

# C# 3.0 Programming in the .NET Framework

## Course OutlineModule 1: Introduction to the .NET Framework

This module explains how to develop applications in a variety of languages for the .NET Framework, and how various runtime mechanisms facilitate the execution of managed programs.

### Lessons

- Introduction to the .NET Framework
- Common Language Runtime Components – Garbage collector (GC), Common Type System (CTS), Just-in-Time compiler (JIT)
- An Overview of Managed Languages
- Microsoft Intermediate Language (IL)
- Native Image Generator (NGEN)
- An Overview of the Framework Class Library (FCL)
- .NET Version Evolution – from .NET 1.0 to .NET 3.5

After completing this module, students will be able to:

- Appreciate the relationships between the various components comprising the .NET platform.
- Develop applications in a variety of managed languages for the .NET Framework.
- Choose the appropriate .NET Framework version and language for the task at hand.

## Module 2: Introduction to C# 3.0

This module explains how to make the first steps in the Visual Studio Integrated Development Environment (IDE) and use the Framework Class Library (FCL) to develop simple C# applications.

### Lessons

- C# 3.0: Overview and Design Goals
- The Visual Studio Integrated Development Environment
- "Hello World" in C#
- Namespaces and References – Importing types, multi-targeting support, target platform
- Console Operations
- String Formatting
- Disassembling .NET – ILDASM, .NET Reflector

### Lab : Basic Operations

- Simple console operations
- String output formatting

After completing this module, students will be able to:

- Add references to framework types and use them in application code.

- Disassemble .NET applications to look under the hood of code and metadata.
- Use the .NET framework console operations and string formatting facilities.

### **Module 3: The .NET Type System**

This module explains how to choose and use the proper category of types – reference types or value types – for the task at hand, how to convert between different types and be wary of performance penalties introduced by boxing and unboxing.

#### **Lessons**

- The Common Type System
- The Common Language Specification
- Primitives and Built-in Types
- Value Types and Reference Types
- Boxing and Unboxing
- System.Object Class Members
- Type Conversions

#### **Lab : Reviewing Reference Types and Value Types**

- Class exercise – comparing operations on value types and reference types

#### **Lab : Reviewing Object Equality**

- Class exercise – comparing equality operations on value types and reference types

After completing this module, students will be able to:

- Use value types and reference types properly and in the right context.
- Understand the implications of boxing and unboxing and how to prevent them.
- Convert values between different types and representations.
- Develop managed applications with a public surface that can be consumed from other (CLS compliant) managed languages.

### **Module 4: C# Classes**

This module explains how to design and implement C# classes with a variety of member types, and accommodate for the coding guidelines of .NET Framework types.

#### **Lessons**

- Class Members
- Access Modifiers
- Nested Types
- Fields
- Constructors and Static Constructors
- Constants and Readonly Fields

- Properties and Automatic Properties
- Object Initializer Syntax
- Methods and Static Methods
- Static Classes
- Extension Methods
- Partial Types and Partial Methods
- The new Operator
- Parameter Modifiers
- Variable Parameter Lists
- The Entry Point and its Parameters
- Destructors

### **Lab : Basic Class**

- Rectangle class – methods, static methods, fields, properties
- Linked list, partial methods, and extension methods

After completing this module, students will be able to:

- Declare classes with fields, properties, methods, constructors and destructors.
- Define constants and read-only fields.
- Implement static classes with static constructors, and static methods.
- Pass parameters to methods by value, by reference and as output parameters.
- Use variable-length parameter lists.
- Create object instances with the object initializer syntax.
- Apply extension methods to extend existing types without friction.

### **Module 5: Garbage Collection**

This module explains how to interact with the .NET garbage collector (a service that automatically reclaims unused memory), and how to use finalization to execute cleanup code for unmanaged resources.

#### **Lessons**

- Destructor and Finalization
- Tracing Garbage Collection
- Interacting with the Garbage Collector
- Generations
- Weak References

After completing this module, students will be able to:

- Declare finalizers (destructors) to execute cleanup code.
- Interact with the garbage collector to improve application performance.
- Use weak references to manually manage object lifetime.

## Module 6: XML Documentation

This module explains how to document code while developing it and how to generate professional-looking external documentation from XML comments.

### Lessons

- XML Overview
- XML Documentation in Comments
- Auxiliary Tools – Sandcastle, DocumentX!

After completing this module, students will be able to:

- Document code while developing it using XML Documentation.
- Generate XML Documentation files for IntelliSense use.
- Generate professional-looking external documentation from XML comments.

## Module 7: Arrays and Strings

This module explains how to declare and use arrays and strings.

### Lessons

- Array Definition and Usage – Multi-dimensional, jagged, System.Array
- Casting and Enumerating Arrays
- String Class Members
- String Immutability
- StringBuilder
- String Literals

### Lab : Name Processing

- Reading, sorting, and writing strings and files

After completing this module, students will be able to:

- Declare and use arrays.
- Cast arrays between different types.
- Enumerate arrays using the *foreach* statement.
- Declare and use immutable strings.
- Expand and format strings multiple times using StringBuilder.
- Use string literals and verbatim string literals.

**Module 8: Object Oriented Programming in C#** This module explains how to use inheritance and polymorphism in C# classes, including up- and down-casts. **Lessons**

- Inheritance and Polymorphism
- Up Casts and Down Casts

- Inheritance and Overriding Subtleties

### **Lab : Shapes**

- Shape inheritance hierarchy
- Extending the hierarchy – a compound shape (Composite design pattern)

After completing this module, students will be able to:

- Design and implement class hierarchies with virtual, abstract and sealed methods.
- Cast up and down within an inheritance hierarchy.
- Avoid the pitfalls of inheritance, virtual dispatch and method overloading.

### **Module 9: Structures and Enumerations**

This module explains how to implement user-defined value types (structures) in .NET applications with the motivation for doing so, and how to design enumeration types for convenient usage.

#### **Lessons**

- User-Defined Value Types
- Field Initialization
- Nullable Types
- Enumerations and Flags

After completing this module, students will be able to:

- Define value types (structures) with methods, fields, properties and constructors.
- Use nullable types when necessary.
- Use standard and bit-flag enumeration types.

**Module 10: Indexers** This module explains how to implement indexed class properties emulating array access syntax.**Lessons**

- Indexers
- Consuming Indexers from Other .NET Languages

### **Lab : Receptionist Scheduling**

- Indexer access to classes
- Multi-parameter indexers

After completing this module, students will be able to:

- Implement indexed class properties with a varying number of parameters.
- Ensure that indexers can be consumed from other .NET languages.

## Module 11: Exception Handling

This module explains how to design error-reporting using exceptions in managed applications, how to throw, catch and handle exceptions in a resource-oriented environment, and how to declare user-defined exceptions.

### Lessons

- Error Reporting Alternatives
- Throwing and Catching Exceptions
- Exception Types and Objects
- Inner Exceptions
- User-Defined Exceptions
- Resource Management
- Checked and Unchecked Arithmetic
- Exception Design Guidelines and Performance

### Lab : Incorporating Exception Handling

- Adding exception handling to Lab 4

After completing this module, students will be able to:

- Appreciate the advantages of exception-based error-handling.
- Throw and catch framework-defined and user-defined exceptions.
- Manage resource deallocation using *finally* blocks.
- Design and implement user-defined exception types.
- Nest exceptions and retrieve inner exception information.
- Use checked and unchecked arithmetic operations when appropriate.

## Module 12: Interfaces

This module explains how to declare interfaces, how to implement them explicitly or implicitly, and how to use system interfaces that are part of the .NET Framework.

### Lessons

- Interface Declaration and Implementation
- Explicit Interface Implementation
- System Interfaces
- Extending Interfaces using Extension Methods

### Lab : Enumeration Capabilities

- Providing enumeration via foreach to the class from Lab 7 in Module 10
- Providing find (with a comparer) capabilities to the class from Lab 4

After completing this module, students will be able to:

- Declare interfaces with properties, methods and events.
- Implement interfaces explicitly or implicitly, based on the task at hand.
- Use the appropriate system interfaces and implement them correctly.
- Extend existing interfaces using extension methods without interfering in type hierarchy or modifying third party code.

### **Module 13: Operator Overloading**

This module explains how to add user-defined operators to types, in order to provide a more convenient syntactic usage form.

#### **Lessons**

- Overloading Operators
- Operator Names in the CLS
- User-Defined Conversions – Implicit and explicit, sequence of conversions

After completing this module, students will be able to:

- Declare and implement the allowed overloaded operators for custom types.
- Read the IL translation (CLS names) of overloaded operators.
- Define explicit and implicit conversions to and from custom types.

### **Module 14: Delegates and Events**

This module explains how to declare and define delegates as multi-function pointers, how delegates are implemented, how to use anonymous methods (closures) for improving programming productivity, and how to use events to implement common design patterns.

#### **Lessons**

- Delegate Definition and Usage
- Delegate Implementation
- Multi-cast Delegates
- Anonymous Methods
- Lambda Functions
- Events
- Event Design Patterns

#### **Lab : Sorting with Delegates**

- Sort criteria implementation using delegates

#### **Lab : Event-Based Chat System**

- Client and server event-based chat

After completing this module, students will be able to:

- Declare new delegate types and use pre-declared delegate types included in the .NET Framework.
- Use multi-cast delegates to hold a reference to multiple methods.
- Apply anonymous methods and lambda functions instead of defining real methods for each delegate implementation.
- Design and implement events in custom types, and consume events defined by types in the .NET Framework.

**Module 15: Preprocessor Directives** This module explains how to use preprocessor directives to conditionally compile code into C# applications. **Lessons**

- Preprocessing Directives
- Defining and Undefined Preprocessor Directives

After completing this module, students will be able to:

- Define and undefine preprocessor directives.
- Use preprocessor directives to control conditional compilation of code.

**Module 16: Improved C++** This module explains how to avoid common pitfalls when transitioning from C++ to C#. **Lessons**

- Control Flow Statements
- Switch Blocks

After completing this module, students will be able to:

- Use control flow statements with Boolean types appropriately.
- Use switch...case statements with a variety of types supported by C#.

### **Module 17: Metadata and Reflection**

This module explains how to use Reflection to obtain run-time information about types, methods, properties and fields, and how to create object instances and interact with them at run-time without requiring early-binding during compilation.

#### **Lessons**

- Metadata Tables
- Reflection Types
- System.Activator

#### **Lab : Self-Registration with Interfaces**

- Self-registered singleton repository using a marker interface

After completing this module, students will be able to:

- Interrogate assemblies, types, methods, properties and fields at run-time without prior knowledge of these types.
- Create object instances and interact with them at run-time.
- Invoke methods, obtain property values and field values without early-binding to types during compilation.

### **Module 18: Attributes**

This module explains how to decorate code elements with framework-defined and custom attributes, how to design and implement custom attribute types, how to query attributes using Reflection and how to design aspect-oriented applications using attributes.

#### **Lessons**

- Attribute Class
- Attribute Examples
- Applying Attributes
- User-Defined Attributes and Attribute Usage
- Querying Attributes with Reflection

#### **Lab : Logging with Attributes**

- Primitive object serialization for logging purposes

#### **Lab : Self-Registration with Attributes**

- Self-registration (see Lab 12) with attributes instead of a marker interface

After completing this module, students will be able to:

- Apply attributes to code elements, including assemblies, types, type members and method parameters.
- Design and implement custom attribute types.
- Query attributes using Reflection at run-time and make run-time decisions based on attribute information.
- Design aspect-oriented applications using attributes and run-time interrogation.

### **Module 19: Generics**

This module explains how to design and implement generic types and methods for the widest range of data types, how to use constraints to limit the application of generic code, how to interrogate generic types at run-time using Reflection and how .NET generics compare to C++ templates.

#### **Lessons**

- Motivation for Generics
- Generic Constraints
- Generic Interfaces, Methods, and Delegates

- .NET Generics vs. C++ Templates
- Generics and Reflection

After completing this module, students will be able to:

- Develop generic code including system types, interfaces, methods and delegates.
- Use constraints to limit the data types that can be used with custom generic code.
- Investigate generic types at run-time using Reflection and create generic type instances at run-time.
- Compare .NET generics to C++ templates and appreciate the advantages and limitations of each implementation approach.

### **Module 20: Generic Collections**

This module explains how to use the system generic collections to obtain better performance with value types and reference types, and how to use generic system interfaces.

#### **Lessons**

- Built-in Generic Collections
- Generic System Interfaces
- Collection Initializers

#### **Lab : Implementing a Generic Collection**

- Implementing IList on the collection from Lab 4

After completing this module, students will be able to:

- Choose the appropriate generic collection implemented in the .NET Framework 2.0.
- Appreciate the differences between generic and non-generic collections.
- Use generic system interfaces when appropriate, instead of their non-generic counterparts.

### **Module 21: Deployment, Versioning, and Configuration**

This module explains how to deploy, version, configure and register .NET assemblies in a private or shared configuration scenario, how to control versioning and binding policy through application configuration files, and how to create multi-module (and even multi-language) assemblies.

#### **Lessons**

- Deployment and Versioning of .NET Assemblies
- Private and Shared Assemblies – The Global Assembly Cache (GAC)
- Application Configuration Files
- Versioning Policies
- Friend Assemblies
- Multi-Module Assemblies

#### **Lab : Creating and Registering Assemblies**

- Creating a privately deployed assembly
- Using probing configuration to access an assembly at a sub-directory
- Registering a shared assembly in the GAC
- Controlling versioning (binding) policy using application configuration

After completing this module, students will be able to:

- Apply versions to .NET assemblies.
- Deploy .NET assemblies as private assemblies or as shared assemblies in the Global Assembly Cache (GAC).
- Configure applications using application configuration files.
- Control versioning policies and binding policies through configuration files.
- Manage friend assemblies.
- Create multi-module (multi-language) assemblies.

### **Module 22: Unsafe Code and Interoperability**

This module explains how to use the .NET interoperability features to integrate managed and unmanaged code within the same application, and how to use unsafe code (C# pointers) to obtain performance and interoperability benefits.

#### **Lessons**

- .NET Interoperability Options
- Introduction to Platform Invoke (P/Invoke)
- Unsafe Code – C# Pointers

#### **Lab : Calling Exported C Functions from C#**

- Calling a custom exported C function from C
- Calling a Win32 API (requiring a reverse P/Invoke callback)

After completing this module, students will be able to:

- Appreciate the various .NET interoperability options.
- Use Platform Invoke (P/Invoke) to call unmanaged code from C#.
- Use unsafe code (C# pointers) in high-performance scenarios.

### **Module 23: Introduction to Language-Integrated Query (LINQ)**

This module explains how to use Language-Integrated Query (LINQ) constructs and associated language features to increase productivity and model declarative data-driven applications instead of implementing them in the standard imperative manner.

#### **Lessons**

- Anonymous Types and Implicit Variables
- Expression Trees

- Query Operators and the Query Pattern
- Language-Integrated Query Keywords and Query Translation
- LINQ to Objects

### **Lab : Using LINQ**

- Implementing extension methods
- Implementing custom query operators
- Implementing the query pattern
- Writing declarative LINQ queries against object models

After completing this module, students will be able to:

- Use Language-Integrated Query (LINQ) to design and implement declarative data-driven applications.
- Apply C# 3.0 anonymous types and implicit variable typing with and without LINQ.
- Manipulate lambda expressions through expression trees to bridge data and code