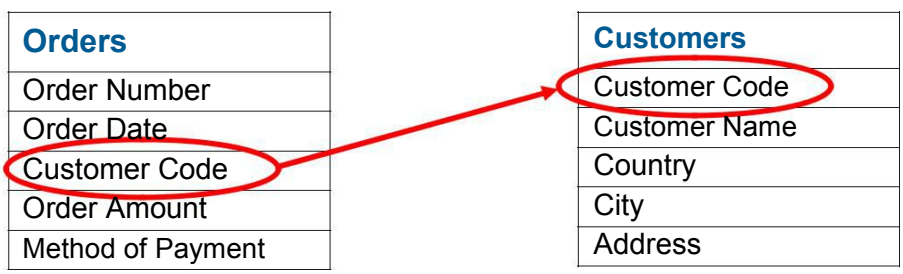
The following table compares the one-to-one and the one-to-many data relationships:

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.

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).



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.

Defining the Orders Data Source

In the Data repository, create a data source and name it **Orders**.

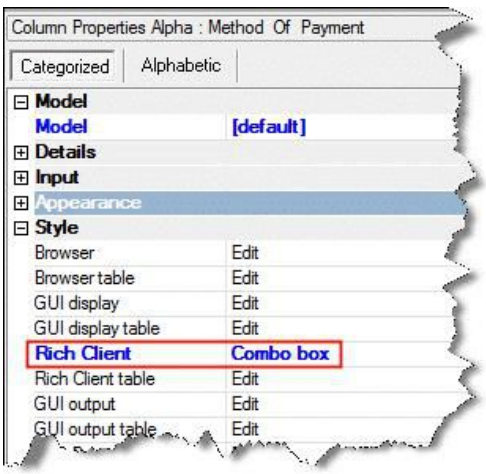
In the **Data source name** column, type: **Orders**.

In the **Database** column, select **Getting Started**.

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

1. Park on the **Method Of Payment** column in the **Data Repository**.
2. Open the Column Properties.
3. From the **GUI Display** style property, select **Combo box**.



In this example, you will display the following options in the **Method of Payment** column: **Cash**, **Check**, **Credit Card**, and **Coupons**.

1. From the **Online** 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.

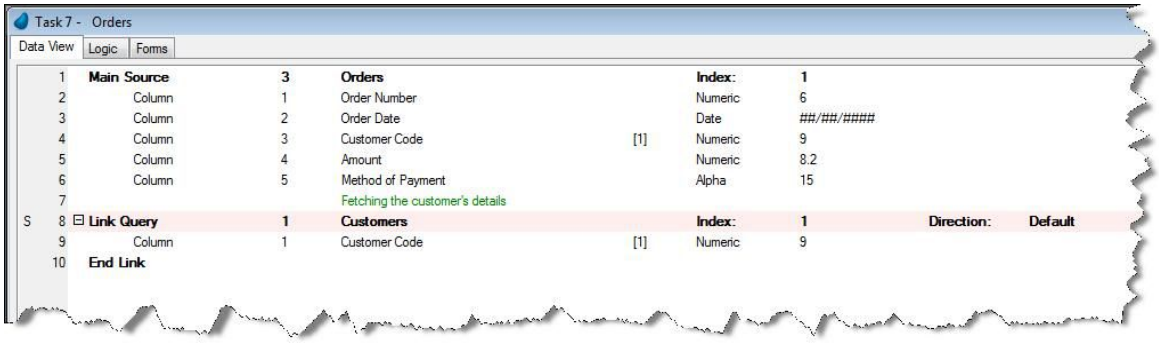
1. Create a program and name it **Orders**.
2. Zoom to the **Orders** program.
3. From the **Task Properties** dialog box, set the **Task type** to **Online.**
4. Set the **Initial Mode** to **Query**.
5. Open the Data View Editor of the **Orders** program.
6. Create a Main Source definition for the **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.

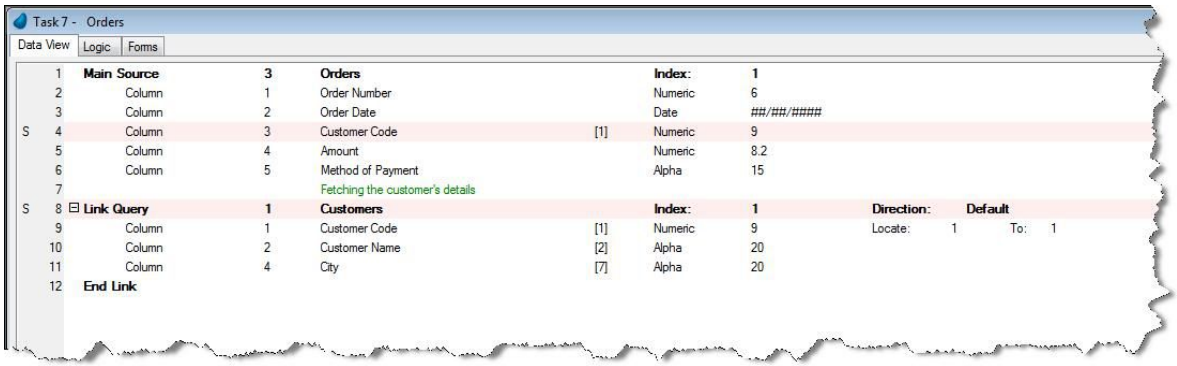


6. Park on the **Customer Code** column definition.
7. Create a line.
8. Add the **Customer Name** and the **City** columns from the **Customers** data source.

   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

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.



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

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.

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.

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.

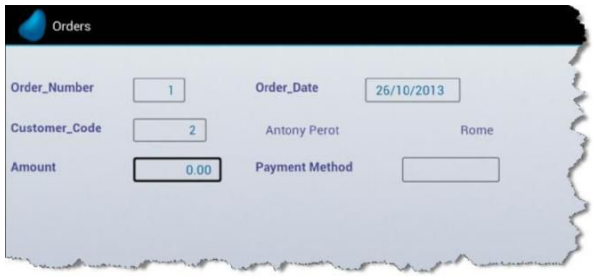
Execute the application.

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



iOS

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.

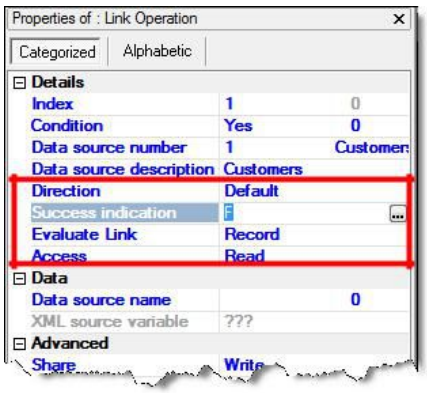
Each time the **Customer Code** was changed, the customer's details were also changed. Magic xpa re-evaluates the Link condition and attempts to locate the appropriate customer record in the **Customers** data source for the new **Customer Code** value.

This behavior is referred to in Magic xpa as **Link Recompute**.

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 in the Customer table..

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.



1. Open the Data View Editor of the **Orders** program.
2. Add a **RT: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(RT: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: 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** 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.

5. 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

6.Define two unique indexes:

a. ProductCode with the **Product_Code** index segment

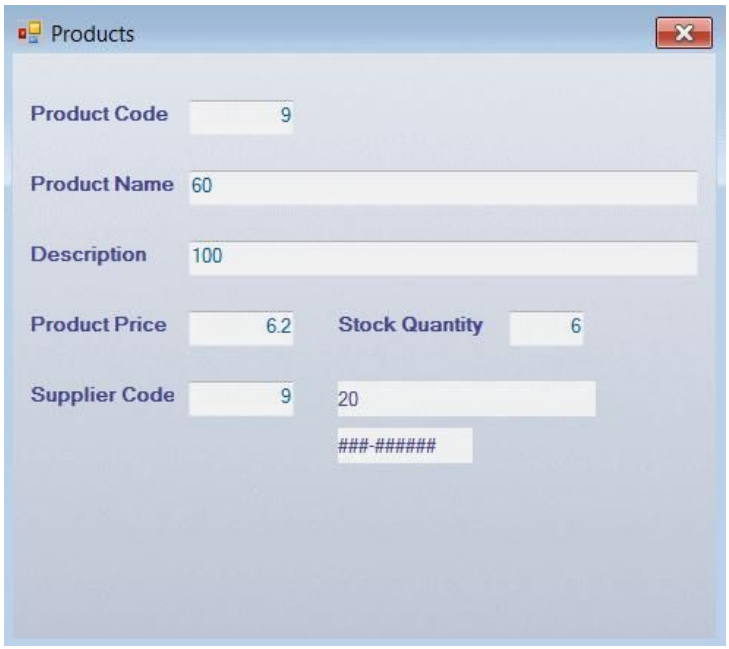
b.SupplierCode with the **Supplier_Code** and **Product_Code** index segments

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.

7. Park on the **Products** data source.
8. From the **Options** menu, select **Generate Program**.
9. In the **Option** parameter, select **Import**.
10. 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.
11. The program will be added to the Program repository. Execute the program to import the data.

12. Add a new program that browses the **Products** data source.

13. Set up the form as shown in the image below.



14. 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:

15. 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.

Magic® University

Lesson 12

Selecting Data from a List

This lesson covers various topics including:

- Selection tables
- Defining a Data control

Selection List

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.

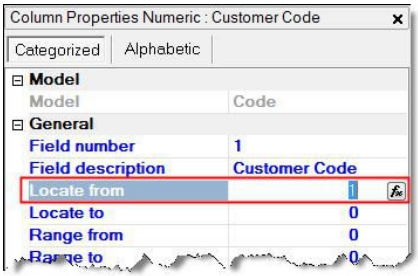
1. Create a program named **Select Customers**.
2. From the **Task type** property, select **Online**.
3. From the **Initial mode** property, select **Query**. A selection list is used to select values from a list. Other operations, such as modifying or creating new records, should not be used here. Therefore, it is better to set the task's **Initial mode** to **Query**.
4. From the **Selection table** property, select **Yes**.

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 **PR: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**.



Returning the Selected Value

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 **Display Only** model.
7. Select the **Customer_Name** variable and place it on the table. This is also already linked to the **Display Only** 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 **Button style** property and select **Image Button** from the combo box.
15. Park on the **Default Image** name property and type **%WorkingDir%\images\OK.png**.
16. Park on the **Format** style and remove the text, which is initially **Button**.
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 **Default Image** 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

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**.**Set trigger Type to:None.**

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 (0,'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. Execute the application.
2. Park on any order and tap the **Edit** button.
3. The **Orders** program will appear.
4. 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 **Data control** is a Combo Box control that displays one column from a data source.

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 **Products** 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.

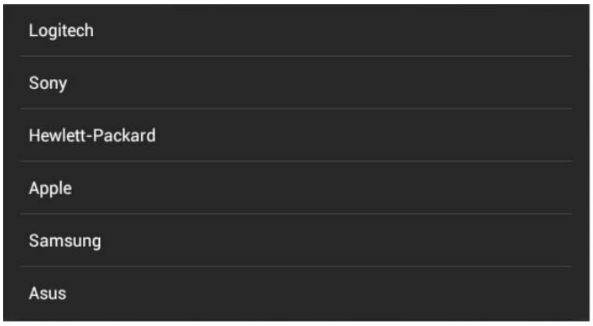
Remember that you set placement for other controls, so you need to set placement for this control as well. You only need to move the control if the height of the form increases.

17. Zoom into the **Placement** property and enter **100** in the **Y** property. The placement will look like this: **{0,0,100,0}**.

You can see that the Data control displays the supplier name value since you set the **Supplier_Name** as the displayed field.

18. Park on the **Supplier** combo box and select a different supplier.

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 (%WorkingDir%¥products¥xxx.jpg)** 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.

   Use the Image control.
   Set the "Data" expression.

   In addition, it is better to set
   the "Image Style" to "Scaled to Fit".

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.

   In the Task Properties,
   set "Initial mode" to "Query",
   "Selection table" to "Yes".

   On the "Product Code" column,
   set "Locate from" to the parameter.

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.

Magic® University

Lesson 13

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.

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

The **one-to-many** data relationship will be explained using the **Orders** and **Order Lines** data sources.

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

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

Note : The first 5 columns are from **Orders** table and the last 4 Columns are from **Order Lines** table

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:

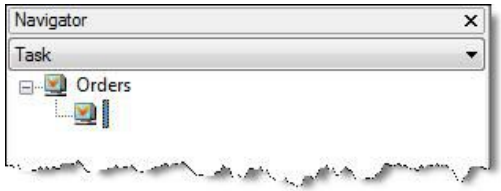
8. Set the **Order_Number** to **6**.
9. Set the **Line_Number** to **3**.
10. Set the **Product_Price** to **6.2**.
11. Set the **Product_Quantity** to **3**.

Establishing the One-to-Many Data Relationship

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.



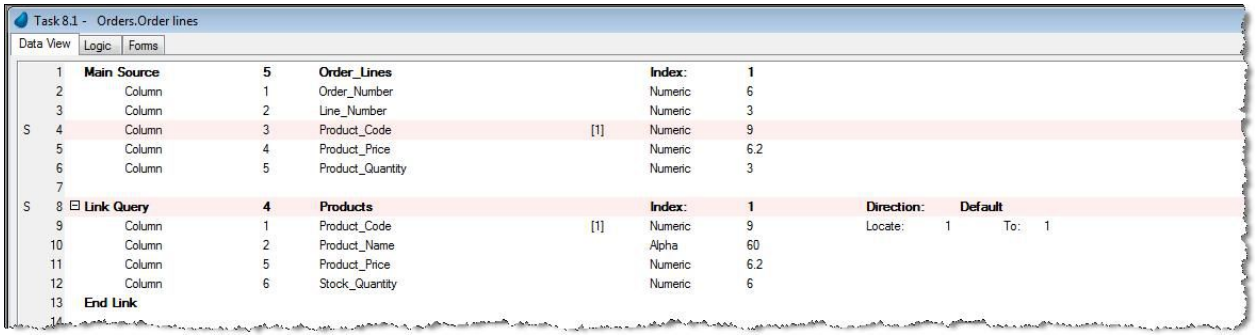
The **Task Properties** dialog box (of the new task) opens.

4. In the **Task name** property, type: **Order Lines**. This is automatically defined as a **Online** task because the parent task is a **Online** 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)→not needed**.
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.



Maintaining Data Integrity

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:

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: **PR: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 **PR: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**.
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:

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**.

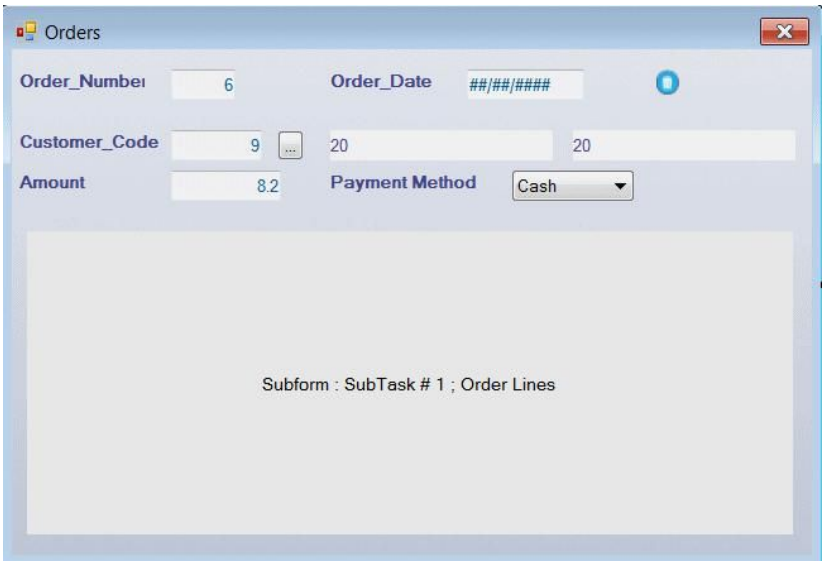
You have almost finished creating the program. What is left is to be able to add a row and to change a row.

Subform Control

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.



You can add the **Add Lines** program.

1. The **Task Properties** dialog box (of the new task) opens. In the **Task name** property, type: **Add / Edit Line**. Set an **Online** task. This task will be called when you edit a line and when you add a line
2. In the Data View Editor, add 4 **Parameters:**
   PR:TaskMode(A1)
   PR:Order_Number(N6)
   PR:Line_Number(N3)
   PR:Amount(*)(N8.2)
3. Set the Main Source to **Order_Lines** and the index to **Lines**.
4. Add all of the **Order_Lines** columns.
5. Zoom into the **Range from** property of the **Order_Number** column and set the range to the PR:Order_Number parameter from the current task.
6. 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.
7. 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 **PR:Line_Number** from the **Order Lines** subtask
8. Set the same expression in the **Range to** expression.

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.

9. Add a Virtual variable with a **Logical** attribute named **RT:ProductExists**.
10. Create a Header line and select **Link Query**.
11. Select the **Products** data source with the **ProductCode** index.
12. Add the following columns from the **Products** data source: **Product_Name**, **Product_Price**, **Stock_Quantity—not needed**.
13. 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.
14. 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 sub program will be in create mode and if not it will be in modify mode.
15. From the **Task** menu, select **User Events**.
16. Create an event named **Select Product** and from the **Force Exit** column, select **Editing**.

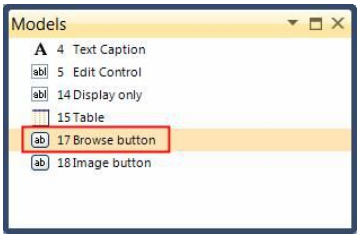
17. In the Logic Editor, create a Header line for the **User** event, **Select Product**.
18. Create a line for the **Call Program** operation.
19. Call the **Select Products** program.
20. From the **Arguments** field, zoom to the Arguments repository.
21. Create a line, zoom and select the **Product_Code** 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



Format:
Width: 3.5
Height: 1.25

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 and **Set the Model**.
7. Add the **Product_Price and the Product_Quantity from Order_Lines.**

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.

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 the **Add / Edit line** program.
7. From the **Arguments** field, zoom to the Argument repository.
8. Create four lines:

1. 'C'
2. Order_Number(from the current task)
3.Line_Number (from the current task)
4.Amount (from the parent task)

Note:You do not need to Raise Event "View Refresh" because of Step 7.

'.

9.Create a Header line for the **User** event, **Change order line**.

10.Create a line for the **Call Program** operation.Select the **Add / Edit line** program.

11.From the **Arguments** field, zoom to the Argument repository.

12. Create four lines:

1. 'C'
2. Order_Number(from the current task)
3.Line_Number (from the current task)
4.Amount (from the parent task)

Note:You do not need to Raise Event "View Refresh" because of Step 7.

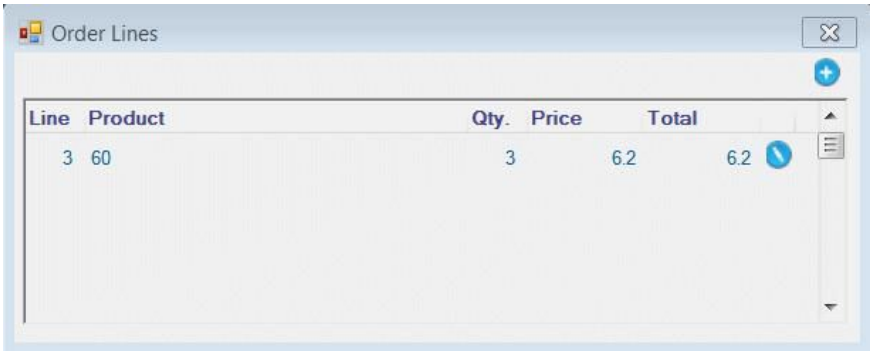
Now you need to add the push buttons to the form that will raise the events.

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 **default image file** 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.

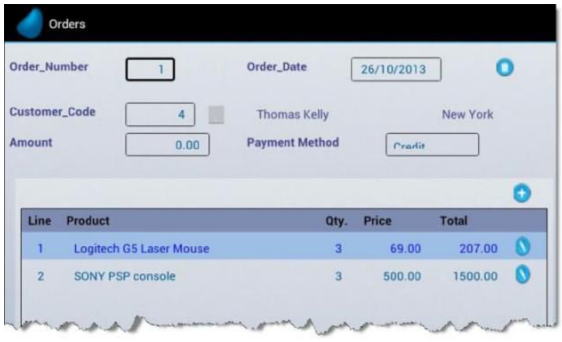
3.The order will open but no lines will be displayed.

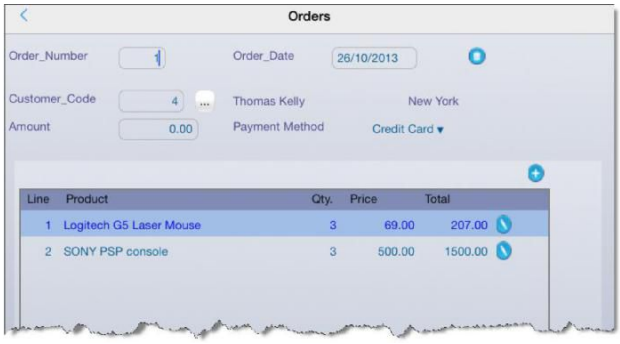
4. Click the Add icon and add the following line:

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





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.

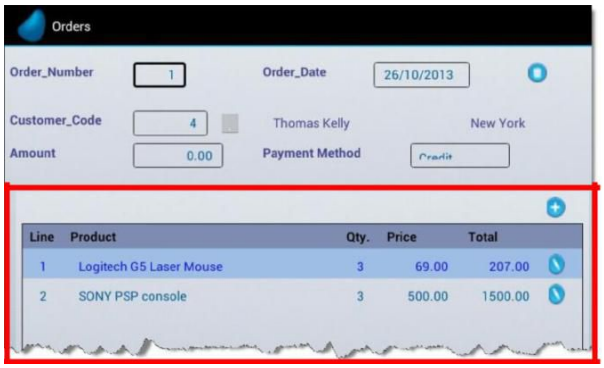
The subform view is also refreshed automatically by Magic xpa. This occurs when the value of one of the passed parameters is changed and the **Automatic Refresh** property is set to **Yes**.

Subtask Form Transparency

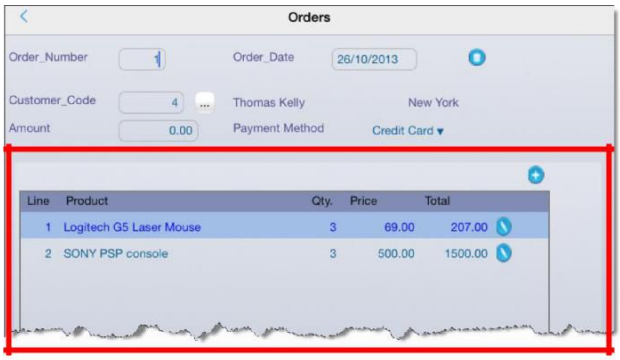
Look at the **Orders** program while it is executing.

Can you see that there are actually two wallpapers: one for the **Orders** task and one for the **Order_Lines** subtask?

Android



iOS



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.

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 **GUI Display table** property.
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 **False.**
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 **PR:Amount(*) parameter** 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**.Set also the model to **Display Only.**

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**).

Click the edit button on order line number 3.

Change the **Product Price** from **199.95** to **340**.

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.

Magic® University

Lesson 14

Non-Interactive Processing

Then, you will learn about the Magic xpa Batch task.

This lesson covers various topics including:

- How the Magic xpa engine works in Batch tasks
- The differences between **Online** tasks and **Batch** tasks
- When to use Batch tasks
- How the Magic xpa engine processes Batch tasks
- Maintaining data integrity using a Batch task

Batch Programming

Up until now you learned about **Online** 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.**

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

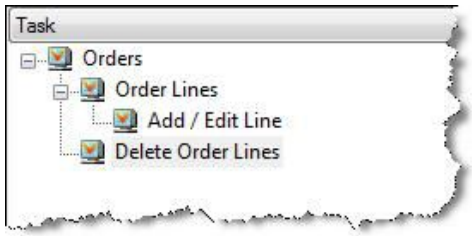
Rich Client Task	Batch Task
Enables end-user interaction.	Does not allow end-user interaction.
Enables the end user to navigate through the Main Source records. Only the records that are scrolled are scanned.	The task scans all of the Main Source records within the range criteria. If no Main Source is 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

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.



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.

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

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 Online tasks and Batch tasks. You also learned about the Batch task's engine cycle compared to the Rich Client task's engine cycle.

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.

Magic® University

Lesson 15

Reports

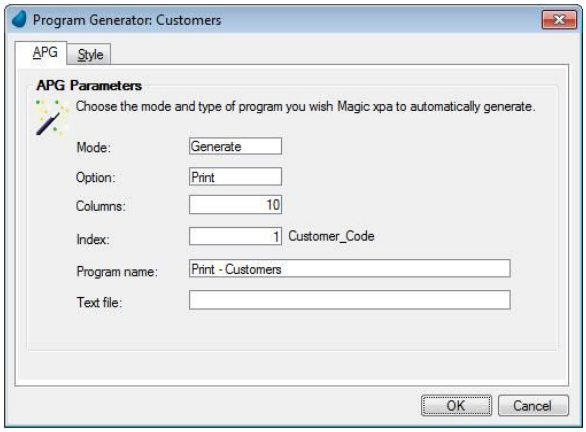
This lesson covers various topics including:

- 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.

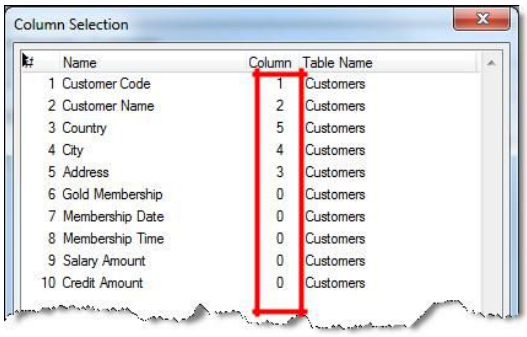
1. Open the Data repository.
2. Park on the **Customers** data source.
3. From the **Options** menu, select **Generate Program**.
4. The **Program Generator** dialog box opens.
5. In the **Mode** field, select **Generate**.
6. In the **Option** field, select **Print**.
7. Park on the **Columns** field.
8. Zoom to the **Column Selection** window.



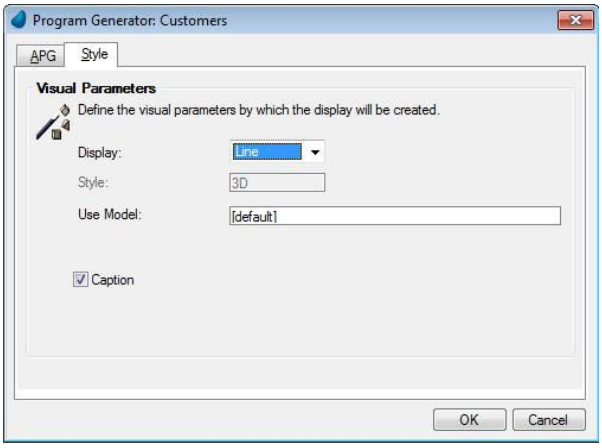
The value in the **Column** column shows the position of the column on the form.

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**.
12. Click the **Style** tab.
13. Select the **Caption** check box.



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.

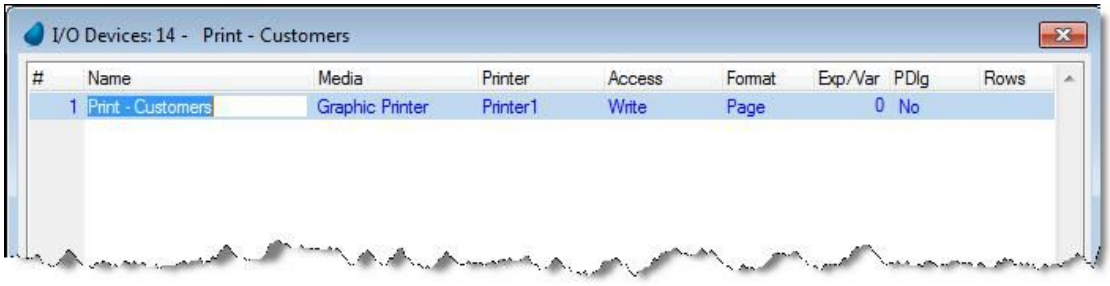
5. In the Data View Editor, set the **Main Source** to **Customers**.
6. Set the **Index** to **Customer_Code**.
7. Add the first 5 **Customers** columns from the data source to the Data View Editor.

Defining I/O Devices

When running a report on a desktop computer, the output is printed on a printer connected to the computer.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.

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**.



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** . Magic xpa has a logical name, **%TempDir%**, which returns the temporary folder.

5. Create an expression for the file name:
   '**%TempDir%CustomerList.pdf**'

   Usually, %TempDir% is

   "C:¥Users¥training¥AppData¥Local¥Temp¥".

   But you can directly use the path of
   "C:¥Temp¥CustomerList.pdf"
   in this lesson.

6. Open the **Print - Customers** I/O Properties (**Alt+Enter**).
7. Set the **PDF** property to **Yes**.

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**.

Note that the **Class** column value is larger than zero.

5. From the **Interface Type** column, select **GUI Output**.

You will now set the form properties according to the explanation above.

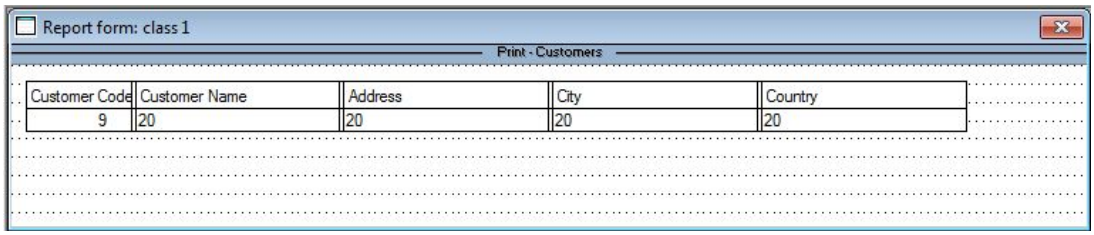
6. Open the **Form Properties** sheet.
7. In the **Form units** property, select **Centimeters**.

8. In the **Width** property, type: **20**. This will give the form a 1-cm margin.
9. In the **Height** property, type: **3**.
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**.

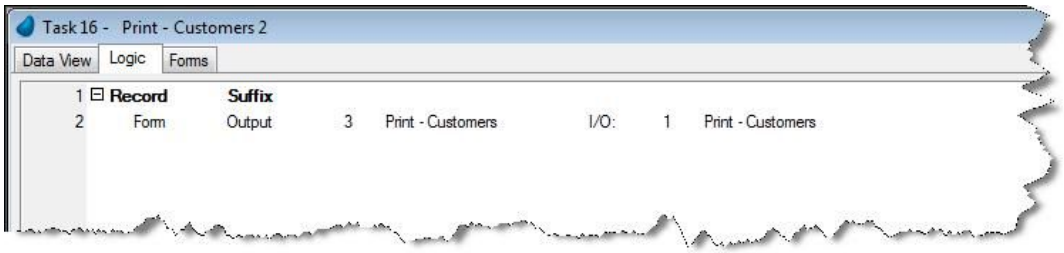
The form should look like the image below.



14. Exit the Form Editor (**ESC**).

Using the Form Output Operation

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.



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

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 Options menu, select “Setting”-“Colors”
2. Park on the last line and create an entry.
3. In the **Name** column, type: **Report Header**.
4. Zoom to the **FG** column.
5. In the **System** entry, select the blank (first) option.
6. In the **Basic Colors** area, define the following shade of blue:
   RGB = 0,64,128.
7. Return to the Application Color repository.
8. Zoom to the **Report Header: BG** dialog box.
9. Check the **Transparent** box.
10. Return to the Application Color repository.
11. Click **OK** (to return to the **Application Properties** dialog box).

In the **Save As** dialog box:

12. In the **Effective immediately** entry, select **Yes**.

Adding a New Font

1. Park on the last line and create a new entry.
2. In the **Name** column, type: **Report Header**
3. Zoom from the **Font** column.
4. 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.
5. In the **Font Style** section, select **Bold Italic**.
6. In the **Size** section, select **18**.
7. Return to the Application Font repository.

In the **Save As** dialog box:

8 . In the **Effective immediately** entry, select **Yes**.

Modifying the output

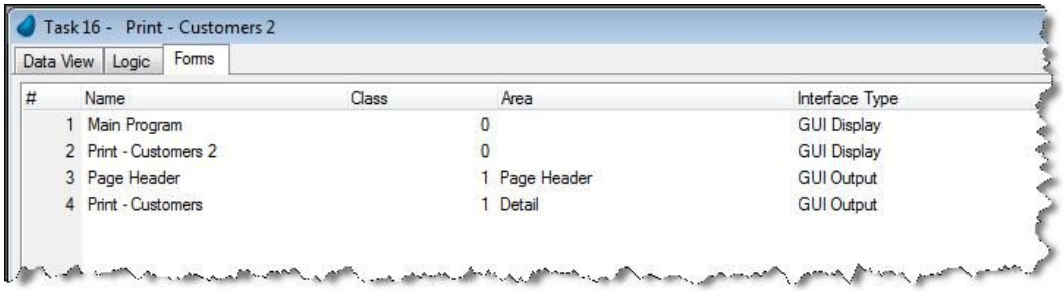
You will now update the color and font for the table's column titles.

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.

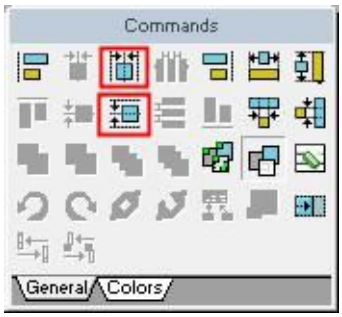


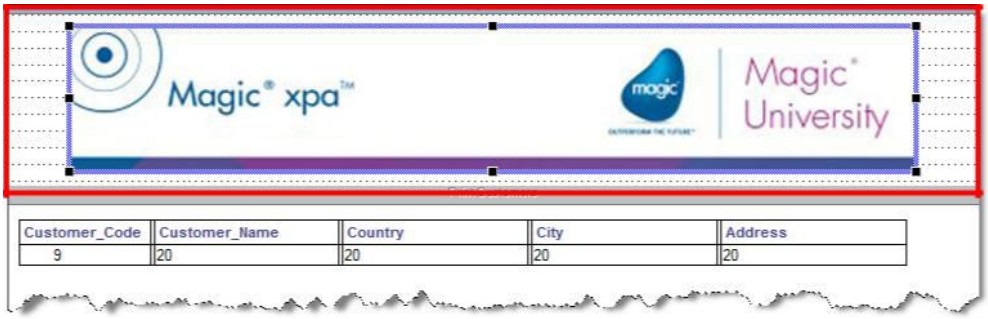
The physical placement of Header, Detail, and Footer forms in the Form Editor is important. This form should also use centimeters as the form unit and have a width of 20. Additional forms in this course will also have these settings. Instead of manually changing this for each form, a better way to do 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**.
9. Zoom to the **Page Header** form.
10. Place an **Image** control on the form.**(Print_Logo.jpg)**
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.

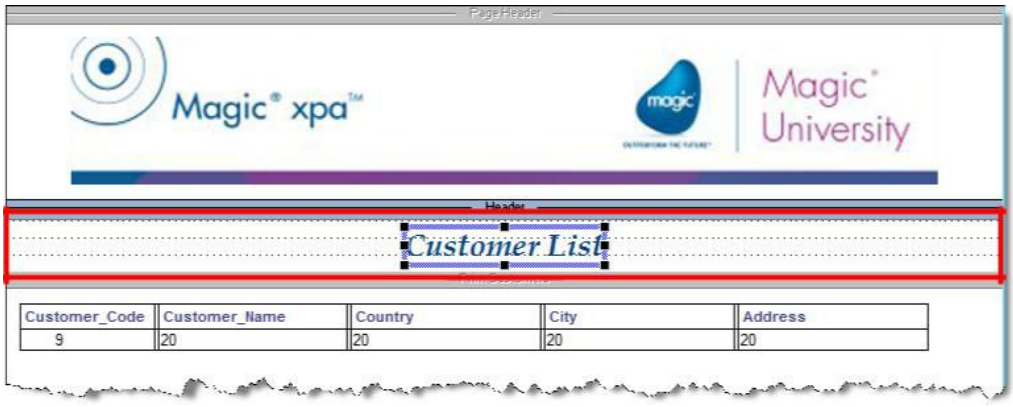




Adding a Report Header

A report header is used to provide a descriptive caption for the report.
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.



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**.

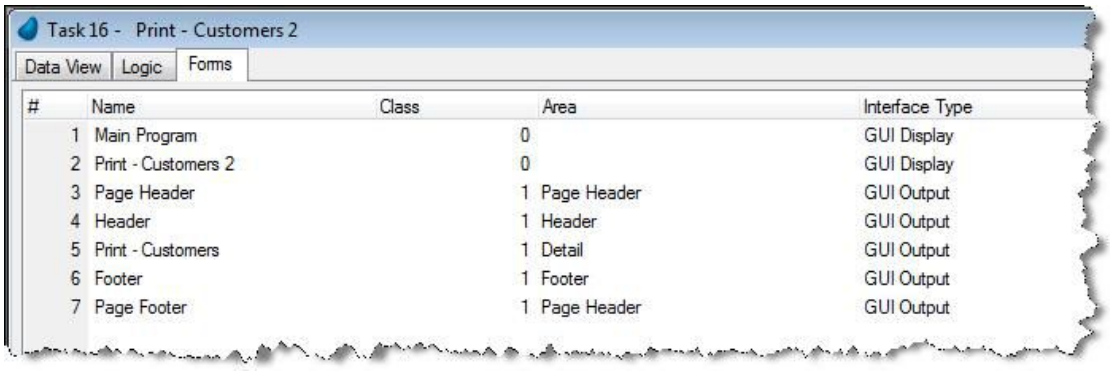
In this example, you will display the total number of printed customers in this report. The **Counter (0)** function returns the number of records processed in the current task. You will use the **Counter (0)** function to return the number of customers printed in the report. The Counter function is only relevant for Batch tasks and non-interactive Online 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**.



#	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. Select **Print_Footer.jpg**.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.

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.

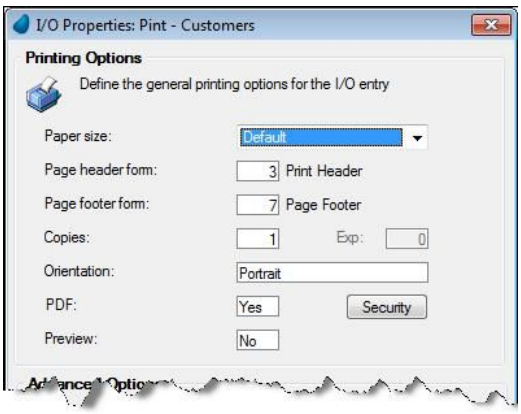
The Footer form is the report footer and it should be printed once on the last page of the report. Therefore, it should be printed from the Task Suffix logic unit, which is executed only once when the program ends.

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 **Page 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.



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**. 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.

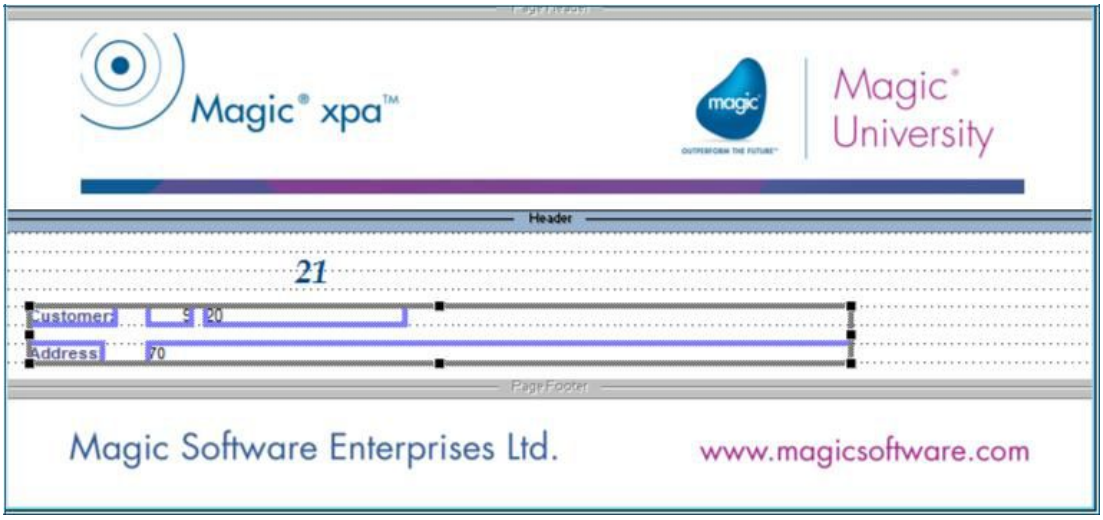
Now you will create the Header form. This form will contain the report name as well as the order title information.

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
    a. In the **Text** property, type: **Amount:**
    b. From the **Font** property, select the **Text Caption** font.
    c. 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- Invoice** 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.
9. Check syntax and excute the program.Open the temp folder:”C:¥Users¥training¥AppData¥Local¥Temp if **“Invoice.pdf”** exist.

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.

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.

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.

Magic University

Lesson 16

Processing Data by Groups

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

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

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 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 Caption.**
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

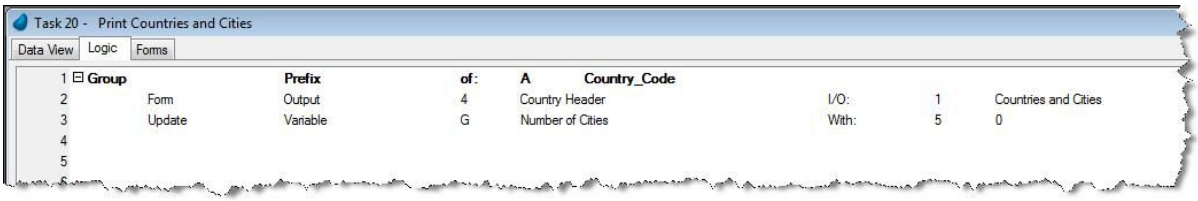
1. From the **Task** menu, select **I/O Devices**.
2. Create a line and set the **Name** to **Countries and Cities**.
3. From the **Exp/Var** column, zoom and set the following expression:

   '%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** operation and print the **Country Header** form.
4. Add an **Update Variable** operation and update the **Number of Cities** variable with zero. You are resetting the counter every time the **Country_Code** changes.



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

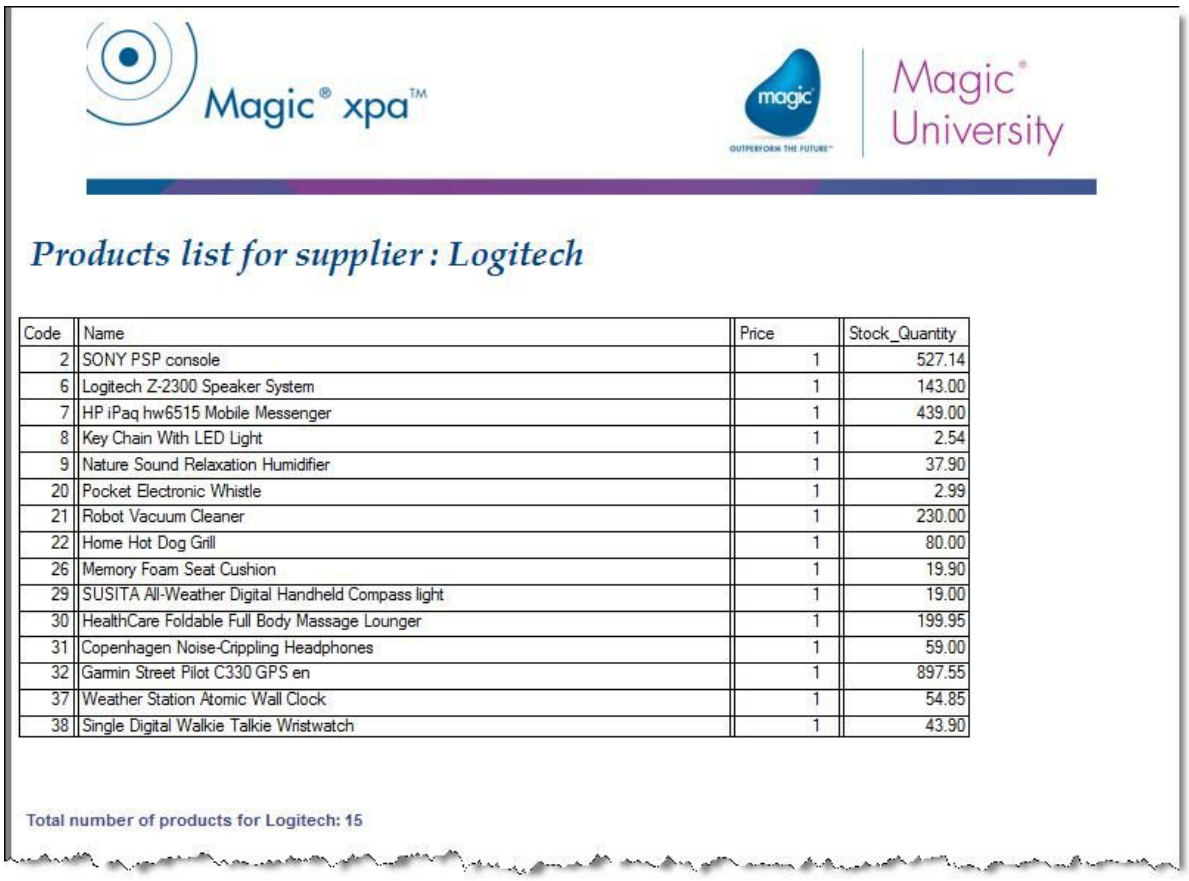
You can now run the program. You can also execute the program using the same methods that you used in the previous lesson, such as adding a Print icon to the **Countries** program.

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.



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

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.

Magic University

Lesson 17

Menus

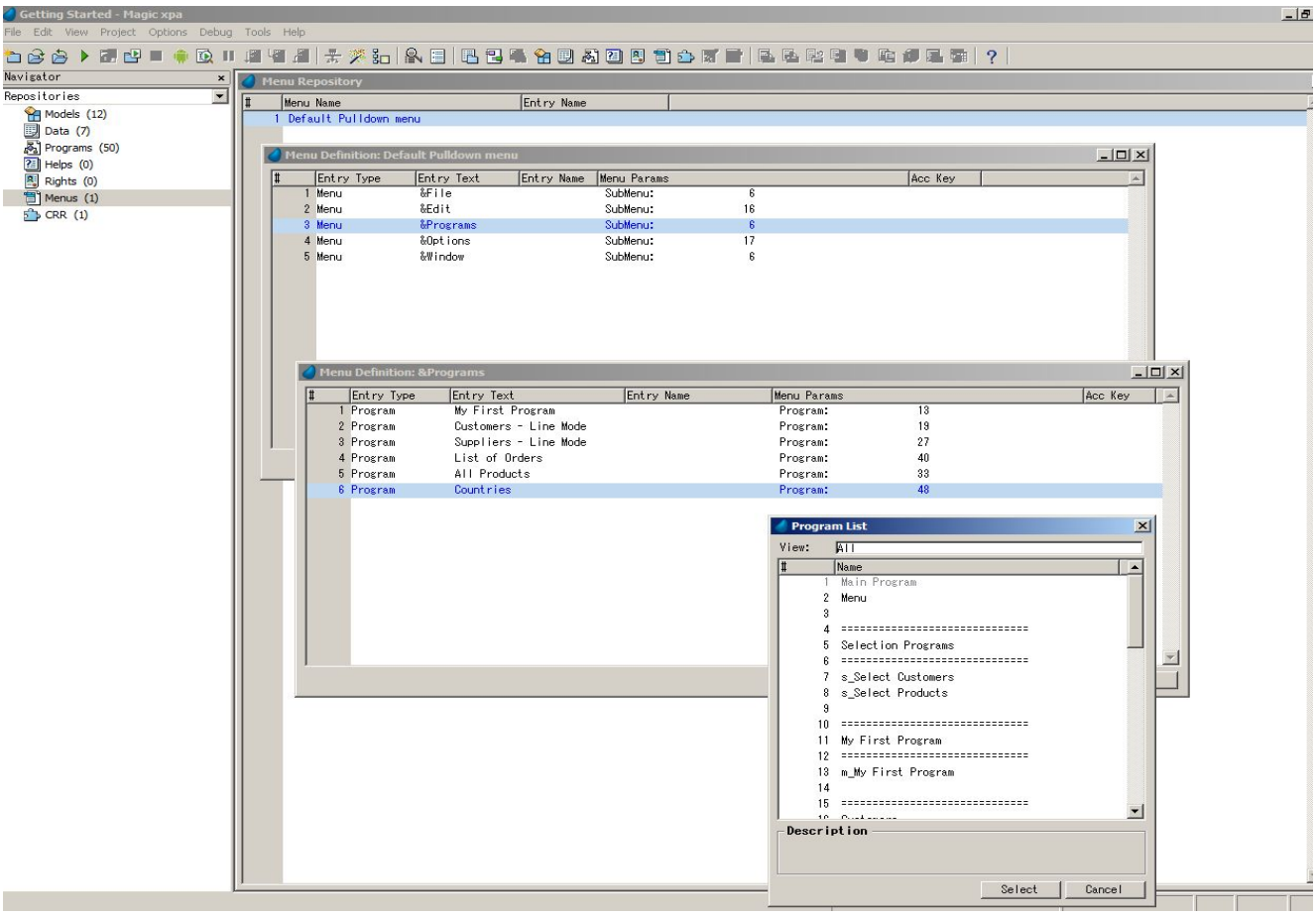
This lesson covers various topics including:

- What menus are used for
- Defining pulldown menus

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).
2. Zoom on **Default Pulldown.**



Lesson 18

How to Debug

This feature can help you pinpoint problems that you are having in an application.

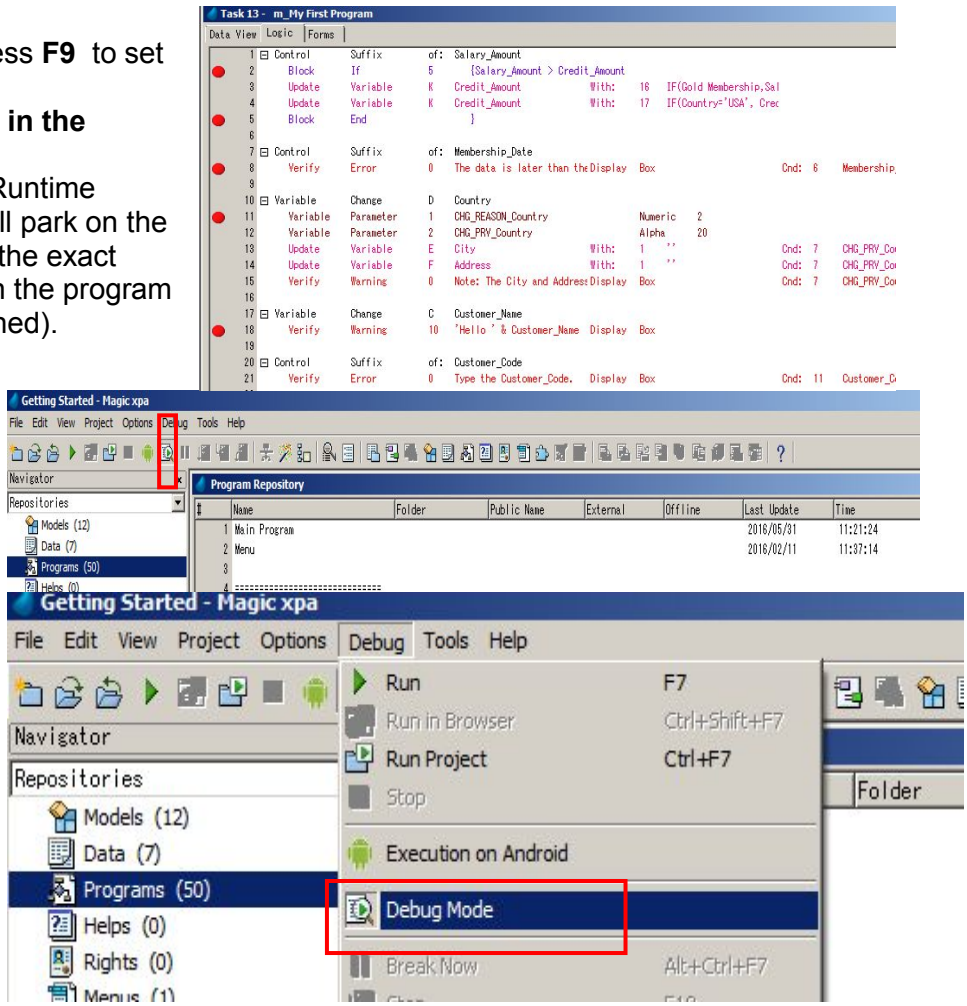
When you are in **Debug mode** (you selected **Debug Mode** from the **Debug** menu), you can run a program in Runtime mode and the Debugger will park on the Studio read-only screen at the exact location of where you are in the program (when a breakpoint is reached).

Breakpoints enable you to halt execution of the program at a specific location in the Studio screen.

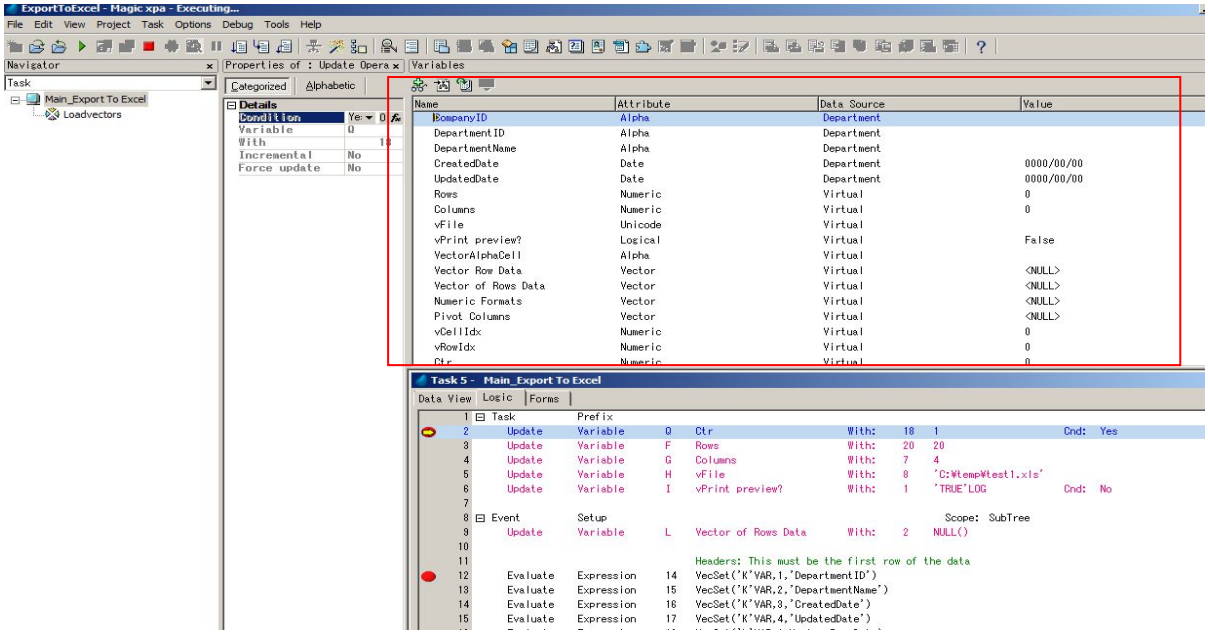
View the Magic xpa flow within the Studio

1. Create a breakpoints or press **F9** to set and remove breakpoints.
2. Make sure the Dubugger is **in the Debug Mode.**
3. You can run a program in Runtime mode and the Debugger will park on the Studio read-only screen at the exact location of where you are in the program (when a breakpoint is reached).

4. Press **F10** to move the cursor to the next breakpoint.



5. Yo can also view the value of each variables using **View Menu--View Variables**



* TEACH ALSO THE PROGRAMMING RULES IN XPA

* HOW TO RELEASE

* GIVE THE ACTIVITIES

ENDING OF TUTORIALS

CONGRATS STUDENTS AND TEACHERS!!