Progress®



OpenEdge® Development:
GUI for .NET Mapping Reference

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....7
Purpose....7
Audience....8
Organization....8
Using ABL documentation....9
References to ABL compiler and run-time features....9
References to ABL data types....9
Typographical conventions....9

Chapter 1: Mapping Reference Entries....11
C# .NET code mapped to OpenEdge GUI for .NET ABL....12
OpenEdge GUI ABL widgets mapped to similar .NET forms and controls....23
OpenEdge GUI ABL window with a browse compared to an OpenEdge GUI for .NET form with a grid.25
OpenEdge GUI dialog compared to OpenEdge GUI for .NET dialog....27
OpenEdge GUI behavior compared to OpenEdge GUI for .NET behavior....28
OpenEdge GUI events compared to OpenEdge GUI for .NET events....31
Common public .NET form and control methods....34
Common public .NET form properties....36
Common public .NET control properties....37
Common public .NET form events....40
Common public .NET control events....40
Implicit data type mapping between ABL primitive and .NET types....41
Explicit data type mapping from ABL primitive to .NET types....44

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions

Purpose

The *OpenEdge GUI for .NET* provides a powerful mechanism for building ABL applications that have a .NET user interface. One of the challenges in working with the GUI for .NET is to understand how things you might have done previously with ABL in the past can be done using the new GUI for .NET. This challenge has two different aspects:

- How to accomplish a traditional ABL GUI task using the OpenEdge GUI for .NET
- How to use some feature of the OpenEdge GUI for .NET that is primarily documented in Microsoft and other native .NET documentation

The first issue involves understanding how to code ABL to do something in .NET that the ABL supports as a native feature. The second issue involves understanding how to work with standard .NET features in ABL that are described in .NET documentation in terms of native .NET languages.

This manual provides a brief topical reference intended to help you with these issues. Each topic includes one or more tables that list particular features in one frame of reference and describes how to understand them in the other. So, for example, a table of data type mappings shows how .NET data types map to ABL data types. Another table lists common behaviors supported by the OpenEdge GUI and how identical or similar behaviors are supported using the OpenEdge GUI for .NET. Yet another table describes common C# .NET syntax and how to use the same features in ABL, and so on.

This manual is not intended to be exhaustive, and most of the information is covered in other ABL manuals in the OpenEdge documentation set. However, it will hopefully provide a quick reference to some of the more common questions that you might have when reading .NET documentation or when thinking about how to do something in the GUI for .NET that you have previously done a different way.

In addition to this mapping reference, you can find more information on using the GUI for .NET in the following manuals:

- *OpenEdge Getting Started: GUI for .NET Primer* — A brief overview of the OpenEdge GUI for .NET and the class-based ABL that supports it.
- *OpenEdge Development: ABL Reference* — The complete reference to the ABL, including all language elements that support the GUI for .NET.
- *OpenEdge Development: GUI for .NET Programming* — A programmer's guide for using the GUI for .NET. This manual describes how to access .NET objects using ABL to implement the GUI for .NET in your OpenEdge applications.
- *OpenEdge Deployment: Managing ABL Applications* — A guide to managing and deploying ABL applications, with sections on requirements and features for deploying GUI for .NET applications.

Audience

This manual assumes that you are an ABL application developer who wants to access .NET objects from your ABL applications, especially for building an OpenEdge GUI for .NET. You should be familiar with object-oriented programming and .NET concepts, and you should also be familiar with object-oriented programming using ABL. To make the best use of this manual, you might want to review *OpenEdge Development: GUI for .NET Programming*.

Organization

The topics in this reference manual are not in any particular alphabetical or numeric sequence, but are arranged in a rough order from the more basic to more advanced (or more general) in nature. Some, but not all, of the topic tables do list the elements they describe in alphabetical order.

The later sections provide lists of some of the more common and generally useful .NET classes and class members that you can access in an OpenEdge GUI for .NET application. Third-party documentation describes these elements in far greater detail, along with many more available classes and class members.

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all `UPPERCASE`, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever *character* appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever *decimal* appears, this is a reference to the `DECIMAL` data type.
- Wherever *numeric* appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

1

Mapping Reference Entries

The topical entries in this mapping reference provide an understanding of how to code ABL to do something using .NET objects and how to work with standard .NET features in ABL that are described in .NET documentation in terms of native .NET languages. Some entries also compare common built-in OpenEdge GUI features with OpenEdge GUI for .NET features.

For details, see the following topics:

- C# .NET code mapped to OpenEdge GUI for .NET ABL
- OpenEdge GUI ABL widgets mapped to similar .NET forms and controls
- OpenEdge GUI ABL window with a browse compared to an OpenEdge GUI for .NET form with a grid
- OpenEdge GUI dialog compared to OpenEdge GUI for .NET dialog
- OpenEdge GUI behavior compared to OpenEdge GUI for .NET behavior
- OpenEdge GUI events compared to OpenEdge GUI for .NET events
- Common public .NET form and control methods
- Common public .NET form properties
- Common public .NET control properties
- Common public .NET form events
- Common public .NET control events
- Implicit data type mapping between ABL primitive and .NET types
- Explicit data type mapping from ABL primitive to .NET types

C# .NET code mapped to OpenEdge GUI for .NET ABL

The following table compares features of C# .NET code with similar features in ABL code using the OpenEdge GUI for .NET, listed in approximate order of definition and usage.

Table 1: C# .NET code mapped to OpenEdge GUI for .NET ABL

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Object type-name syntax	**C# Syntax**
	[namespace .] dotNET-object-name
	[.inner-type-name]
	Example .NET type names:
	`System.Windows.Forms.Control`
	`System.Windows.Forms.Control.ControlCollection`
	`System.Drawing.Point[]`
		ABL Syntax
		["]
		[namespace .]
		dotNET-object-name
		[+inner-type-name]
		["]
		Example .NET type names:
		`System.Windows.Forms.Control`
		`System.Windows.Forms.Control+ControlCollection`
		`"System.Drawing.Point[]"`

Allowing unqualified type references	**C# Syntax**
	`using namespace;`
	Example:
	`// Unqualified reference to`
	`// all types in a namespace`
	`using System.Windows.Forms;`
		ABL Syntax
		`USING namespace.* [ FROM ASSEMBLY ] .`
		`USING namespace.type-name [ FROM ASSEMBLY ] .`
		Examples:
		`/* Unqualified reference to all types in a namespace */`
		`USING System.Windows.Forms.* FROM ASSEMBLY.`
		`/* Unqualified reference to the Button class type */`
		`USING System.Windows.Forms.Button FROM ASSEMBLY.`

Class instantiation	**C# example:**
	`using System.Windows.Forms;`
	`Button rButton = new Button( );`
		ABL example:
		`USING System.Windows.Forms.* FROM ASSEMBLY.`
		`DEFINE VARIABLE rButton AS Button.`
		`rButton = NEW Button( ).`

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Type casting[1]	**C# example:**
	`using System.Windows.Forms;`
	`Control rControl = new Control( );`
	`Button rButton = new Button( );`
	`rControl = rButton;`
	`rButton = (Button) rControl;`
		ABL example:
		`USING System.Windows.Forms.*.`
		`DEFINE VARIABLE rControl AS Control.`
		`DEFINE VARIABLE rButton  AS Button.`
		`rControl = NEW Control( ).`
		`rButton = NEW Button( ).`
		`rControl = rButton.`
		`rButton = CAST ( rControl, Button ).`

Instance member reference	**C# Syntax**
	object-reference.***member-reference***
	Example:
	`using System.Windows.Forms;`
	`Button rButton = new Button( );`
	`rButton.`**`Left`**` = 10; /* Property ref. */`
	`rButton.`**`Focus( );`**`  /* Method call */`
		ABL Syntax
		object-reference:***member-reference***
		Example:
		`USING System.Windows.Forms.*`
		`  FROM ASSEMBLY.`
		`DEFINE VARIABLE rButton AS Button.`
		`rButton = NEW Button( ).`
		`rButton:`**`Left`**` = 10. /* Property ref. */`
		`rButton:`**`Focus( ).`**`  /* Method call */`

Static member reference	**C# Syntax**
	object-type-name.***member-reference***
	Example:
	`using System.Windows.Forms;`
	`MessageBox.`**`Show`**`(`
	`  System.Math.`*`PI.ToString`*`( ) );`
		ABL Syntax
		object-type-name:***member-reference***
		Example:
		`USING System.Windows.Forms.*`
		`  FROM ASSEMBLY.`
		`MessageBox:`**`Show`**`(`
		`  STRING(System.Math:`**`PI`**`) ).`

[1] Unlike for C# and other .NET languages, ABL only supports type casting between object types. ABL supports type conversion (without the need for type casting) between compatible ABL primitive or array types and corresponding .NET types. For more information, see the Type conversion and mapping feature in this table.

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Creating and using .NET array objects	**C# example:**
	`using System.Windows.Forms;`
	
	`Button[] btnArrayObj`
	`  = new Button[ 3 ];`
	`btnArrayObj[0] = new Button( );`
		ABL examples:
		`USING System.Windows.Forms.*`
		`  FROM ASSEMBLY.`
		`USING Progress.Util.*`
		`  FROM ASSEMBLY.`
		
		`/*Creating and using a`
		`  .NET array object directly */`
		`DEFINE VARIABLE btnArrayObj`
		`  AS "System.Array".`
		`btnArrayObj`
		`  = System.Array:CreateInstance(`
		`  TypeHelper:GetType( "Button" ),`
		`  3 ).`
		`btnArrayObj:SetValue( NEW Button( ),`
		`  0 ).`
		
		`/* Creating and using an`
		`  ABL object array to create a`
		`  .NET array object by mapping`
		`*/DEFINE VARIABLE btnArrayObj`
		`  AS "Button[]".`
		`DEFINE VARIABLE btnArray`
		`  AS Button EXTENT 3.`
		`btnArray[1] = NEW Button(`
		`).btnArrayObj = btnArray.`

Class definition[2]	**C# sample:**
	`using System.*;`
	`using System.Windows.Forms;`
	`using System.Collections;`
	`using System.Windows.Forms.Layout;`
	
	`public class`
	`  Control.ControlCollection :`
	`  ArrangedElementCollection,`
	`  IList, ICollection,`
	`  IEnumerable, ICloneable {`
	`  . . .`
	`}`
		ABL equivalent:
		`USING System.*.`
		`USING System.Windows.Forms.*.`
		`USING System.Collections.*.`
		`USING System.Windows.Forms.Layout.*.`
		
		`CLASS`
		`  Control+ControlCollection`
		`  INHERITS ArrangedElementCollection`
		`  IMPLEMENTS IList, ICollection,`
		`  IEnumerable, ICloneable :`
		`  . . .`
		`END CLASS.`

[2] Shown for comparison only, to illustrate how inheritance and interface implementation is indicated in C#. An ABL class would not, and cannot, implement all of the interfaces listed here (for example, IList).

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Method calling and overriding in class hierarchy[3]	**Example inherited C# method:**
	`protected virtualbyte[] DoOp1 (`
	`  decimal x,`
	`  refshort y,`
	`  outulong z`
	`) { . . . }`
	
	Example C# usage of inherited .NET `DoOp1( )` method:
	`// Calling DoOp1 within hierarchy`
	`byte[] arr;`
	`decimal x;`
	`short y;`
	`ulong z;`
	
	`arr = DoOp1( x, ref y, out z );`
	
	`// Overriding DoOp1`
	`public override`
	`  byte[] DoOp1 (`
	`  decimal x,`
	`  refshort y,`
	`  outulong z`
	`) { . . . }`
		Example ABL usage of inherited .NET `DoOp1( )` method:
		`/* Calling DoOp1 within hierarchy */`
		`DEFINE VARIABLE arr AS INTEGER`
		`  EXTENT NO-UNDO.`
		`DEFINE VARIABLE x AS DECIMAL NO-UNDO.`
		`DEFINE VARIABLE y AS INTEGER NO-UNDO.`
		`DEFINE VARIABLE z AS DECIMAL NO-UNDO.`
		
		`arr = DoOp1( INPUT x,`
		`  INPUT-OUTPUT y,`
		`  OUTPUT z ).`
		
		`/* Overriding DoOp1 */`
		`METHOD PUBLIC OVERRIDE`
		`"System.Byte[]" DoOp1(`
		`  INPUT x AS DECIMAL,`
		`  INPUT-OUTPUT y AS SHORT,`
		`  OUTPUT z AS UNSIGNED-INT64`
		`) : . . . END METHOD.`

[3] The ABL data type keywords, SHORT and UNSIGNED-INT64, are used in these examples to explicitly map instances of a corresponding ABL primitive type (INTEGER) to specific .NET types (C# short and ulong, respectively). For more information, see the Explicit data type mapping from ABL primitive to .NET types on page 44.

<table>
<tr><th>Feature</th><th>Microsoft .NET C#</th><th>OpenEdge GUI for .NET ABL</th></tr>
<tr><td>Event declaration and subscription</td><td>C# event declaration and delegate for the .NET FormClosing event:


```
using System.Windows.Forms;

// Event declaration
public event FormClosingEventHandler
  FormClosing;

// Event delegate
public delegate void
FormClosingEventHandler (
  System.Object sender,
  FormClosingEventArgs e
)
```

C# example event subscription and handler:

```
using System.Windows.Forms;


Form rFrm = new Form( );


. . .
// Subscribe handler to event
rFrm.FormClosing += new
Form.FormClosingEventHandler(
    rFrm_FormClosing);
. . .
private void rFrm_FormClosing(
  System.Object sender,
  FormClosingEventArgs e
) { . . . }
```

</td><td>ABL example event subscription and handler:


```
USING System.Windows.Forms.*
  FROM ASSEMBLY.

DEFINE VARIABLE rFrm
  AS Progress.Windows.Form NO-UNDO.
rFrm = NEW Form( ).
. . .
/* Subscribe handler to event */
rFrm:FormClosing:Subscribe(
   rFrm_FormClosing ).
. . .
METHOD PRIVATE VOID rFrm_FormClosing(
  sender AS System.Object,
  e      AS FormClosingEventArgs
) : . . . END METHOD.
```

</td></tr>
<tr><td>Publishing an event in a derived class</td><td>.NET method that fires a FormClosing event:


```
protected virtual void OnFormClosing
(
  FormClosingEventArgs e
)
```

Example of publishing the event in a C#-derived .NET class:

```
using System.Windows.Forms;

FormClosingEventArgs rClosingArgs
  = new FormClosingEventArgs(
          CloseReason:UserClosing,
          FALSE
        );

. . .

/* Publish FormClosing event */
OnFormClosing( rClosingArgs );
```

</td><td>Example of publishing the event in an ABL-derived .NET class:


```
USING System.Windows.Forms.*
  FROM ASSEMBLY.

DEFINE VARIABLE rClosingArgs
  AS FormClosingEventArgs NO-UNDO.

rClosingArgs
  = NEW FormClosingEventArgs(
          CloseReason:UserClosing,
          FALSE
        ).
. . .
/* Publish FormClosing event */
OnFormClosing( rClosingArgs ).
```

</td></tr>
</table>

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Enumeration types and operations	In .NET, you can operate on enumeration types using the same operators defined for integral primitive types. For example, an operator such as + is overloaded for both the C# int and enumeration types.
	
	C# use of an enumeration type:
	
	`using System.Windows.Forms;`
	
	`Button rBtn = new Button( );`
	
	`rBtn:Anchor = AnchorStyles.Bottom & AnchorStyles.Left;`
		In ABL, there is no operator overloading and .NET enumerations are treated only as class-based objects. To support operations on .NET enumerations, OpenEdge provides Progress.Util.EnumHelper, a .NET helper class that defines the following static methods you can use to operate on enumeration objects as System.Enum types: Add( ), And( ), AreEqual( ), AreNotEqual( ), Complement( ), IsGreater( ), IsGreaterOrEqual( ), IsLess( ), IsLessOrEqual( ), Or( ), Subtract( ), and Xor( )
		
		ABL use of an enumeration type:
		
		`USING System.Windows.Forms.*.`
		`USING Progress.Util.*.`
		
		`DEFINE VARIABLE rBtn AS Button.`
		`rBtn = NEW Button( ).`
		
		`rBtn:Anchor = CAST( EnumHelper:AND ( AnchorStyles:Bottom, AnchorStyles:Left ), AnchorStyles ).`

<table>
<tr><th>Feature</th><th>Microsoft .NET C#</th><th>OpenEdge GUI for .NET ABL</th></tr>
<tr><td>Indexed properties</td><td>.NET has a concept both of non-default indexed properties and default properties (also referred to as indexers or indexed properties). Virtually all indexed properties in the .NET Framework are indexers defined with the default property name Item.

The following code shows a sample C# indexer declaration you might find in .NET documentation for an <code>Item</code> property, in this case on the <code>ControlCollection</code> class:

<pre>using System.Windows.Forms;

// C# indexer declaration
public virtual Control this [
  int index] { get; }</pre>Example C# indexer access on a ControlCollection object:

<pre>Control rCtrl;
ControlCollection rCol = new
  ControlCollection(. . .);
. . .

rCtrl = rCol[1];</pre></td><td>ABL documentation refers to .NET non-default indexed properties simply as indexed properties and refers to .NET default properties (or indexers) as default indexed properties. However, most indexed properties you are likely to access are default indexed properties.

Where C# considers the indexer to be the indexed property, ABL refers to an indexer as the subscript syntax used to access a default indexed property, which you can access in one of two ways. In C#, you can only access a default indexed property using an indexer on a reference to a class instance. However, ABL allows you to access a default indexed property using the indexer either on the property name (<code>Item</code>) or on a reference to the class instance.

Example ABL default indexed property access on a ControlCollection object:

<pre>USING System.Windows.Forms.*.

DEFINE VARIABLE rCtrl AS
  Control NO-UNDO.
DEFINE VARIABLE rCol AS
  Control+ControlCollection NO-UNDO.
rCol = NEW
  Control+ControlCollection(. . .).

/* Next two lines are equivalent */
rCtrl = rCol[1].
rCtrl = rCol:Item[1].</pre></td></tr>
</table>

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

Boxing and unboxing support	Boxing is the process of converting a value type (such as a C# `int` or .NET `System.Int32`) to a reference type object. Boxing a value type wraps the value inside a `System.Object`. Unboxing extracts the value type from the object. Boxing and unboxing between a value type and `System.Object` can occur during assignment or parameter passing.
	
	In C#, boxing is done automatically but can be done explicitly. Unboxing can only be done explicitly by casting from the `System.Object` to the value type you want to unbox.
	
	C# examples:
	
	`int iNetVal = 99;`
	`int[] iNetArr = int[3];`
	`System.Object rBox;`
	
	`rBox = iNetVal;`
	`iNetVal = (int) rBox;`
	`iNetVal = (System.Int32) rBox;`	

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL

		ABL provides three basic levels of automatic boxing and unboxing support:
		• Conversion between ABL primitive types (such as `INTEGER`) and a .NET `System.Object` when you assign between them
		• Conversion from an ABL array of compatible types (including ABL primitive and .NET object type elements) to a .NET `System.Object` when you pass the ABL array to a .NET method parameter defined as `System.Object` or when you assign the ABL array to a `System.Object`
		• Conversion between compatible ABL arrays and .NET array objects for both assignment and .NET method parameter passing. Arrays are boxing-compatible between: 1) ABL arrays of primitive types and .NET arrays of corresponding implicitly mapped types (see Implicit data type mapping between ABL primitive and .NET types on page 41), and 2) ABL arrays of non-mapped .NET object types and .NET arrays of type-compatible .NET object types.
		In all other situations, for example, parameter passing for ABL routines and assignment from a System.Object to an ABL primitive type, automatic boxing is not supported. Where automatic boxing is not supported, you can explicitly box and unbox between these ABL and .NET types using the ABL BOX and UNBOX built-in functions, You can also use these functions where automatic boxing is supported.
		Note: Automatic boxing support between ABL arrays of primitive types and .NET arrays of implicitly mapped types does **not** include widening support between corresponding array elements. The array elements must be implicitly mapped.
		ABL examples using iNetArr from .NET:
		`DEFINE VARIABLE iVal AS INTEGER.`
		`DEFINE VARIABLE iArr AS INTEGER`
		`  EXTENT 3 INITIAL [33,66,99].`
		`DEFINE VARIABLE rBox`
		`  AS System.Object.`
		`rBox = iVal.`
		`iVal = rBox.`
		`rBox = iArr.`
		`iArr = UNBOX(rBox).`
		`iNetArr = iArr.`

Feature	Microsoft .NET C#	OpenEdge GUI for .NET ABL
		`iArr = iNetArr.`

<table>
<tr><th>Feature</th><th>Microsoft .NET C#</th><th>OpenEdge GUI for .NET ABL</th></tr>
<tr><td rowspan="4">Type conversion and mapping</td><td>In .NET, type conversion can only occur using boxing and unboxing. For more information, see the Boxing and unboxing support in this table..NET also supports aliasing between each .NET primitive type (such as C# <code>int</code>) and a corresponding .NET value type object (in this case, <code>System.Int32</code>). This allows you to assign between the primitive type and its corresponding object type and to otherwise refer to one type as the other without casting. However, this is not a type conversion, as the two types (for example, C# <code>int</code> and <code>System.Int32</code>) are really identical. Aliasing, thus, allows all .NET languages to use the same type library.</td><td>ABL also provides a form of boxing support to convert ABL primitive and array types to .NET object types (see the Boxing and unboxing support in this table).In addition, ABL supports a form of mapping between ABL primitive or array types and compatible .NET scalar or array types. For ABL primitive types, this mapping relies on existing .NET aliasing, but supports true type conversion between the ABL and corresponding .NET types. For more information, see Implicit data type mapping between ABL primitive and .NET types on page 41.[4]</td></tr>
<tr><td>Example of type aliasing in C#:

<code>public intProperty1{get;set;}</code>
<code>public System.Int32Property2{ . . . }</code>

<code>Property1 = Property2;</code>
<code>Property2 = Property1;</code></td><td>Example of type mapping in ABL using <code>Property1</code> and <code>Property2</code> from .NET:

<code>DEFINE VARIABLE iVal AS INTEGER.</code>

<code>iVal = Property1.Property1 = iVal.</code>
<code>iVal = Property2.Property2 = iVal.</code></td></tr>
<tr><td colspan="2">Note: ABL provides built-in data type conversion functions that you can call to convert one ABL primitive type to another. For example, a call to <code>INTEGER("99")</code> converts the string, <code>"99"</code>, to the <code>INTEGER</code> value, <code>99</code>. However, note that these functions are not casting operations. The value from a data type conversion function is not the same object represented as a new type, as in object type casting, but is actually a value of one primitive type converted to an equivalent value in another primitive type.</td></tr>
<tr><td colspan="2">In addition, if you call an ABL data type conversion function on an object reference, ABL does not attempt to convert the value to the specified data type that the referenced object happens to represent. Instead, the function returns the ID of the referenced object as the primitive type specified by the conversion function:

<code>MESSAGE "Object" INTEGER(y)
  "has the value" y
  VIEW-AS ALERT-BOX.</code>

Thus, this statement might display the following message, where 1022 is the object ID of the <code>System.Object</code> referenced by <code>y</code>:

<code>"Object 1022 has the value 99"</code>

Note also that when you reference a <code>System.Object</code> directly in a <code>MESSAGE</code> statement and the <code>System.Object</code> represents a .NET mapped type (such as a <code>System.Int32</code>), the statement automatically calls the <code>ToString( )</code> method on the object (in this case, <code>y:ToString( )</code>), which returns the object's value as a string (in this case, <code>"99"</code>) instead of its object ID.</td></tr>
</table>

[4] Sometimes, you must explicitly map to a given .NET type, for example, for .NET method overloading and overriding. For more information, see the Explicit data type mapping from ABL primitive to .NET types on page 44 .

OpenEdge GUI ABL widgets mapped to similar .NET forms and controls

The following table lists each ABL widget from the OpenEdge GUI together with one or more similar .NET forms and controls available with the OpenEdge GUI for .NET, including both the Microsoft .NET GUI Controls and the OpenEdge Ultra Controls for .NET. For a short description of any control that OpenEdge installs as a design component with the Visual Designer, see Implicit data type mapping between ABL primitive and .NET types on page 41 entries.

Table 2: OpenEdge GUI ABL widgets mapped to similar .NET forms and controls

ABL Widget	Similar OpenEdge GUI for .NET forms and controls
BROWSE	System.Windows.Forms.DataGridView System.Windows.Forms.TableLayoutPanel Infragistics.Win.UltraWinGrid.UltraGrid
BUTTON	System.Windows.Forms.Button Infragistics.Win.Misc.UltraButton
COMBO-BOX	System.Windows.Forms.ComboBox Infragistics.Win.UltraWinEditors.UltraComboEditor
CONTROL-FRAME	Progress.Windows.Form System.Windows.Forms.Form
DIALOG-BOX	Progress.Windows.Form System.Windows.Forms.Form
EDITOR	System.Windows.Forms.TextBox Infragistics.Win.UltraWinEditors.UltraTextEditor
FIELD-GROUP	—
FILL-IN	System.Windows.Forms.DateTimePicker System.Windows.Forms.TextBox Infragistics.Win.UltraWinEditors.UltraCurrencyEditor Infragistics.Win.UltraWinEditors.UltraDatetimeEditor Infragistics.Win.UltraWinEditors.UltraNumericEditor Infragistics.Win.UltraWinEditors.UltraTextEditor

ABL Widget	Similar OpenEdge GUI for .NET forms and controls
FRAME	Progress.Windows.UserControl Progress.Windows.WindowContainer System.Windows.Forms.FlowLayoutPanel System.Windows.Forms.GroupBox System.Windows.Forms.Panel System.Windows.Forms.SplitContainer Infragistics.Win.Misc.UltraGroupBox
IMAGE	System.Windows.Forms.ImageList Infragistics.Win.UltraWinEditors.UltraPictureBox
LITERAL	System.Windows.Forms.Label Infragistics Win.Misc.UltraLabel
MENU MENU-ITEM SUB-MENU	System.Windows.Forms.ContextMenuStrip System.Windows.Forms.ToolStrip System.Windows.Forms.ToolStripContainer Infragistics.Win.UltraWinListBar.UltraListBar Infragistics.Win.UltraWinExplorerBar.UltraExplorerBar Infragistics.Win.UltraWinToolbars.UltraToolbarsManager
RADIO-SET	System.Windows.Forms.RadioButton Infragistics.Win.UltraWinEditors.UltraOptionSet
RECTANGLE	System.Drawing.Rectangle[5] Infragistics.Documents.Graphics.Rectangle[5]
SELECTION-LIST	System.Windows.Forms.ListView Infragistics.Win.UltraWinGrid.UltraCombo Infragistics.Win.UltraWinListView.UltraListView
SLIDER	System.Windows.Forms.TrackBar[5]
TEXT	System.Windows.Forms.Label Infragistics Win.Misc.UltraLabel

[5] Not installed with Visual Designer as a visual design component.

ABL Widget	Similar OpenEdge GUI for .NET forms and controls
TOGGLE-BOX	System.Windows.Forms.CheckBox Infragistics.Win.UltraWinEditors.UltraCheckEditor
WINDOW	Progress.Windows.Form Progress.Windows.MDIChildForm System.Windows.Forms.Form[6]

OpenEdge GUI ABL window with a browse compared to an OpenEdge GUI for .NET form with a grid

Whether you are working with the OpenEdge GUI or with the OpenEdge GUI for .NET, the steps you take to create a functional window with a browse or data grid are similar using both technologies—only the syntax is different. In either case you need to do the following:

- Create the window or form
- Define the query
- Enable the browse or data grid
- Display the window or form
- Wait for the window or form to close

The following table lists various ABL statements that compare how you build a window with a browse using the OpenEdge GUI compared to building a form with a data grid using the OpenEdge GUI for .NET. Note that the OpenEdge GUI for .NET code assumes the following `USING` statements:

```
USING System.Windows.Forms.* FROM ASSEMBLY.
USING Infragistics.Win.UltraWinGrid.* FROM ASSEMBLY.
```

[6] Progress Software Corporation recommends that you always use Progress.Windows.Form, because this form class is designed especially for use in an ABL session.

Table 3: OpenEdge GUI ABL window with a browse compared to an OpenEdge GUI for .NET form with a grid

OpenEdge GUI	OpenEdge GUI for .NET

`DEFINE VARIABLE C-Win AS HANDLE.`
`CREATE WINDOW C-Win ASSIGN . . .`
	`DEFINE VARIABLE custForm`
	`  AS CLASS Progress.Windows.Form.`
	`custForm`
	`  = NEW Progress.Windows.Form( ).`

`DEFINE QUERY custQry FOR Customer`
`  SCROLLING.`
`. . .`
	`DEFINE QUERY custQry FOR Customer`
	`  SCROLLING.`
	
	`. . .`
	`DEFINE VARIABLE custSource AS`
	`  Progress.Data.BindingSource.`
	`custSrc = NEW  Progress.Data.BindingSource(`
	`  custQry ).`

`DEFINE BROWSE CustBrowse QUERY`
`  CustQry . . .`
	`DEFINE VARIABLE custGrid`
	`  AS UltraGrid.`
	`custGrid = NEW UltraGrid( ).`
	`custGrid:DataSource = custSrc.`

`DEFINE FRAME Default-Frame CustBrowse`
`. . .`
 	
`ENABLE CustBrowse`
`  WITH FRAME Default-Frame`
`  IN WINDOW C-Win.`
	`custForm:Controls:Add( custGrid ).`
	`. . .`
	
	`/* controls enabled by default */`

`VIEW C-Win.`	`custForm:Show( ).`
`WAIT-FOR CLOSE OF THIS-PROCEDURE.`	`WAIT-FOR Application:Run( custForm ).`

`ON WINDOW-CLOSE OF C-Win DO:`
`  APPLY "CLOSE":U TO THIS-PROCEDURE`
	`custForm:FormClosing:Subscribe(`
	`  Form_Closing ).`
	
	`METHOD PRIVATE VOID`
	`  Form_Closing ( . . . ):`
	`  . . .`
	`END METHOD.`

`ON CLOSE OF THIS-PROCEDURE DO:` `  . . .`	—

OpenEdge GUI dialog compared to OpenEdge GUI for .NET dialog

The OpenEdge GUI provides two types of dialogs, a modal dialog box that you can display using a frame widget and a modal message box with several built-in options that you can display using a `MESSAGE` statement. The OpenEdge GUI for .NET provides a similar choice of dialogs with a richer set of options, one based on .NET forms and the other based on a message box that you can display using the `Show( )` method of the `MessageBox` class.

The following table shows implementations for comparable dialogs using each type of GUI.

Table 4: OpenEdge GUI dialog compared to OpenEdge GUI for .NET dialog

OpenEdge GUI dialog	OpenEdge GUI for .NET dialog

`DEFINE BUTTON bOK AUTO-ENDKEY DEFAULT`
`  LABEL "OK".`
`DEFINE FRAME frmCustomer`
`  "Hello, world!"`
`  bOK`
`  WITH FRAME frmCustomer`
`  DEFAULT-BUTTON bOK`
`  VIEW-AS DIALOG-BOX.`
 	
`ENABLE ALL WITH FRAME frmCustomer.`
`VIEW FRAME frmCustomer.`
	`USING Infragistics.Win.Misc.*.`
	`USING System.Windows.Forms.*.`
	
	`DEFINE VARIABLE rForm`
	`  AS Progress.Windows.Form.`
	`DEFINE VARIABLE rLabel`
	`  AS UltraLabel.`
	`DEFINE VARIABLE rOKButton`
	`  AS UltraButton.`
	
	`rForm = NEW Progress.Windows.Form( ).`
	`rLabel = NEW UltraLabel( ).`
	`rOKButton = NEW UltraButton( ).`

OpenEdge GUI dialog	OpenEdge GUI for .NET dialog

`WAIT-FOR WINDOW-CLOSE`
`  OF FRAME frmCustomer.`
	`rLabel:Text = "Hello, world!".`
	`rOKButton:Text = "OK".`
	`rOKButton:Left = rLabel:Width + 1.`
	`rOKButton:DialogResult`[7]
	`  = DialogResult:Ok.`
	`rForm:AcceptButton = rOKButton.`
	`rForm:Controls:Add(rLabel).`
	`rForm:Controls:Add(rOKButton).`
	`rForm:Height = 3 * rOKButton:Height.`
	`rForm:Width = rOKButton:Right + 10.`
	
	`WAIT-FOR rForm:ShowDialog( ).`
	`rForm:Dispose( )`[8]

`MESSAGE "Hello, world!"`
`  VIEW-AS ALERT-BOX.`
	`USING System.Windows.Forms.*.`
	
	`MessageBox:Show("Hello, world!").`

OpenEdge GUI behavior compared to OpenEdge GUI for .NET behavior

The following table lists mechanisms for a variety of common behaviors that are available with the OpenEdge GUI, and for each such mechanism, it lists one or more OpenEdge GUI for .NET mechanisms that provide identical or similar behavior.

Table 5: OpenEdge GUI behavior compared to OpenEdge GUI for .NET behavior

OpenEdge GUI behavior	OpenEdge GUI for .NET behavior
`ALWAYS-ON-TOP` attribute	`TopMost` property on a .NET form or control.
`APPLY 'ENTRY'` statement	`Focus( )` method on a .NET control.
`AUTO-END-KEY` attribute	`CancelButton` property on `Progress.Windows.Form`, which specifies the button clicked when the user presses the **Esc** key.
`AUTO-GO` attribute	`Assign( )` method on a `Progress.Data.BindingSource` object.

[7] Setting the DialogResult property of a button to an appropriate DialogResult enumeration value causes this value to be automatically assigned to the DialogResult property of the dialog-box Form class when the user clicks the button in the form. Setting this property on a modal form automatically causes the dialog box to close and the ShowDialog( ) method to return. If you do not set the DialogResult property on a button, you must handle a button event (such as Click) in order to explicitly set the property on the form and close the dialog box.

[8] When a GUI for .NET dialog box closes, .NET does not garbage collect the dialog box object unless you also call the Close( ) method on the dialog box, which automatically calls the Dispose( ) method to prepare the object for garbage collection. When a dialog box closes, .NET hides it from view so it can be re-displayed without having to re-create the object. So, to ensure that .NET garbage collects a dialog box after it closes, you must call Dispose( ) on the object to tell .NET to garbage collect it.

OpenEdge GUI behavior	OpenEdge GUI for .NET behavior
BGCOLOR attribute	BackColor property on a .NET control. The value is a System.Drawing.Color object.
CANCEL-BUTTON attribute	CancelButton property on Progress.Windows.Form, which specifies the button clicked when the user presses the **Esc** key.
DEFAULT-BUTTON attribute	AcceptButton property on Progress.Windows.Form, which specifies the button clicked when the user presses the **Enter** key.
DISABLE statement	Enabled property on a .NET for or control.
DROP-TARGET attribute	AllowDrop property on a .NET form or control.
COLUMN (X) and ROW (Y) attributes	Location property on any .NET control. The value is a System.Drawing.Location object. A Left property, Top property, and Right property is also available on .NET forms and controls.
ENABLE statement	Activate( ) or Focus( ) method on a .NET form or control.
EXTENT statement executed on an ABL indeterminate array or an ABL determinate array assigned to an ABL indeterminate array; and the EXTENT function to return the size of that ABL array	An INTEGER parameter on the static CreateInstance( ) method on System.Array to set the size of a newly-created one-dimensional .NET array, and the Length property (or GetLength( ) method) on an instance of the .NET array class to return that value.
FGCOLOR attribute	ForeColor property on a .NET form or control. The value is a System.Drawing.Color object.
FONT attribute	Font property on a .NET form or control. The value is a System.Drawing.Font object.
GET-TEXT-WIDTH-PIXELS( ) method	MeasureText( ) method of the System.Windows.Forms.TextRenderer class to obtain the size of a given character string in a given font; or the AutoSize property, which you can set to TRUE on a control that you want to resize itself based on its contents.
HIDDEN attribute	Visible property on a .NET form or control.
HEIGHT-* and WIDTH-* attributes	Size property on a .NET form or control. The value is a System.Drawing.Size object. A Height property and Width property is also available on .NET forms and controls.
HIDE statement	Hide( ) method on a .NET form or control.
ICON attribute	Icon property on a .NET form. The value is a System.Drawing.Icon object.

OpenEdge GUI behavior	OpenEdge GUI for .NET behavior
ON event statement	OpenEdge GUI for .NET Subscribe( ) method—invoked using the following general syntax: publisher-objref:event-name:Subscribe(method-or-procedure).
MOUSE-POINTER attribute	Cursor property on the .NET form or control.
MOVABLE attribute	DoDragDrop( ) method on a .NET control.
MOVE-AFTER-TAB-ITEM( ) andMOVE-BEFORE-TAB-ITEM( ) methods	TabIndex and TabStop properties of a .NET control.
MOVE-TO-BOTTOM( ) method	SendToBack( ) method on a .NET form or control.
MOVE-TO-TOP() method	BringToFront( ) method on a .NET form or control.
PARENT attribute	Owner property on a .NET form.
PRIVATE-DATA attribute	Tag property on a .NET form or control. The value is a System.Object, which can be any type of .NET data.
READ-ONLY attribute	ReadOnly property on the .NET control.
RESIZABLE attribute	Locked property on a form or control design component in the Visual Designer of Progress Developer Studio for OpenEdge.
RETURN NO-APPLY statement/option	Cancel property in a System.ComponentModel.CancelEventArgs object, which is passed as a parameter to a cancellable event handler, such as for the Validating event of a System.Windows.Forms.Button or other .NET control.
ROW (Y) and COLUMN (X) attributes	See the COLUMN (X) and ROW (Y) attributes in this table.
SCREEN-VALUE attribute	Text property on a .NET control.
SELECTABLE attribute	Locked property on a form or control design component in the Visual Designer of Progress Developer Studio for OpenEdge.
SENSITIVE attribute	Enabled property on a .NET form or control.
TITLE attribute	Text property on a .NET form or control.
TOOLTIP attribute	Add the UltraToolTipManager component to a form. Set the ToolTipText property on the UltraToolTipInfo object instantiated for the .NET control.
TOP-ONLY attribute	No similar .NET behavior.

OpenEdge GUI behavior	OpenEdge GUI for .NET behavior
VIEW statement	Show( ) method on a .NET form or control. Requires an active OpenEdge GUI for .NET WAIT-FOR statement in order for a form and its controls to become visible.
VISIBLE attribute	Visible property on a .NET form or control.
WIDTH-* and HEIGHT-* attributes	See the HEIGHT-* and WIDTH-* attributes in this table.
Widget labels	Text property on a .NET control. Also, the System.Windows.Forms.Label control or the Infragistics Win.Misc.UltraLabel control.
Widget siblings	NEXT-SIBLING property and PREV-SIBLING property on all class instances, including .NET objects
Widget tab order	See the MOVE-AFTER-TAB-ITEM( ) and MOVE-BEFORE-TAB-ITEM( ) methods in this table.
Widget Z order	See the MOVE-TO-BOTTOM( ) method or MOVE-TO-TOP() method in this table.

OpenEdge GUI events compared to OpenEdge GUI for .NET events

The following table lists high-level ABL widget events and their OpenEdge GUI for .NET counterparts (if any).

Table 6: OpenEdge GUI events compared to OpenEdge GUI for .NET events

OpenEdge GUI event	OpenEdge GUI for .NET event
CHOOSE	**Click** — Button control, UltraButton control, MenuItem control **ToolClick** — UltraToolBarsManager control
DEFAULT-ACTION	No similar event in .NET
DROP-FILE-NOTIFY	**DragDrop** — Control base class for all controls
END	**OffEnd** — Progress.Data.BindingSource control
END-SEARCH	No similar event in .NET
ENTRY	**AfterCellActivate** — UltraGrid control **AfterRowActivate** — UltraGrid control **Enter** — Control base class for all controls **CellEnter** — DataGridView control **GotFocus** — Control base class for all controls **RowEnter** — DataGridView control
HOME	No similar event in .NET
ITERATION-CHANGED	See VALUE-CHANGED in this table
LEAVE	**BeforeRowDeactivate** — UltraGrid control **BeforeCellDeactivate** — **UltraGrid** control **CellLeave** — DataGridView control **Leave** — Control base class for all controls **RowLeave** — DataGridView control
MENU-DROP	**AfterApplicationMenuDropDown** — UltraToolBarsManager control **MenuActivate** — MenuStrip control
OFF-END	No similar event on the DataGridView control. Can invoke the GetNextRow( ) method on the DataGridRowCollection returned by the Rows property. If the return value of the method is -1, the method is attempting to read beyond the last row in the DataGridView control.

OpenEdge GUI event	OpenEdge GUI for .NET event
	No similar event on the `UltraGrid` control. Can test the `HasNextSibling` property on the current `UltraGridRow` control to determine if there is a following (sibling) row in the `UltraGrid`.
`OFF-HOME`	No similar event on the `DataGridView` control. Can invoke the `GetPreviousRow( )` method on the `DataGridRowCollection` returned by the `Rows` property. If the return value of the method is -1, the method is attempting to read before the first row in the `DataGridView` control. No similar event on the `UltraGrid` control. Can test the `HasPrevSibling` property on the current `UltraGridRow` control to determine if there is a previous (sibling) row in the `UltraGrid`.
`PARENT-WINDOW-CLOSE`	No similar event in .NET
`ROW-DISPLAY`	**`AfterRowActivate`** — `UltraGrid` control **`AfterRowExpanded`** — `UltraGrid` control **`RowPostPaint`** — `DataGridView` control
`ROW-ENTRY`	**`AfterRowActivate`** — `UltraGrid` control **`RowEnter`** — `DataGridView` control
`ROW-LEAVE`	**`BeforeRowDeactivate`** — `UltraGrid` control **`RowLeave`** — `DataGridView` control
`SCROLL-NOTIFY`	**`AfterCardsScroll`** — `UltraGrid` control **`AfterColRegionScroll`** — `UltraGrid` control **`AfterRowRegionScroll`** — `UltraGrid` control **`MouseWheel`** — `Control` base class for all controls **`Scroll`** — `DataGridView` control
`START-SEARCH`	**`ColumnHeaderMouseClick`** — `DataGridView` control **`DoubleClickHeader`** — `UltraGrid` control
`VALUE-CHANGED`	**`AfterCellUpdate`** — `UltraGrid` control **`CellChange`** — `UltraGrid` control **`CellValueChanged`** — `DataGridView` control **`CheckedChanged`** — `CheckBox` control **`ItemSelectionChanged`** — `ListView` control **`RowEnter`** — `DataGridView` control **`SelectionChangeCommitted`** — `UltraComboEditor` control **`SelectionChanged`** — `DataGridView` control (and others)

OpenEdge GUI event	OpenEdge GUI for .NET event
	SelectedValueChanged — ComboBox control **TextChanged** — TextBox control (and others) **ToolValueChanged** — UltraToolbarsManager control
WINDOW-CLOSE	**FormClosed** — System.Windows.Forms.Form control[9] FormClosing — System.Windows.Forms.Form control[9]
WINDOW-MAXIMIZED	No similar event on a form control. Test the WindowState form property for a FormWindowState enumeration value of Maximized.
WINDOW-MINIMIZED	No similar event on a form control. Test the WindowState form property for a FormWindowState enumeration value of Minimized.
WINDOW-RESIZED	**ResizeBegin** — System.Windows.Forms.Form control[9] **ResizeEnd** — System.Windows.Forms.Form control[9]
WINDOW-RESTORED	No similar event on a form control. Test the WindowState form property for a FormWindowState enumeration value of Normal.

Common public .NET form and control methods

The following table lists common public .NET form and control methods:

Table 7: Common public .NET form and control methods

Method	Description
Activate( )	Brings the form to the front and publishes the Activated event.
Close( )	Closes the open form, publishing the FormClosing event followed by the FormClosed event. This method also calls the Dispose( ) method on the form.

[9] Progress Software Corporation recommends that you use the OpenEdge-derived form class, Progress.Windows.Form, instead.

Method	Description
`Dispose( )`	Releases all resources held by the control or control container in preparation for garbage collection. You cannot override this method, as it is defined as `FINAL` in ABL. Note that the object on which you call this method is not garbage collected immediately and the object reference continues to work until .NET garbage collects the object. **Note:** Typically, you never have to call this method directly in ABL. If you execute the ABL DELETE OBJECT statement on the form or control object reference, this sufficiently prepares the .NET object for garbage collection wherever you might otherwise call the `Dispose( )` method in .NET. However, for .NET modal forms (dialog boxes, including ABL-derived forms), you must call this method to ensure that the form object is garbage collected. For more information, see the sections on blocking for .NET dialog boxes in *OpenEdge Development: GUI for .NET Programming*. Before executing `DELETE OBJECT`, you should also call this method on any instances of `Progress.Data.BindingSource` that you use to bind a ProDataSet to a .NET control. For more information, see the sections on using the `Dispose( )` method of a ProBindingSource, also in *OpenEdge Development: GUI for .NET Programming*.
`Focus( )`	Sets input focus on the control.
`Hide( )`	Conceals the control or control container from the user by setting its `Visible` property to `FALSE`.
`ResumeLayout( )`	Resumes layout logic for the control or control container after it was suspended using the `SuspendLayout( )` method. Typically invoked on a form or other control container with a single `LOGICAL` argument set to `FALSE` in order to suppress the immediate resumption of pending layout logic until after all initialization of the control container is complete. This represents a performance improvement, especially for initializing control containers that have large numbers of child controls added.
`Show( )`	Initializes the control or control container for display or reveals the form to the user by setting its `Visible` property to `TRUE`. You do not always need to call this method to initially display a form, depending on how you block for input on the form.
`ShowDialog( )`	Displays and blocks for input on the form as a modal dialog box. You can **only** call this method using the `WAIT-FOR` statement for .NET forms (.NET `WAIT-FOR` statement). **Note:** To block for input on the form as a non-modal window (whether for a single form or for an MDI form), you must call the static `Run( )` method on the `System.Windows.Forms.Application` class from within a .NET `WAIT-FOR` statement.
`SuspendLayout( )`	Temporarily suspends layout of the control, form, or other control container in order to prevent `Control.Layout` events from firing unnecessarily while completing control initialization. After control container initialization completes, you invoke the `ResumeLayout( )` method in order to allow the control initialization settings to take effect.

Common public .NET form properties

The following table lists common public .NET form properties,

Table 8: Common public .NET form properties

Property	Description
`AcceptButton`	Gets or sets the object reference to the button control on the form whose `Click` event is published when the user presses the **ENTER** key.
`Bounds`	Gets or sets a `System.Drawing.Rectangle` structure that specifies the location and size of the form within the client area of the screen.
`CancelButton`	Gets or sets an object reference to the button control on the form whose `Click` event is published when the user presses the **ESC** key.
`ClientSize`	Gets or sets a `System.Drawing.Size` structure that specifies the client area of the form. The client area is everything within the form except non-client elements, such as the title bar of a form.
`ControlBox`	Gets or sets an ABL `LOGICAL` value that indicates if the control box icon is displayed in the caption (title) bar of the form. The control box is a system control that provides access to the system menu for the form. The icon used to represent it is specified by the `Icon` property.
`Controls`	Gets the object reference to the `System.Windows.Forms.ControlCollection` class instance for the form. Every .NET form has a control collection where you can add or remove all the application controls that the form contains, such as button, text, grid, and menu controls that you create and initialize for the form.
`DialogResult`	Gets or sets a `System.Windows.Forms.DialogResult` enumeration value that indicates the result of a form displayed as a dialog box.
`Enabled`	Gets or sets a value indicating whether the control can respond to user interaction.
`FormBorderStyle`	Gets or sets a `System.Windows.Forms.FormBorderStyle` enumeration value that indicates whether the form can be resized by the user and how the graphics of the frame border appear and function.
`HelpButton`	Gets or sets a value indicating whether a Help button should be displayed in the caption box of the form.
`Icon`	Gets or sets the object reference to a `System.Drawing.Icon` class instance that represents the icon displayed for the form on the taskbar and also the icon for the control box enabled by the `ControlBox` property.
`Location`	Gets or sets a `System.Drawing.Point` structure that specifies the pixel screen coordinates of the upper-left-hand corner of a form. You can also use the `Top` and `Left` properties to get or set the location of the form.

Property	Description
MainMenuStrip	Gets or sets the object reference to a System.Windows.Forms.MenuStrip class instance that represents the primary menu container for the form.
MaximizeBox	Gets or sets an ABL LOGICAL value indicating whether the **Maximize** button is displayed in the title bar of the form.
MdiParent	Gets or sets the current multiple-document interface (MDI) parent form of this (MDI child) form.
MinimizeBox	Gets or sets an ABL LOGICAL value indicating whether the **Minimize** button is displayed in the title bar of the form.
Modal	Gets an ABL LOGICAL value indicating if the form is displayed modally (as a dialog box).
Owner	Gets or sets the form that owns this form. OpenEdge also allows .NET forms to own ABL windows and for ABL windows to own .NET forms using the ABL PARENT attribute. For more information, see the chapter on .NET forms in *OpenEdge Development: GUI for .NET Programming*.
Size	Gets or sets a System.Drawing.Size structure that specifies the width and height of the form in pixels. You can also use the Height and Width properties to get or set the size of the form.
StartPosition	Gets or sets the starting position of the form at run time. This is a FormStartPosition enumeration value.
Text	Gets or sets an ABL CHARACTER value that represents the title displayed in the title (caption) bar of the form.
TopMost	Gets or sets a value indicating whether the form should be displayed as a topmost form.
Visible	Gets or sets an ABL LOGICAL value that indicates whether the form is visible or can be displayed on the screen. The form Show( ) method also sets this property to TRUE. You do not always need to initialize this property to TRUE in order to initially display the form, depending on how you block for input on the form.
WindowState	Gets or sets the form's window state. This is a FormWindowState enumeration value that indicates the form is maximized, minimized, or displayed normally.

Common public .NET control properties

The following table lists common public .NET control properties.

Table 9: Common public .NET control properties

Property	Description
Anchor	Gets or sets a System.Windows.Forms.AnchorStyles enumeration value that specifies the edges of the container to which a control is bound and determines how a control is resized with its parent container. **Note:** This property is mutually exclusive with and manages somewhat different behavior than the Dock property. The last property that you set takes precedence over the other.
Bottom	Gets or sets an ABL INTEGER value that specifies the pixel distance between the bottom edge of the control and the top edge of the client area of its control container.
Bounds	Gets or sets a System.Drawing.Rectangle structure that specifies the location and size of the control or control container within the client area of its control container. For a form, the container is the screen area.
ClientSize	Gets or sets a System.Drawing.Size structure that specifies the client area of the control or control container. The client area is everything within the bounds of the control or control container except non-client elements, such as the title bar of a form.
Controls	Gets the object reference to the System.Windows.Forms.ControlCollection class instance for the form or other control container. Every .NET form or control container has a control collection where you can add or remove all the application controls that the form or control container contains, such as button, text, grid, and menu controls that you create and initialize for the form or control container. For more information on working with control collections, see the section on working with collections in *OpenEdge Development: GUI for .NET Programming*.
DataBindings	Gets the data bindings for the control. This property returns a ControlBindingsCollection object.
DataSource	Gets or sets the data source that the Microsoft BindingSource object binds to. See also the **Handle** property.
DataMember	Gets or sets the specific list in the data source to which the Microsoft BindingSource object currently binds to. See also the TableSchema property.
Dock	Gets or sets a System.Windows.Forms.DockStyles enumeration value that specifies which borders of the control are docked to its parent container and determines how a control is resized with its parent. **Note:** This property is mutually exclusive with and manages somewhat different behavior than the Anchor property. The last property that you set takes precedence over the other.
Enabled	Gets or sets an ABL LOGICAL value indicating whether the control can respond to user interaction.
Font	Gets or sets the font for the control or control container.

Property	Description
GenerateMember	A design-time property provided for each design component that specifies when the Visual Designer generates a variable data member for a control that you have added to the form.
Handle	Gets or sets the handle to the ABL data source object that the Progress.Data.BindingSource (ProBindingSource) object binds to, which can be a buffer, query, or ProDataSet.
Height	Gets or sets an ABL INTEGER value that specifies the pixel height of the control.
Left	Gets or sets an ABL INTEGER value that specifies the pixel distance between the left edge of the control and the left edge of the client area of its control container.
Location	Gets or sets a System.Drawing.Point structure that specifies the pixel coordinates of the upper-left-hand corner of the control within the client area of its parent control container.
Modifiers	A design-time property provided for each design component that specifies the access mode (PRIVATE or PROTECTED) that the Visual Designer generates for the variable data member associated with a control that you have added to the form.
Name	Gets or sets a name for a .NET control, which is a character string that you can use to identify the control other than by its variable name or type name.
Parent	Gets or sets the object reference to a System.Windows.Forms.Control instance that is the parent form or control container for the control.
Right	Gets or sets an ABL INTEGER value that specifies the pixel distance between the right edge of the control and the left edge of the client area of its control container.
Size	Gets or sets a System.Drawing.Size structure that specifies the width and height of the control or control container in pixels.
TabIndex	Gets or sets an ABL INTEGER value that specifies the zero(0)-based tab order of the control within its control container. TabIndex values do not have to be contiguous. .NET simply observes the INTEGER order of the specified values.
TableSchema	Gets or sets the Progress.Data.TableDesc object to a Progress.Data.BindingSource (ProBindingSource) object that describes the table schema for the data source that the ProBindingSource object binds to.
TabStop	Gets or sets an ABL LOGICAL value indicating if the control participates in the tab order. **Note:** This property is always TRUE for an instance of any System.Windows.Form class (including Progress.Windows.Form).
Tag	Gets or sets any data that can be represented as a System.Object value. This property on a .NET control can be used like the PRIVATE-DATA attribute on an ABL widget.
Text	Gets or sets an ABL CHARACTER value that represents a string value displayed for the control or control container, as defined for the particular control or control container.

Property	Description
Top	Gets or sets an ABL INTEGER value that specifies the pixel distance between the top edge of the control and the top edge of the client area of its control container.
Visible	Gets or sets an ABL LOGICAL value that indicates whether the control or control container is visible to the user. The control or control container Show( ) method also sets this property to TRUE.
Width	Gets or sets an ABL INTEGER value that specifies the pixel width of the control.

Common public .NET form events

The following table lists common public .NET form events:

Table 10: Common public .NET form events

Event	Description
Activated	Raised when the form receives focus. The form also publishes this event when you call the form Activate( ) method.
Deactivate	Raised when the form loses focus.
FormClosed	Raised when the form is closed by the user, by the form Close( ) method, or by the static Exit( ) method on the System.Windows.Forms.Application class.
FormClosing	Raised as the form is closing, but before the FormClosed event is raised. You can set the Cancel property on the FormClosingEventArgs parameter when handling this event to cancel the close of the form and prevent the FormClosed event from being raised.
Load	Raised before the form is displayed for the first time, allowing you to allocate resources for and further configure the form before it actually becomes visible.
Shown	Raised immediately after the form is displayed for the first time.

Common public .NET control events

The following table lists common public .NET control events.

Table 11: Common public .NET control events

Event	Description
`Click`	Raised when the control or control container is clicked. **Note:** This is a higher-level event than a `MouseClick` event and is raised for other actions, such as pressing the **Enter** key when the control has focus.
`DoubleClick`	Raised when the control or control container is double-clicked.
`EnabledChanged`	Raised when the `Enabled` property value of the control or control container has changed.
`Enter`	Raised when the control or control container receives focus from any of several different actions. **Note:** This is the first of several events raised in a particular order, depending on how focus is change to the control.
`Move`	Raised when the control or control container is moved.
`Paint`	Raised when the control or control container is redrawn. This occurs for a number of different actions, such as resizing and moving other controls over this control. This event is especially useful for maintaining graphics that you draw for a control as its position and screen environment changes. **Note:** Use cautiously, and only when necessary, as it can impact performance.
`TextChanged`	Raised when the `Text` property of the control or control container changes value.
`VisibleChanged`	Raised when the `Visible` property of the control or control container changes value.

Implicit data type mapping between ABL primitive and .NET types

The following table lists implicit data type mappings between ABL primitive types and corresponding .NET data types. This implicit mapping means that the corresponding data types can be assigned to each other:

Table 12: Implicit data type mapping between ABL primitive and .NET types

Implicit .NETobject type	Implicit C#primitive type	Implicit ABLprimitive type
`System.Boolean`	`bool`	`LOGICAL`
`System.Byte`	`byte`	`INTEGER`

Implicit .NETobject type	Implicit C#primitive type	Implicit ABLprimitive type
System.SByte	sbyte	INTEGER
System.DateTime	—	DATETIME
System.Decimal	decimal	DECIMAL
System.Int16	short	INTEGER
System.UInt16	ushort	INTEGER
System.Int32	int	INTEGER
System.UInt32	uint	INT64
System.Int64	long	INT64
System.UInt64	ulong	DECIMAL
System.Double	double	DECIMAL
System.Single	float	DECIMAL
System.Char	char	CHARACTER
System.String	string	CHARACTER LONGCHAR

ABL also automatically maps one-dimensional .NET arrays to ABL arrays. So, assuming that the arrays are the same size, the corresponding .NET and ABL array types in the following table can be assigned to each other.

Table 13: Example implicit mappings between one-dimensional .NET arrays and ABL arrays

.NET array type	Corresponding ABL array type
System.Drawing.Point[]	System.Drawing.Point EXTENT
System.Windows.Forms.Button[]	System.Windows.Forms.Button EXTENT
System.Int32[]	INTEGER EXTENT
System.Byte[]	INTEGER EXTENT

Notice that you can assign between the same ABL primitive array type and a .NET array of the same size and any implicitly mapped .NET type. So, the same ABL `INTEGER` array can be assigned to a .NET array of `System.Int32` or `System.Byte` (as shown), or of any other supported mapping, such as `System.Int16`, and so on.

The following table lists the default match between the specified ABL primitive type and its corresponding .NET data type.

Table 14: Default match between specified ABL primitive types and corresponding .NET data types

ABLprimitive type	Default match (.NET object type)	Default match (C# primitive type)
CHARACTER	System.String	string
DATETIME	System.DateTime	—
DECIMAL	ystem.Decimal	decimal
INT64	System.Int64	long
INTEGER	System.Int32	int
LOGICAL	System.Boolean	bool

This means that in the following situations, the ABL type matches the corresponding .NET type by default:

- When assigning the ABL primitive type to a `System.Object`
- When passing the ABL primitive type as a parameter to an overloaded .NET method
- When overriding a .NET method or implementing a .NET interface

For all of the above situations, you can also explicitly indicate any available .NET data type mapping other than the default (see Explicit data type mapping from ABL primitive to .NET types on page 44.)

The following table shows examples of implicit data type mappings.

Table 15: Example implicit mappings between ABL and .NET types

Operation description	Operation using *Type* of data element
Assign an `INTEGER` value to a `System.Int16` data element (creating a new `System.Int16` object in the process)	*`System.Int16 = INTEGER.`*
Assign a `System.Int16` value to an `INTEGER` data element	*`INTEGER = System.Int16.`*
Pass an `INTEGER` value to the `System.Int16` parameter of a .NET method (creating a new `System.Int16` object in the process)	*`Method( INPUT INTEGER).`*
Assign the elements of a `DECIMAL` array to a `System.Double[]` array data element (creating a new `System.Double[]` array object that contains all the element values of the `DECIMAL` array source)	*`System.Double[] = DECIMAL EXTENT.`*
Assign the elements of a `System.Double[]` array object to a `DECIMAL` array of the same extent	*`DECIMAL EXTENT = System.Double[].`*
Pass a `DECIMAL` array value to the `System.Double[]` parameter of a .NET method (creating a new `System.Double[]` array object containing all the element values of the `DECIMAL` array argument)	*`Method( OUTPUT DECIMAL EXTENT ).`*

Explicit data type mapping from ABL primitive to .NET types

Sometimes, when you assign or pass an ABL primitive type to a .NET property or when you pass an ABL primitive type to a .NET method parameter, and more than one .NET type mapping is possible, you must explicitly indicate the mapping you want. You might also need to indicate the explicit .NET type when you assign (and box) an ABL primitive type to a `System.Object` or when you pass the ABL primitive as an argument to a `System.Object` or overloaded .NET method parameter. You must indicate explicit .NET data types when you define an ABL method or property in order to override a .NET method or implement a .NET interface (see the C# .NET code mapped to OpenEdge GUI for .NET ABL on page 12 C# .NET code mapped to OpenEdge GUI for .NET ABL

For a .NET mapped type that is the default or only match for a given ABL data type, you only use the ABL data type to explicitly indicate the .NET type mapping. For all other .NET mapped types, ABL provides a special type keyword (ABL `AS` data type) that indicates the explicit .NET type mapping. The following table shows all the mappings between ABL and .NET primitive types as well as the `AS` data type keyword that you can use to indicate the explicit mapping to a .NET type other than the default match for the ABL primitive type.

Table 16: Explicit data type mapping from ABL primitive to .NET types

Explicit .NETobject type	Explicit C#primitive type	ABLprimitive type	ABLAS data type
System.Boolean	bool	LOGICAL	—
System.Byte	byte	INTEGER	UNSIGNED-BYTE
System.SByte	sbyte	INTEGER	BYTE
System.DateTime	—	DATETIME	—
System.Decimal	decimal	DECIMAL	—
System.Int16	short	INTEGER	SHORT
System.UInt16	ushort	INTEGER	UNSIGNED-SHORT
System.Int32	int	INTEGER	—
System.UInt32	uint	INT64	UNSIGNED-INTEGER
System.Int64	long	INT64	—
System.UInt64	ulong	DECIMAL	UNSIGNED-INT64
System.Double	double	DECIMAL	DOUBLE
System.Single	float	DECIMAL	FLOAT
System.Char	char	CHARACTER	SINGLE-CHARACTER
System.String	string	CHARACTER/ LONGCHAR	—

The following table shows examples of an explicit .NET type mapping using the AS data type keyword.

Table 17: Example explicit data type mappings using the AS data type keyword

Operation description	Operation with *Type* of data element
Assign an INTEGER as a System.Int16 value to a System.Object property or variable (creating a new System.Int16 object in the process).	*System.Object* = BOX(*INTEGER*, "SHORT").
Pass an INTEGER as a System.Int16 argument to the System.Object parameter of a .NET method (creating a new System.Int16 object in the process).	*Method*(INPUT *INTEGER* AS SHORT).