.NET Fundamentals for Magic xpa Developers

Chapter

1

What Is .NET?

We have all heard about .NET. It has been around for a number of years already. You probably have applications on your computer running on .NET. So what is it?

Microsoft .NET is pronounced as “dot net” and is sometimes written as .NET and sometimes as DotNET. This is a component that runs on the Windows operating system. .NET provides tools and libraries that enable developers to create Windows software much faster and easier. In order to run a .NET application you need to have the .NET Framework installed on the PC.

But what does Microsoft say?

Microsoft explains .NET like this: “Microsoft .NET is the Microsoft Web services strategy to connect information, people, systems, and devices through software. Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services. With .NET-connected solutions, businesses can integrate their systems faster, and sooner realize the promise of getting information anytime, anywhere, on any device.” When talking about .NET, Web services are not the first thing that comes to mind.

However, Microsoft goes on to explain that “The .NET Framework is Microsoft's platform for building applications that have visually stunning user experiences, seamless and secure communication, and the ability to model a range of business processes. By providing you with a comprehensive and consistent programming model and a common set of APIs, the .NET Framework helps you to build applications that work the way you want, in the programming language you prefer, across software, services, and devices.” Developing applications, stunning user experience etc, this is what we want.

As Magic xpa programmers we would like to be able to take advantage of all the stunning options that the .NET framework has to offer us.

But before we can embark on this mission, we have to take a step backwards and understand some basic .NET concepts. An understanding of these basic concepts will help you take full advantage of all that Magic xpa and .NET have to offer.

Chapter

2

The .NET Architecture

In the words of a classic song, “Let’s start at the very beginning, a very good place to start”, we will start with the basics, the .NET architecture.

The .NET architecture consists of various levels, which we will now discuss.

Common Language Runtime

The Common Language Runtime or as it is known, the CLR, is the most vital component of the .NET architecture. This is the .NET runtime and is therefore responsible for executing .NET code. The primary role of the CLR is to locate, load, and manage .NET types on your behalf. The CLR converts the .NET compiled code into platform native code.

The CLR allows .NET programmers to ignore the operating system’s low level operations and simply concentrate on the task at hand. The CLR also provides services such as memory management, thread management, exception handling, garbage collection and security, which programmers can use in their code. Does this seem familiar to you?

.NET programming languages

In order to develop .NET applications, programmers need to select the language they intend to code in. There are many .NET languages, including C# (pronounced C Sharp), VB .NET, J# and others.

Common Language Specification

The Common Type System, CTS, is the specification that describes all the .NET types and programming rules supported by the CLR. The CTS was created by Microsoft, and the Microsoft .NET framework is their implementation of their CTS standard.

A .NET language might not support every feature that is defined by the CTS specification. That is where the *Common Language Specification*, *CLS*, comes to our aid. The CLS is a specification that defines a subset of the CTS specification’s common types and programming constructs that all .NET programming languages can agree on.


The Magic xpa engine adheres to the .NET CLS. This means that Magic xpa builds .NET types that expose CLS-compliant features.

Managed Code

The .NET environment enables you to develop .NET code using many programming languages, such as VB .NET and C#. When you compile this code, the compiler creates compiled code. This compiled code is based on a standard called the *Common Language Infrastructure* (CLI), which defines the execution environment for program code. This compiled code is also known as *Managed Code* or *Microsoft Intermediate Language (MLI)*. Any programming language that can be compiled to Managed Code can also

be referred to as a .NET-aware language.

When you execute a compiled .NET program, the CLR has a built-in *just-in-time compiler*, which takes the Intermediate Language and converts the code into platform native code and then executes it. The just-in-time technique is a well known method for improving performance at runtime.

Base Class Libraries

The *Base Class Libraries* (BCL) is a part of the .NET platform and is therefore available to all .NET-aware languages. The BCL contains libraries and services for use by these languages, such as types that facilitate string manipulations, I/O access, collections, including arrays and lists, database access, manipulation of XML documents and many others.

Assemblies

As discussed earlier, the CLR takes partially compiled code and at runtime uses its internal just-in-time compiler to compile the code and then compile it into machine language. This partially compiled code is called an **assembly** and contains *Common Language Infrastructure (CLI)* code, which is byte code that is not compiled to platform-specific instructions.

An assembly can consist of one or more files. If an assembly is composed of a single *.dll or *.exe module you have a *single-file assembly*. These assemblies contain all the necessary CIL, metadata, and associated manifest in a single package. Multiple file assemblies comprise numerous .NET binaries or *modules*. When building a multiple file assembly, one of the modules contains the assembly manifest together with CIL instructions and metadata for various types.

 Magic xpa provides the ability to write .NET code within a program. This can be done via the Invoke operation. Figure 1 shows the use of .NET code inside Magic xpa. When creating the Magic xpa cabinet file (ECF), the Magic xpa Studio compiles the code by invoking the .NET complier, which is one of the services provided by the .NET framework. In essence, the cabinet file now contains .NET assemblies.

During runtime, the Magic xpa deployment engine on the client machine invokes the CLR to execute the code.

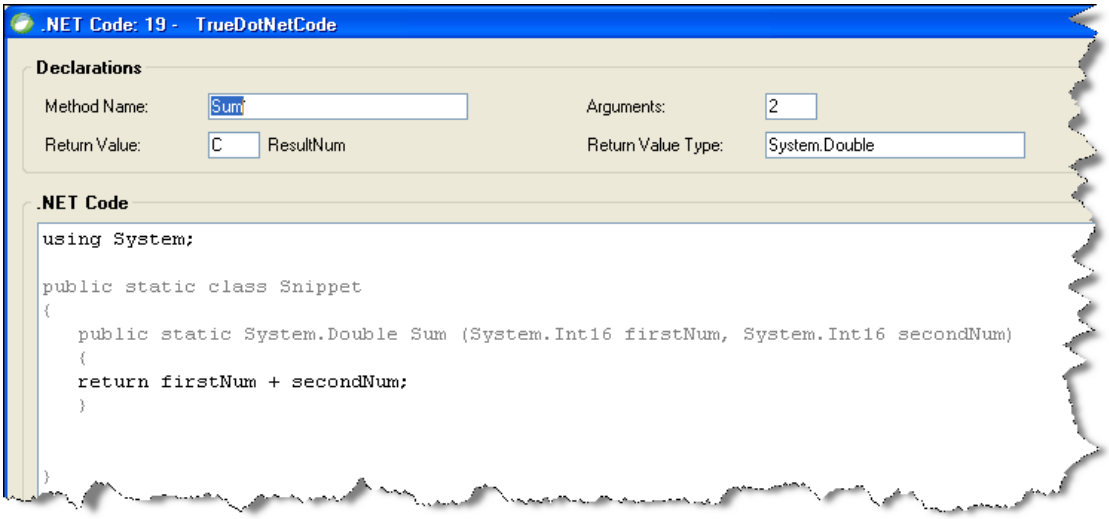

Figure 1 .NET code created in Magic xpa

Garbage Collection

A garbage collector is a feature of the .NET framework that attempts to reclaim memory used by objects that are no longer in use by the application.

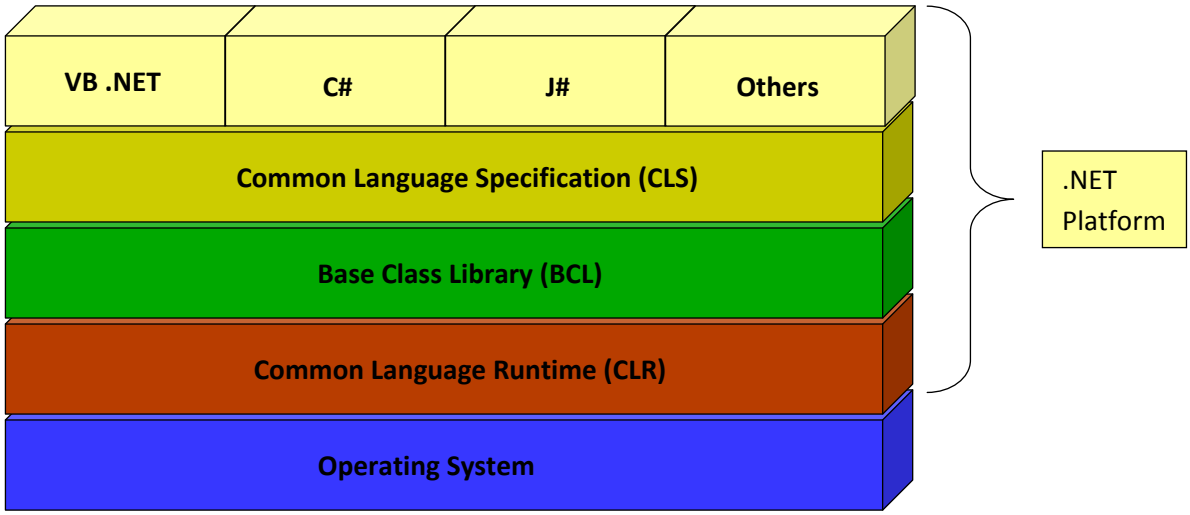

Figure 2 .NET Platform Architecture

Chapter 3

.NET Language Components

Now that you have an understanding of the .NET framework, you need to have a basic understanding of the terminology used in .NET languages. This will help you to easily implement .NET code in Magic xpa.

Namespaces

Namespaces are a way of grouping type names, which reduces the possibility of having duplicate names. It's a method of putting types inside a container so that they can be clearly distinguished from other types with the same name. A namespace can contain other namespaces and types.

For example, the *System* namespace contains the *Windows* namespace, which in turn contains the *Forms* namespace. In .NET this can be written in shorthand as System.Windows.Forms.

Classes

A class is simply a definition or a template containing properties, methods, and events. The class is one of the defining ideas of object-oriented programming. A class can have subclasses that inherit all or some of the characteristics of the class. A subclass can also define its own methods and variables that are not part of their parent class. The structure of a class and its subclasses is referred to as the class hierarchy. An example of a .NET class is the Button class. Accessing this class in .NET shorthand would be System.Windows.Forms.Button.

Objects and Instances

When we use a class we are actually creating an instance of the class. This instance is called an object. This object inherits all the properties, methods and events defined by the class. You can define multiple objects of the same class. Every time you instantiate a new object from a class, you get a new instance of that class.

In Magic xpa we can define a .NET variable. When we define the variable and select the .NET Object Type, we are actually selecting the .NET class. Figure 3 shows an example of selecting the *MonthCalendar* class from the *System.Windows.Forms* namespace. If the .NET variable is used as a control on the form, Magic xpa will automatically create an instance of this class (instantiate) in runtime. If the .NET variable is not used on the form, it will not be automatically instantiated.

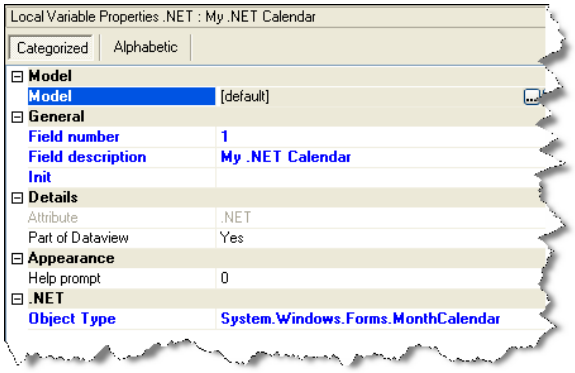

Figure 3 Defining a .NET variable in Magic xpa

Methods

A method is a procedure that is defined as part of a class. Methods define the actions that an object of a class can perform. They are generally used to perform a processing task on an object. A class can have more than one method. These methods are included in any object created for that class. They can receive parameters and can return a value.

You can think of a Magic xpa program that receives parameters as a kind of method. A Magic xpa function is also a kind of method.

Void

The **void type** is the result of a function or procedure that does not return a value. This is not new to .NET; we have seen the void return type when using Windows functions. The function receives arguments and can update the values of the arguments passed to it, but it has no return value.

Method overloading

Method overloading is the ability to define several methods all with the same name but with different parameters and return values.

Constructor

A constructor is a method that creates an instance of a class. It is responsible for initializing an object. It is different from other methods in that it is defined without a return type, not even void. Constructors have the same name as the class that defines it.

A constructor, like other methods, can receive parameters or not. A constructor that does not receive parameters is sometimes referred to as the default constructor.

As mentioned earlier, if a .NET variable is not used on the form, it will not be automatically instantiated. The program will need to instantiate the object. Figure 4 shows an example of an expression instantiating a variable using a constructor.

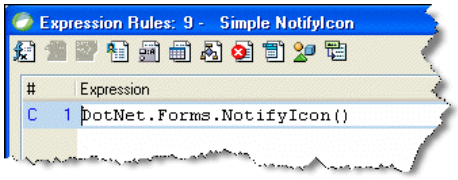

Figure 4 Magic xpa example showing a constructor

Signature

The method signature defines the inputs and outputs for the method. It specifies the method's return type and the types of its parameters. *Int32* is an example of a return type.

Events

As in Magic xpa, an event is invoked when something happens to the object. The program may decide to handle that event or not.

By Value or by Reference

As in Magic xpa, when we create a program we often need to pass information. This information can be passed to the called program through parameters.

In .NET, we can pass parameters by value or by reference. In .NET, the procedure defines whether it receives an argument by reference or by value. When we define an argument to be passed by reference we are defining that any changes to that argument within the method will be reflected in the variable in the calling method.

Most of the objects, such as *String*, are reference types and automatically passed by reference. Other objects, such as *Int32*, are passed by value.

Properties

In Magic xpa, every control has a list of properties that define that property. During runtime we can define a value for some of those properties if we set their value as an expression. However, we are unable to retrieve the current value of that property.

In .NET, properties are not only defined on controls but as class members. These are *named members* of the class. We are able to both retrieve and change the value of a property. Figure 5 shows an example of using the Magic xpa DNSet function to update the ShowWeekNumbers property of the MonthCalendar class.

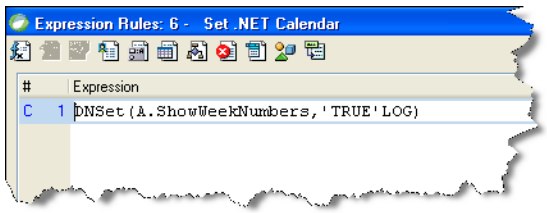

Figure 5 Magic xpa example using the DNSet function to update a property

Fields

A .NET field is a variable defined in a class. When a new instance of a class is created these fields are initialized. It is usually the constructor that initializes the values of these fields.

Access Modifier

When declaring a class or a member of a class, you can define their accessibility level. This controls whether they can be accessed only from within your assembly or also from other assemblies.

- ***public*** means that the member can be accessed by everyone.
- ***private*** means that the member can only be accessed from within that class.
- ***protected*** means that the member can only be accessed from within that class or in any derived class.
- ***internal*** means that the member can only be accessed within the same assembly and not from another assembly.
- ***protected internal*** means that the member can be accessed within the same assembly or a derived class of a different assembly.

Static

Static members are class members that can be accessed without creating an instance of the class. For example, the String.Compare() method is a static method of the String class that can be invoked on the class itself instead of on an instance of the class. Static field members have the same value for all instances of the class.
A static class can only contain static members, cannot be instantiated, cannot be inherited and cannot contain a constructor.

Enum or Enumeration

Enumerations are a list of named constants whose actual values are numbers. As an example, we can take the Windows function MessageBox. When the dialog box is displayed, we can control the number of buttons and the text that appears on them, such as *MB_YESNOCANCEL*, which displays three buttons. This is a constant whose actual value is 3.

In Magic xpa, we can use .NET enumerations via the DNSet function. Figure 6 shows the use of the *Blue* enumerator to set the BackColor property of the MonthCalendar class.

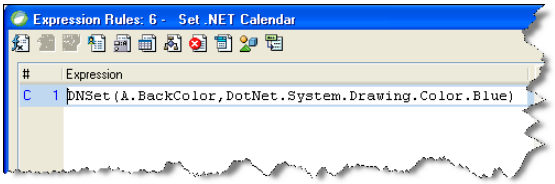

Figure 6 Magic xpa example showing the use of an enumerator

Chapter

4

Final Words

We hope this document provided you with a basic understanding of the .NET framework and how Magic xpa interacts with .NET.

Magic xpa is a .NET-aware environment that enables you to take full advantage of all that .NET has to offer, specifically what is defined in the CLS.

We are confident that the advantages of Magic xpa as well as .NET will allow you to create .NET applications in a short period of time, thereby giving you RROI – Rapid Return on Investment.

As a parting note, we advise you to read and implement the .NET Samples Guide to find out how to use .NET in Magic xpa and to get a full understanding about this subject.