

1 Lecture 2: Controls, Events, and User Interaction

Course: Scientific Visual Programming

Topic: Windows Forms Controls and Event-Driven Programming

Recommended duration: 90 minutes

1.1 Learning objectives

Students should be able to select appropriate Windows Forms controls, handle keyboard and mouse events, use menus and dialogs, validate input, and explain the event-driven execution model.

1.2 1. Event-driven programming

In a procedural console program, execution often follows a predictable sequence from the first statement to the last. A Windows Forms application is different. After initialization, it waits for events such as a click, key press, selection change, timer tick, or form closing. The framework dispatches each event to the registered handler.

The application should respond quickly. Long calculations inside a UI event handler can make the form appear frozen. Expensive operations should be designed carefully and, when appropriate, executed asynchronously.

1.3 2. Input controls

TextBox is appropriate for short text or numeric input. Set `ReadOnly` when the user should not edit a displayed value. `MaskedTextBox` can guide formats such as telephone numbers. `NumericUpDown` is preferable to a free-form text box when the user selects a bounded numeric value.

`CheckBox` represents independent choices, such as "Include tax." `RadioButton` represents one choice in a mutually exclusive group. `ComboBox` is useful when the list is known but space is limited. `DateTimePicker` provides a structured date input and reduces formatting errors.

1.4 3. Validation and feedback

Validation should occur close to the point where invalid data enters the application. Use `ErrorProvider` to display a non-blocking error message beside a control. A form can validate one field when it loses focus and validate all fields before saving.

```
private bool ValidateForm()
{
    errorProvider1.Clear();
    bool valid = true;

    if (string.IsNullOrEmpty(txtEmail.Text))
    {
        errorProvider1.SetError(txtEmail, "Email is required.");
        valid = false;
    }

    if (numAge.Value < 18)
```

```

    {
        errorProvider1.SetError(numAge, "Age must be at least 18.");
        valid = false;
    }

    return valid;
}

```

Validation messages should say what is wrong and how to correct it. Avoid presenting only “Invalid input.”

1.5 4. Keyboard and mouse events

The `KeyPress` or `KeyDown` event can support keyboard behavior, but validation should not depend only on key filtering because users can paste content or input can arrive from other sources. The `KeyDown` event is useful for shortcuts such as `Ctrl+S`.

Mouse events include `MouseEnter`, `MouseLeave`, `MouseDown`, and `MouseMove`. Use them for meaningful interaction, not for effects that distract from the task. Accessibility and keyboard support should remain available.

1.6 5. Menus, toolbars, and dialogs

A `MenuStrip` organizes commands such as New, Open, Save, and Exit. A `ToolStrip` provides quick access to frequently used commands. Use the same method for a menu item and its toolbar button when they perform the same action.

Common dialogs include `OpenFileDialog`, `SaveFileDialog`, `ColorDialog`, and `FontDialog`. A dialog returns a result, so check the result before using its selected value:

```

using (var dialog = new OpenFileDialog())
{
    dialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        txtFilePath.Text = dialog.FileName;
    }
}

```

1.7 6. Timers and periodic actions

A `Timer` can raise an event at a regular interval. It is useful for clocks, progress indicators, and limited UI updates. It should not be used as a replacement for proper background processing. Stop the timer when the form closes or when the periodic action is no longer needed.

1.8 7. Designing a temperature converter

Create a form with a Celsius text box, a Fahrenheit text box, and two buttons. One button converts Celsius to Fahrenheit; the other performs the reverse conversion. Use `decimal.TryParse`, display two decimal places, and place validation errors beside the relevant input.

The conversion logic should be separate from the event handler:

```
private decimal CelsiusToFahrenheit(decimal celsius)
{
    return (celsius * 9m / 5m) + 32m;
}
```

1.9 8. Common mistakes

Do not place all application logic in one large `button_Click` method. Do not use message boxes for every minor validation issue. Do not ignore keyboard navigation or assume that a user always follows the happy path. Avoid duplicate event handlers that implement slightly different versions of the same command.

1.10 Practice activity

Build a student registration form with name, email, department, age, and enrollment date. Use at least four different controls, an `ErrorProvider`, a menu item for clearing the form, and a confirmation dialog before closing if unsaved data exists.

1.11 Summary

Windows Forms programming is event-driven. Controls collect input and generate events; handlers validate data and invoke application logic. Good interfaces provide immediate, specific feedback, support keyboard interaction, and avoid blocking the user interface.

1.12 References