

## Programming C# Extended Features: Hands-On - 4 Days Streamlining Entity Framework Applications

*Course 973 Overview*

- You Will Learn How To**
- Streamline data-centric applications with C# extended features and the Entity Framework (EF)
  - Apply lambda expressions and extension methods for middle-tier processing
  - Exploit Language Integrated Query (LINQ) keywords to filter and order data
  - Transfer complex logic with generic **Func<T,R>** delegates
  - Generate dynamic enumerations using **IEnumerable<T>** within developer-written generic classes
  - Leverage LINQ to simplify XML processing
- Course Benefits** C# has evolved since its introduction into a full data-manipulation language. .NET extended features enable programmers to streamline data access, increase productivity and improve overall performance of applications. In this course, you apply C# extended features, including LINQ and EF, to effectively integrate the object-oriented and data-manipulation capabilities.
- Who Should Attend** Experienced C# programmers who want to improve their software development capabilities by using extended language features, and in particular Language Integrated Query (LINQ) in conjunction with the Entity Framework. C# programming experience at the level of Course 419, "C# Programming," is assumed.
- Hands-On Training** Exercises using a progressive case study provide experience applying C# extended features and include:
- Applying auto-properties and object initializers
  - Writing lambda expressions and extension methods
  - Implementing the data tier with LINQ and EF
  - Employing DataContext objects and LINQ query keywords to access and update a database
  - Mapping entity classes using the O/R Designer
  - Achieving dynamic enumerations with **yield**
  - Parsing XML documents using LINQ
  - Accessing stored procedures with C#/LINQ

## Programming C# Extended Features: Hands-On - 4 Days

### Streamlining Entity Framework Applications

*Course 973 Outline*

#### Introduction

- Review of object-oriented programming
- Implementing to an interface
- Existing data-access technologies
- Generic and nongeneric collections

#### Harnessing C# Language Features

##### Language shortcuts

- Applying auto-implemented properties
- Benefiting from implied-type declaration

##### Streamlining program logic

- Instantiating entity objects
- Simplifying construction with object initializers
- Anonymous object construction

#### Employing Extension Methods for Middle-Tier Data Manipulation

##### Filtering and ordering data with lambda expressions

- Constructing lambda expressions
- Comparing delegates and lambda expressions
- Calling **Count**, **Reverse**, **Union**, **Except** and other extension methods
- Passing types and functionality into methods

##### Applying extended features

- Making code more flexible with delegates
- Parameterizing delegates and lambda expressions
- Streamlining code with **Func<T,R>** generic delegates

#### Leveraging LINQ Query Keywords

##### Syntax and semantics

- Coding LINQ queries
- Ordering data and objects
- Filtering using **from**, **where**, **orderby** and **group**

##### Iterating collections

- Interfacing between C# and LINQ using **IEnumerable<T>**
- Converting from **IEnumerable<T>** to **List<T>**

#### Adapting LINQ and EF

##### Comparing ADO.NET with LINQ

- **DataSet** objects vs. generic lists

- **SqlDataReader** vs. **IEnumerable<T>**
- Processing data-tier information

#### Connecting to and reading from a database

- Establishing an **ObjectContext**
- Attaching to databases via the Entity Framework (EF)
- Correlating entity classes and data tables
- Preserving the object-oriented paradigm

#### The Object Relational Designer tool

- Mapping data tables to entity classes
- Establishing inheritance relationships
- Importing stored procedures

#### Updating a database

- Inserting, updating and deleting data
- Error handling and exceptions
- Committing and rolling back transactions

#### Enumerations and Generic Classes

##### Examining enum

- Comparing **IEnumerable<T>** and **IEnumerator<T>**
- Generating dynamic enumeration
- The **yield** keyword

##### Writing generic classes

- Reducing duplicate classes
- Implementing a generic fast enumerator

#### Processing Data with LINQ Queries

##### Manipulating data

- Combining and aggregating similar data with **group**
- Performing inner, outer and group joins
- Generating data subsets with the **into** clause

##### Advanced LINQ techniques

- Creating anonymous query results
- Retrieving composite views using nested **from** clauses
- Enhancing LINQ queries using delegates and lambda expressions

#### Applying LINQ to XML

##### Exploiting the XML Namespace

- Loading XML dynamically via the Web
- Creating and saving XElement content

#### Processing XML

- Retrieving the document, elements and attributes
- Parsing an XML document using LINQ