Learning MS Dynamics AX 2012 Programming

Develop and customize your very own Microsoft Dynamics AX solution quickly and efficiently

Mohammed Rasheed
Erlend Dalen
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 “Understanding Dynamics AX 2012”
- A synopsis of the book’s content
- More information on Learning MS Dynamics AX 2012 Programming

About the Authors

Mohammed Rasheed is a Dynamics AX solutions architect and a self-proclaimed Dynamics AX evangelist. He has had a remarkable career as a Dynamics AX expert and has made key contributions to the success of a number of leading Microsoft partners such as IBM (UK), HSO (UK), and Crowe Horwath LLP (USA).

Listed as one of the most influential people by http://www.dynamicsworld.co.uk/, Mohammed considers himself fortunate to have met and worked with the most amazing team of people. He attributes his success to his parents, his family, and his team—a team of leaders with the zeal to redefine an industry, managers with the resolve to achieve their aspirations, and young consultants with limitless energy.

Mohammed calls Chicago his home, where he lives with his wife, Sakeena, and their beautiful baby daughter, Zara.
Erlend Dalen started working as a developer with the first version of Axapta in 1998. He was then working with some of the first Axapta implementations in Norway and was also part of a team that created an advanced product configurator for Axapta. From 2000 to 2002, he worked on developing e-commerce, mobile, and integration solutions in Java for a Norwegian IT consultancy company. He joined Columbus IT in 2002, working first as a senior developer in Norway and in the USA and then as the technology manager of the Norwegian branch, where his responsibilities included implementing new technology areas, creating an e-commerce solution for Dynamics AX, and being the technology solution architect in internal and external projects.

For the past four years, Erlend has been working as a web developer for Norsk Test AS, where he spends his days programming in Python using the Django web framework.

I would like to thank my wife, Benedikte, for backing me up on this project. Whenever I felt that things moved slowly, she motivated me to push forward to finish the project. I would also like to thank Columbus IT, Norway, for letting me take some time off to write the book and my coworkers at Columbus IT, Norway, for helping me out whenever I had a question. Last but not least, I would like to thank my daughter, Ellinor, for forcing me to take breaks every now and then. You put a smile on my face when I was struggling with the book.
Microsoft Dynamics AX 2012 is unique in the fact that it is an incredibly utilitarian ERP system as well as a powerful platform. The purpose of this book is, quite simply, to help new developers and keen consultants to explore the platform side of AX.

Erlend Dalen, the author of the first edition (Microsoft Dynamics AX 2009 Programming: Getting Started), and I have tried to make it easier for the reader to understand AX concepts by providing real-world examples of focusing our energy on elements that matter the most. For example, AX has a useful metadata-based development that enables developers to activate most of their goals without actually writing any code.

I thank you for your interest in Microsoft Dynamics AX and I hope you enjoy reading this book as much as I have enjoyed writing it.

What This Book Covers

Chapter 1, Understanding Dynamics AX 2012, takes you through the development environment in AX and explains some of the development tools that are included in AX. You will also be able to write your first AX program, if you have never done so before.

Chapter 2, The X++ Language, covers the basics of the AX programming language that is X++ and explains in brief how data types, statements and loops, operators, classes, methods, and macros work.

Chapter 3, Storing Data, takes you through the process of creating extended data types, tables, and relations.

Chapter 4, Data User Interaction, shows you how to create forms where users can read, update, insert, and delete data. You will also learn how to create reports, menu items, navigation pages, and menus.

Chapter 5, Searching for Data, explains different options that you can use when you need to search and retrieve a set of data from the database. It shows you how to create a query, how to create a view, and how to write different select statements.

Chapter 6, Manipulating Data, explains how to insert, update, and delete data using table methods to manipulate one record at a time and discusses set-based manipulation to manipulate a set of data.

Chapter 7, Integrating Data, explains how to read and write data to and from different kind of files and to and from a database using ODBC. It also provides an example of how to write a generic import/export program in AX.
Chapter 8, *Integrating with Standard AX*, explains how some of the main modules in AX (Inventory, Ledger, Account Receivable, and Accounts Payable) are built by discussing their entity schemas. You will also see examples of how to achieve typical tasks within these modules.

Chapter 9, *Creating a New Module*, explains how to create a new module in AX by creating number sequences, parameter tables, license codes, configuration keys, and security keys.

Chapter 10, *Working with .NET and AX*, explains how you can use .NET classes in AX with the Common Language Runtime and how you can write .NET code that use AX classes by using .NET Business Connector for AX.

Chapter 11, *Web Services*, explains how to create a web service that exposes the AX logic, how to publish the web service to Internet Information Services, and how to make AX consume a web service.

Chapter 12, *Enterprise Portal*, explains how to expose AX data to web by using the Enterprise Portal and SharePoint.

Appendix A, *Links*, provides useful links related to Dynamics AX.

Appendix B, *Debugger*, gives you a quick guide on how to use the debugger.
In this chapter, you will learn how the development environment is structured and what tools are accessible to the developer in Microsoft Dynamics AX. The famous Hello World code will be a piece of a cake for you to understand after the first step-by-step walkthrough in this book, and you will know what the different nodes in the application object tree represent.

Here are some of the topics you will learn in this chapter:

- A tour of the development environment
- A look at the tools available
- Creating a job that prints something on the screen
- An overview of the AX architecture

Understanding the development environment

Let's have a look at the main features of the Dynamics AX development model:

- The AX programming language
- The Application Object Tree
- The X++ code editor
- The compiler
- Labels
Programming language
The programming language in AX is called X++ and its syntax is similar to Java and C#. In addition to being an object oriented programming language, it also includes embedded SQL. This means that writing SQL statements in AX is very easy, because as a developer, you don't need to create a connection to the database, then create a statement that is executed on the connection, and so on. Instead, you can write the SQL statements directly into the X++ code, much like you now can with language-integrated query (LINQ) for .NET.

You will get to know the syntax, features, and how the X++ language works in Chapter 2, The X++ Language.

MorphX
In addition to the programming language, Dynamics AX provides an integrated development environment called MorphX. It is all about visual development in AX and it lets the developer create code elements in a graphical way, much like Visual Studio, by dragging-and-dropping and setting properties on an element. This also is the recommended approach to development in most cases. Most of the AX elements can be edited using MorphX.

Application Object Tree
The Application Object Tree (AOT) is where you find all code elements in AX.

You can open it by clicking on the AOT icon, shown in the following screenshot, from the top menu or by pressing Ctrl + D from anywhere in AX; this will launch the development workspace.
You will then see the AOT presented in a tree view, as shown on the right-hand side in the following screenshot:

![AOT Tree View](image)

**Data Dictionary**

The AOT is organized to have all elements that relate to how the data is stored in the database and those that relate to security, such as **Security Keys**, **Configuration Keys**, and **License Codes**, under a node called **Data Dictionary**. Under this node, you will find **Tables**, **Maps**, **Views**, **Extended Datatypes**, **Base Enums**, **License Codes**, **Configuration Keys**, **Security Keys**, **Table Collections**, and **Perspectives**.
Macros
Macros are predefined values used throughout the solution, and they are just mechanisms by which a value (a bit of string) can be defaulted within your code, so that you won't have to hardcode values.

Classes
In the development world, an object is a self-contained component that contains properties and methods needed to make a certain type of data useful. An object’s properties are what it knows and its methods are what it can do.

Objects are the fundamental building blocks of applications from an object-oriented perspective. You will use many objects of many different types in any AX function you develop. Each different type of object comes from a specific class of that type.

A class is a blueprint, template, or a set of instructions to build a specific type of object, and every object is built from a class.

Forms
Forms are UI elements of Dynamics AX. They can open windows to search for data, view data, change data, create new records, or delete records. In addition, forms very often contain buttons that link to other forms, reports, and classes that execute business-logic (it is not recommended to write business logic within forms; classes are the right place for this).

Datasets
Datasets are used by Visual Studio components such as Reporting Services reports and AX User Controls, as you will see in later chapters. The datasets define which tables are used by these Visual Studio components by adding the tables to the datasets, much like you would do with a table in a data source.

SSRS Reports
All AX 2012 reports are executed and rendered using SQL Server Reporting Services. This node holds all the SSRS reports that will be used within AX.

Reports
Reports are the standard AX reports that display data in the form of reports to the users or write the data to a printer or file. This is available as a node to manage legacy reports only and is not the recommended approach to create new reports.
Report Libraries
Report libraries are links to reporting projects in Visual Studio. You can't create a report library directly from AX. You have to create a report project in VS first and import it into AX. Then it becomes a report library. This is available as a node only to manage legacy reports and is not the recommended approach to create new reports.

Queries
Queries in the AOT are predefined static queries that can be reused throughout the whole application. A query contains a hierarchy of data sources that defines which data should be available for the query. Queries can also be used by other elements in AX, such as forms, reports, and classes.

Jobs
Jobs are static methods intended to directly execute a small piece of code. They work fine for code that is supposed to execute just once, but are not intended to be part of a solution. Jobs are typically used to correct/update a particular table/record that might have been incorrectly modified in the data migration process.

Menus
Menus present the end users with options to open the most used forms and reports and to start periodic tasks. Menus are effectively groupings of forms; end users typically view menus as modules (that is, functional area).

Menu Items
Menu items are pointers to forms, reports, and classes. There are three different types of menu items, namely, Display, Output, and Action. The display menu items are used to open forms, the output is used to open reports, and the action is used to execute a class with a main method (startpoint).

Web
The Web node consists of subnodes that in turn hold different types of elements relevant for the development of web applications such as the Enterprise Portal (EP).

Services
Services are used to set up web services that let other applications use AX classes through the Application Integration Framework (AIF). The AIF is used for electronic communication with trading partners or applications.
Workflow

The workflow framework in AX lets developers create workflows that define certain execution flows, depending on certain events. The typical scenario for using a workflow in AX is the purchase requisition where one employee enters a purchase order. If the purchase order is over a certain limit, it has to be approved by a manager. If denied by the manager, a message is sent to the employee who created the order. If approved, the order is automatically sent to the vendor.

Resources

Resources are files that are imported into the AX database and can be referenced in the code from the resources node in the AOT rather than using a directory and filename from the disk. These files are typically images, icons, cascaded stylesheets, and XML files. You can preview the files by selecting a Resources node in the AOT, right-clicking it, and selecting Open.

System Documentation

At the bottom of the AOT, you can find help in the form of system documentation. It consists of documentation about the AX core. The elements found here are not open source. This means that you, as a developer, have to relate to these elements as black boxes. This means that only you know what to put into methods and what you can expect to get back. You can find information about global functions here, such as strlen(). This function and most other functions are documented with an explanation of the input parameters, return value, and an example of how to use the function.

The same goes for core classes and their methods. Typical core classes are Map, Array, AsciiIo, FormRun, and XppCompiler, just to mention a few.

Application Developer Documentation

Application Developer Documentation consists of documentation regarding standard AX tables and classes. This documentation is basically just another way of displaying information that you as a developer should be able to find by browsing the AOT, reading code, and looking at properties.
Properties

So, how do I look at properties, you might ask. Well, at any time while you are in the AOT, you can hit Alt + Enter or right-click and select properties to open the **Properties** window, as shown in the following screenshot:

![Properties window screenshot](image)

Each element type in the AOT will have a fixed set of properties that can be changed by the developers. The properties marked with bold text have been changed specifically for the selected element. The pink background you can see in the **Name** property in the preceding screenshot means that it is mandatory. The yellow background means that it is best practice to have something in that property. The rest of the properties are defaulted by the system.
The properties are by default organized by relevance, relative to the element they represent. This sorting can, however, be set to alphabetical order from the user options form. Click on the Microsoft Dynamics AX button (Alt + F) and navigate to **Tools | Options**, to open the user options form. Under the **Development** tab, you can enable the **Sort alphabetically** checkbox to sort the properties alphabetically, as shown in the following screenshot:

### X++ code editor

The X++ code editor that ships with AX 2012 is based on Visual Studio's code editor and hence, it supports color coding, word completion, automatic indenting, scripting, zoom, multiline editing, and many other features that are available in Visual Studio. This is where all the X++ code will be written.

X++ code editor consists of three sections; method list, menu bar, and code window. The method list shows all the methods in the selected element if you double-click an element. If you double-click a method within an element, you will only see that method in the method list. If you then double-click another method in the same element, you can see both the selected methods in the method list. When you change a method, it displays an asterisk after the method name in the method list, to show you a changed method that has not yet been saved.
The code window obviously shows the X++ code for the selected method in the method list:

The menu bar shown in the preceding screenshot has the following options, listed from left to right:

The available options are explained as follows:

- **New**: This option creates a new method within the selected element.
- **Save**: This option saves all the methods open in the method list.
- **Run**: This option executes the method/class if it is executable.
- **Breakpoint**: This option sets a breakpoint at the line where the cursor has been placed.
- **Enable/Disable breakpoint**: This option enables or disable a breakpoint.
- **Remove all breakpoints**: This option removes all breakpoints for the current user.
- **Compile**: This option compiles and checks for errors in the code.
• **Lookup properties/methods:** This option displays help for the selected text if the cursor is placed over a core function/method; for example, list methods for a class, available values for enums, and so on.

• **Lookup label/text:** This option opens the label editor and searches for the selected text.

• **Scripts:** This option opens a context menu, which consists of small code snippets such as the do-while loop, header comments, and so on. You can also create your own scripts by creating new methods in the `EditorScripts` class.

• **Source Code Controls:** The following five options are to manage source code (version control):
  - **Add to Source control:** This option enables version control for the object
  - **Check Out**
  - **Check In**
  - **Undo Check Out**
  - **History**

• **Help:** This option displays the help file for the selected text in the code window, if such a help file exists. If not, it displays the help file for the X++ code editor.

• **Display Changes:** This option is similar to Visual Studio's option with the same name, wherein changes will be color coded in the editor.

• **Display Line numbers:** This option displays line numbers on the left-hand side of the code editor.

**Compiler**

The compiler in AX is in many ways just like any other compiler; it checks for syntax errors and presents the developer with error messages if there are any syntax errors. It can also give warnings if it sees any code that it finds misplaced or not written according to rules followed as best practices.

One of the major differences between the AX compiler and other compilers (such as the Visual Studio's .NET compiler) is that the AX compiler only requires you to compile the code that you have changed. The problem with this is that the code you have changed can make classes that consume your changed code to stop compiling. If they are not recompiled, and probably changed as well, users will experience a runtime error.
To decide how the compiler should act when compiling the code, you can look into some options provided to you when you change the compiler parameters.

First of all, to change the compiler parameters, you have to open the user options form (click the Microsoft Dynamics AX button (Alt + F) and select Tools | Options). From the Development tab in the user options form, click on the Compiler option to open the Compiler setup form, as shown in the following screenshot:

Here you check the Cross-reference flag and you are good to go. Remember that compiling code in AX will take a lot more time when it has to update the cross-references as well. Refer to the Cross-references section for more information regarding the cross-references. The fields in this form control how the X++ compiler should act. The code can be set to give compiler warnings at different levels, by changing the Diagnostic level option. You can choose not to receive any warnings about strange looking code (such as having a line of code after a return statement in a method). Compiler errors will, of course, still show. Level 4 means that the compiler will also run checks for best practices.
The output defines where the compiler messages should appear. They can appear in a compiler output form such as the one shown in the following screenshot:

![Compiler output screenshot](image)

Alternatively, messages can appear in a print window:

![Message window screenshot](image)

The compiler output form is set as the default because it contains more information and has a more standardized look and feel with division into the following tab pages:

- **Status**: This page lets you see the overall status of what you have compiled
- **Errors and warnings**: This page displays any errors or warnings, depending on the diagnostics level set in the compiler setup form
- **Best practices**: This page shows the best practice warnings as specified in the best practice parameters form
- **Tasks**: Tasks are added in the code by writing a comment that has the `TODO` statement:

```csharp
// TODO: Remember to fix the for-loop
```
Labels

The labels in Dynamics AX are a way of translating all user-visible text by only referencing to a label ID in the code. Each language is represented by a language-specific label file, and the file itself is stored in a database along with the AX models. These label files consist of the label ID and the text for that label in that specific language. When opening a label file in a text editor, you can see labels listed, such as the following:

@SYS390 Quantity that is not yet cost accounted in the BOM unit.
@SYS400 Create and compose serial numbers automatically.
@SYS403 Finish must be greater than start.
@SYS418 Transfer customer information?

This means that if you write @SYS403 in the label property of a field in AX, it will display Finish must be greater than start to the users who are viewing that field in a form or report if they are using the language code en-gb (as this was taken from the label file AxSYSen-gb.ald).

To create a new label file, simply start the label file wizard from the Microsoft Dynamics AX button (Alt + M) and navigate to Tools | Development tools | Label file wizard and run through the three self-explanatory steps of the Wizard.

Your label file will then be named AxXXXen-gb.ald, where XXX is a three letter abbreviation chosen when walking through the Wizard.

Creating your first AX program

To make the Hello World example, we simply open the AOT by pressing Ctrl + D from anywhere in the main Dynamics AX window, or pressing the AOT button in the top menu, as shown in the first screenshot of this chapter. Then perform the following steps:

1. Go to the Jobs node in the AOT, right-click on it and select New job.
2. Check if the job opens automatically.
3. A new X++ editor window will open, showing a new job:

```x++
static void Job1(Args _args)
{
}
```
As a job is static and has an input parameter of the `Args` class, it can be run directly. Obviously, the preceding example doesn't do anything, but simply editing the job as shown in the following code will display a `HelloWorld` message in the print window, after running the job by clicking on the Go button or pressing F5.

```c
static void HelloWorld(Args _args)
{
    print "HelloWorld";
    pause;
}
```

The `pause` statement will halt the execution of the program and ask you if you would like to proceed. It is used in this example so that you will see what's in the print window. If the `pause` statement hadn't been put here, the print window would print `HelloWorld` and just close again in the blink of an eye.

**Utilizing the different development tools**

AX comes with many different tools that can help developers in doing their job more efficiently. Some of these tools are however rarely used because they have little value. We will now have a look at some of the tools you will use on a day-to-day basis when programming in AX. Some of the tools you need to know about are as follows:

- Cross-references
- MorphX version control
- Debugger
Cross-references

Cross-references let developers select an element in the AOT and ask the system where the element is used. The result is listed in a form and shows if the element is used for read or write, the names of the elements it is called from, the methods using it, and where it is used in the methods (line/column). Cross-references also enable developers to see all elements used by the selected element. Of course, the best use of the cross-reference tool is to be able to see where a particular element is used.

To be able to see the cross-referenced data, you need to update cross-references on a regular basis, at least if you want to see it in a development environment at a point in time where changes are being made. This can be done by setting up the cross-reference update to run in a batch, or you can set cross-references to update each time you change an element. To set it up to update every time you change an element, you need to change the compile parameters.
First of all though, you have to run a full cross-reference update. This is done by selecting the Microsoft Dynamics AX button (Alt + F) and navigating to Tools | Development tools | Cross Reference | Periodic | Update. You will see the following window:

![Microsoft Dynamics AX](image)

You can choose to delete all existing cross-references before creating them again, update cross-references for all elements in the AOT, or select which elements you would like to have updated by selecting Selected and clicking on the Select button. You will then get a query where you can filter what kind of elements you would like to update cross-references for.

In addition, there are two other useful types of cross-references that can be updated from this form. Updating the data model will enable end users to filter queries not only for the fields in the query that they are running, but also for fields from other related tables. An example of this is when the end user wants to print the report Cust, which lists customers. Let's say they would like to limit the report to show only the customers who have ever created an order. He would then have to add the SalesTable to the structure of the report so that it would create an inner joint between the CustTable and the SalesTable.

Updating the type hierarchy will enable developers to look at a class or an extended datatype and see if it belongs to a hierarchy, as seen in the following screenshot:
The left side of the application hierarchy tree shows which class hierarchy the SalesFormLetter_Confirm class belongs to. All classes in AX automatically extend the Object class, as you can see at the top of the tree. The right part of the form will show you all the methods of SalesFormLetter_Confirm and where they are implemented in the class hierarchy.

**MorphX version control**

If you have done development in Visual Studio on projects with multiple developers, you have most likely used a version control tool and know the essentials of it. Basically, it is a repository where you can check elements in and out, and track changes done to an element over time. If for some reason you would like to use an older version of an element, the version control enables you to do that as well.

When you check an element out, it is locked for other developers, so only you can make changes to it. When you have made your changes to the element, simply check it in and comment on what kind of changes you have made. It will then be available for other developers to change again. If for some reason one of the changes made to the element creates an unwanted situation, you can simply revert the element to the state in which it was before the particular change was made.
Setting up the MorphX version control

To set up the MorphX Version Control System (VCS) in AX, simply click on the Microsoft Dynamics AX button (Alt + F) and navigate to Tools | Development tools | Version Control | Setup | Parameters.

You should then set up this form, as shown in the preceding screenshot. First of all, you have to select Enable in the source control status. Then you need to select the version control type you would like to use. This book only covers the MorphX VCS as it is the best option for a small-sized project with five or less developers. The main reason for this is that it doesn't require each developer to have a full AX installation locally like the Visual Source Safe and Team Foundation Server do.

If you are creating a new module for AX, or if you are already using the Team Foundation Server, selecting the Team Foundation Server as the version control for AX might be a better option.
When using the MorphX VCS, the repository is actually inside the AX database, so you will always have access to it as long as you are logged on to your AX solution.

You can also choose to have the AOT elements color coded, depending on the VCS status, and to receive a warning when reverting objects, as shown in the following screenshot:

![Version control parameters](image)

After setting these parameters, you can choose to create a repository by clicking on the Microsoft Dynamics AX button (Alt + F) and navigate to Tools | Development tools | Version Control | Setup | Create repository.

This will basically check in all elements in the AOT with a shared comment at that point in time. If you have a new solution without any modifications to the standard AX, this is not needed as you always have the option to go back to the original version in the system layers.
You can also set some rules as to what should be allowed or denied when trying to check in an element into the VCS, by clicking on the Microsoft Dynamics AX button (Alt + F) and navigate to Tools | Development tools | Version Control | Setup | System settings.

In this form, you can select to set rules so that the developers can't check the code that doesn't compile or have best practice errors.

You can also set up a test project and run the code through that test project successfully before being able to check in code.

**Using the MorphX version control**

When you have set up the MorphX VCS, you can start adding elements to the VCS as you start working on them.

To add an element to the VCS, simply select the element in the AOT and click on the **Add to version control** button from the AOT menu bar or by right-clicking the element and selecting **Add to version control**.
When an element has been added to the version control, you will be able to check in the element to add your changes to the repository. This is done by clicking on the **Check-In** button from the AOT menu bar, hitting *Alt + I*, or right-clicking the element and selecting **Check-In**.

You will then see a form consisting of all elements that are checked out (or added to the VCS but not yet checked in). You have to type a description of what you have done to the element before clicking on **OK**. The element will then be checked into the repository.

To see the history of what has been done to this element over time, you can select the element in the AOT and click on the **History** button, or right-click the element and select **History**.
The History form will show the history of a specific element in the AOT. The history consists of all the check ins with comments, user ID, date, time, and version ID.

You can also open the .xpo file from the history form by clicking on the View File button.

XPO is the export format that is used by AX to export and import code. It can be used to transfer code from one AX solution to another. The .xpo files are in text format and can be opened in a text editor for verification.

To open the selected version of the element in an AOT window, click on the Open new window button. If you open a version other than the current one or one that is checked out by someone else, you will only be able to read the code (no changes allowed).

Selecting two versions of the same element enables the Compare button, where you can compare the two different versions to track the changes.

The debugger

The most important developer tool is the debugger. This tool helps you in your search to find what is wrong with your code, if you are one of those developers who write code that doesn't work as expected. Even if you never write code with bugs, you might be lucky enough to find other developers' bugs, and the debugger will then hopefully make you curse and swear a little less.

The debugger contains two main sections, the code section at the top and the information section at the bottom.

The code section is where you see the active code being executed. You can execute the code step-by-step, run it from where you are, stop the code's execution, step into, step over (and even go backwards) in the active method, set breakpoints, toggle breakpoints, and remove breakpoints.
The information section contains separate windows for the call stack, local, global, and class variables, as well as breakpoints, output, and the variable watch.

**Dissecting the AX architecture**

Now that you have seen some of the development tools in AX, let's have a look at how AX is built from a technical perspective.

We will look at the following concepts:

- Application object layers
- Network tiers

**Application object layers**

Dynamics AX 2012 consists of 16 application object layers that contain all the elements you see in the AOT. These 16 layers are basically formed out of eight core application layers and a patch layer for each core layer.

These layers can be looked at as an onion with multiple layers; the lowest layer or core application layer being the SYS layer and the outermost layer being the user layer USR.

So when any application element is being executed, the system will look at the outermost code layer first, to see if there is any code for that element. If there isn't, it peels a layer off the onion and tries the next layer. When it hits a layer where the element exists, it will use the code from this layer and will not continue to peel off layers to find code for that element in the innermost layers. The 16 layers of Dynamics AX 2012 are shown here:
The following list is taken from the Dynamics AX 2012 MSDN site and provides an explanation for each layer:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USR</td>
<td>The user layer is for user modifications, such as reports.</td>
</tr>
<tr>
<td>CUS</td>
<td>The customer layer is for modifications that are specific to a company.</td>
</tr>
<tr>
<td>VAR</td>
<td>Value Added Resellers (VAR) can make modifications or new developments to the VAR layer as specified by the customers, or as a strategy of creating an industry-specific solution.</td>
</tr>
<tr>
<td>ISV</td>
<td>When an Independent Software Vendor (ISV) creates its own solution, the resultant modifications are saved in the ISV layer.</td>
</tr>
<tr>
<td>SLN</td>
<td>The solution layer is used by distributors to implement vertical partner solutions.</td>
</tr>
<tr>
<td>FPK</td>
<td>The FPK layer is an application object patch layer reserved by Microsoft for future patching or other updates.</td>
</tr>
<tr>
<td>GLS</td>
<td>When the application is modified to match country- or region-specific legal demands, the modifications are saved in the GLS layer.</td>
</tr>
<tr>
<td>SYS</td>
<td>The standard application is implemented at the lowest level, the SYS layer. The application objects in the standard application can never be deleted.</td>
</tr>
</tbody>
</table>

**Network tiers**

AX 2012 is built as a three-tier solution. This means that it has a data tier, in which the data is stored, a logic tier, where all business logic is executed, and a presentation layer, which presents information to the user and receives input from the user.

As an AX developer, you can choose to have the piece of code you are writing to run on the client or the server. The client will represent the presentation tier and the server will represent the logic tier. The AX server is also called the Application Object Server (AOS). The data tier is hosted on MS SQL Server.

There are also two completely different clients available in AX. The regular Windows client, also known as the rich client, is the one most users think of when we talk about the AX client; but there is also a business connector client. The business connector client can be used by a third-party application to integrate with AX, and it is also used by the enterprise portal.

In addition to these three layers, we also often talk about a fourth component in the tier model, which is the file server. The file server is not really a tier itself, as it does not execute any code but only stores the code.
Case study – Carz Inc.

Most of the examples in this book are related to a case study where you have been assigned with the task of developing a module in AX for Carz Inc., a car rental company.

Carz Inc. rents out cars and sells them after they have reached a certain mileage.

The company needs to store information about their cars and their customers, and they need a booking system within Dynamics AX that enables them to see which cars are available/rented out within different car groups.

The customers should also be able to see availability from an enterprise portal site.

The examples will show you step-by-step how to create extended data types, tables, forms, reports, and everything you need to make the whole module work according to the best practice rules and according to the requirements set by Carz Inc.
Summary

Now that you have made it through the first chapter of this book, you have hopefully gained a good foundation to learn more on how to program in AX.

This chapter has shown you how to write the famous Hello World program in AX. It has taken you through the development environment, where you learned about the different elements in the AOT.

You have seen some of the most important development tools that you will get to know very well once you get going with the AX development.

The application object layers and network tiers have shown you the main concepts behind the technical architecture of AX and you have read a little bit about the case study that we will continue to use throughout the book.

In the next chapter, you will learn the syntax of the X++ language by looking at the different data types you can use in X++, how to write statements and loops (such as if/else and while loops), how different operators are used, and how to create and use classes and methods.
Where to buy this book

You can buy Learning MS Dynamics AX 2012 Programming from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.