

Handling User Input

User Input on a Mobile Web Form

A number of mobile Web Forms controls support user input, including the **TextBox** and **List** controls.

When constructing a form that allows user input, you can place one or more user input controls, along with other mobile controls such as labels, on the form. Limit the number of controls that you place on a single form, because you cannot raise an event for any of the controls until the user completes the entire form.

For most user input forms, you also need to provide a way for the user to submit input back to the server. You usually do this by placing a **Command** control on the form. The samples in this section all use such a technique.

Passwords and Numeric Input

By setting the **Password** or **Numeric** property of a **TextBox** mobile control, you can alter the behavior of the control. Setting the **Password** property renders a text box that hides its input. Setting the **Numeric** property of the control renders a text box that restricts its input to numeric characters. The following sample illustrates the use of these properties.

Important: These properties are only advisory, because the desired behavior works only on target devices that support it. For example, most HTML browsers cannot restrict input of a text box to numeric characters. However, the numeric mode is very useful on many cell phones, because the phones automatically place the keypad in numeric mode when editing a numeric text box.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
TagPrefix="mobile" %>
<html xmlns="http://www.w3.org/xhtml">
<body>

    <script runat="server">
```

```
protected void GoCommand_OnClick(Object sender, EventArgs e)
{
    try
    {
        int age = Int32.Parse(AgeEdit.Text);

        char[] name = NameEdit.Text.ToCharArray();
        Array.Reverse(name);

        Greeting.Text = String.Format("Your code name is {0}", new String(name)
                                     + age * 100);
    }
    catch (Exception ex)
    {
        Greeting.Text = "You entered an invalid age. Please go back and try
                        again.";
    }

    ActiveForm = SecondForm;
}

</script>

<mobile:Form Runat="server" ID="FirstForm">
    <mobile:Label Runat="server" ID="AgeLabel">Enter your age:</mobile:Label>
    <mobile:TextBox Runat="server" ID="AgeEdit" Numeric="true" />
    <mobile:Label Runat="server" ID="NameLabel">Enter your real name:</mobile:Label>
    <mobile:TextBox Runat="server" ID="NameEdit" Password="true" />
    <mobile:Command Runat="server" ID="GoCommand" OnClick="GoCommand_OnClick"
                    Text="Go!" />
</mobile:Form>

<mobile:Form ID="SecondForm" Runat="server">
    <mobile:Label Runat="server" ID="Greeting" />
    <mobile:Link ID="GoBack" Runat="server" NavigateUrl="#FirstForm">
        Go Back to Entry Form
    </mobile:Link>
</mobile:Form>
</body>
</html>
```

Using a Text View Control

To place text in a form you can place it either as if it was an HTML Text or use a **Label** Control which is suitable for small amount of text.

Another control that you can use to display larger amounts of text is the **TextView** control.

This control allows you to dynamically set text that includes some common markup tags. The tags ****, **<i>**, **<p>**, **
, and **<a> (in a limited form) are supported.

When placing large amounts of text on a single form, you can use the pagination feature of the form by setting the **Paginate** property to **true**. The form paginates its contents according to characteristics of the target device.

The following sample shows the **TextView** control in action. On a Pocket PC or similar device, the text occupies one page (though it does scroll). On a WML-based cell phone, the text renders over multiple screens. In each instance, the **TextView** control automatically provides UI for the user to browse between pages of text.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
TagPrefix="mobile" %>
<html xmlns="http://www.w3.org/xhtml">
<body>
  <mobile:Form Runat="server" ID="SingleForm" Paginate="true">
    <mobile:Label Runat="server" ID="CaptionLabel" StyleReference="title">
      Showing a Lot of Text
    </mobile:Label>
    <mobile:TextView Runat="server" ID="TextView1">
This sample shows use of the <b>TextView</b> control, which
can be used to render a large amount of text.
This control supports a number of common markup tags in its
text, and marked up text can be <i>dynamically</i> set on the control.
<br />
This sample turns on the <b>Paginate</b> property of the form to enable
automatic pagination. The entire contents of the form, including the text
contained in this control, are automatically
paginated according to the size of the target device.
The form provides UI to allow you to switch between
pages of text.
On a Pocket PC or similar device, this form will
only occupy one page (though it will scroll). On a four-line cellular phone, this form
will occupy multiple pages.
The default page size is automatically chosen based on the target
device, but you can programatically override it.
    </mobile:TextView>
  </mobile:Form>
</body>
</html>
```

Using a List Control

A typical feature of a mobile user interface is a menu that allows a user to easily browse to a choice of pages. One way that you can construct a menu in a mobile Web Forms page is by using the **List** mobile control.

The following sample shows a menu form using a **List** control. When the user clicks a choice, the control's **ItemCommand** event is raised. The page's code uses the clicked choice to alter the contents of the second form, and then navigate to the form.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>

<script runat="server">

    protected void Menu_OnItemCommand(Object sender, ListCommandEventArgs e)
    {
        this.RedirectToMobilePage(e.ListItem.Value);
    }

</script>

<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form Runat="server" ID="FirstForm">
        <mobile:Label Runat="server" ID="TitleLabel" StyleReference="title">
            Demo Menu
        </mobile:Label>
        <mobile:List Runat="server" ID="CityList"

            OnItemCommand="Menu_OnItemCommand"

            Decoration="Bulleted"

            ItemsAsLinks="True">

            <Item Text="TextBox Demo" Value="01-TextBox.aspx" />
            <Item Text="TextView Demo" Value="02-TextView.aspx" />
            <Item Text="List Demo" Value="03-List.aspx" />
            <Item Text="SelectionList Demo" Value="04-SelectionList.aspx" />
            <Item Text="Validation Demo" Value="05-Validation.aspx" />
        </mobile:List>
    </mobile:Form>
```

```
</body>
</html>
```

List Decoration

The **List** mobile control offers several rendering styles for target devices that support richer environments. The rendering style is set by using the **Decoration** property. Valid settings for this property come from the **ListDecoration** enumeration, and include the following.

Decoration	Description
None	Each list item is shown as is.
Bulleted	Each list item is bulleted.
Numbered	Each list item is numbered.

The following sample allows you to see all the supported rendering styles interactively. This sample works **only on HTML browsers**. (Other browsers do not currently support these styles.)

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>

<script runat="server">

    protected void MenuList_OnItemCommand(Object sender, ListCommandEventArgs e)
    {
        MenuList.Decoration =
            (ListDecoration)Enum.Parse(typeof(ListDecoration), e.ListItem.Text, true);
    }

</script>

<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form ID="MenuListForm" Runat="server">
        <mobile:Label Runat="server" ID="MenuLabel"
            StyleReference="title" Text="Menu List" />
        <mobile:List Runat="server" ID="MenuList"
            OnItemCommand="MenuList_OnItemCommand">
            <Item Text="None" />
            <Item Text="Bulleted" />
```

```
<Item Text="Numbered" />
</mobile:List>
</mobile:Form>
</body>
</html>
```

Using a Selection List Control

The **SelectionList** mobile control differs from the **List** mobile control in that it:

- Requires an additional submission mechanism on the form, such as a **Command** control, to send results back to the server.
- Does not generate **OnItemCommand** events.
- Maintains state information about one or more currently selected items.
- Provides different renderings by using list boxes, drop-down lists, option buttons (radio buttons), and checklists.

The following sample shows a selection list at work. Because state information is maintained across forms, the second form on the page is data bound to the current selection of the selection list.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>

<script runat="server">
    protected void GoCommand_OnClick(Object sender, EventArgs e)
    {
        BoundCompanyLabel.Text = Company.Text;
        ActiveForm = SecondForm;
    }
    protected void SecondForm_OnActivate(Object sender, EventArgs e)
    {
        SecondForm.DataBind();
    }
</script>
<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form Runat="server" ID="FirstForm">
        <mobile:Label Runat="server" ID="TitleLabel" StyleReference="title">
            Company Directory Lookup
        </mobile:Label>
```

```

<mobile:Label Runat="server" ID="CompanyLabel">
    Enter company name:
</mobile:Label>
<mobile:TextBox Runat="server" ID="Company" />
<mobile:Label Runat="server" ID="IndustryLabel">
    Choose an industry:
</mobile:Label>
<mobile:SelectionList Runat="server" ID="IndustryList">
    <Item Text="Communications" Value="COMM" />
    <Item Text="Energy" Value="ENGY" />
    <Item Text="Finance" Value="FNCE" />
</mobile:SelectionList>
<mobile:Command Runat="server" ID="GoCommand"
    OnClick="GoCommand_OnClick" Text="Go" />
</mobile:Form>
<mobile:Form ID="SecondForm" Runat="server" OnActivate="SecondForm_OnActivate">
    <mobile:Label Runat="server" id="InfoLabel"
        StyleReference="title" Text="You Selected:" />
    <mobile:Label Runat="server" ID="BoundCompanyLabel"></mobile:Label>
    <mobile:Label Runat="server" ID="BoundIndustryLabel">
        Industry: <%= IndustryList.Selection.Value %>
    </mobile:Label>
</mobile:Form>
</body>
</html>

```

Multi-Mode Selection List

The data features of the **SelectionList** control are very similar to the **List** control. However, the **SelectionList** control differs in that it is used exclusively to represent selection in a form, and requires another control, such as a **Command** control, to submit the selection. In addition to single selection, the **SelectionList** control supports multiple selection, rendering a UI appropriate for the device.

The following sample demonstrates use of a **SelectionList** control in multi-select mode, and shows how to use the selection information.

```

<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>
<script runat="server">

```

```
protected void CityForm_OnSubmit(Object sender, EventArgs e)
{
    string citiesChosen = "";
    string separator = ", ";
    foreach (MobileListItem item in CityList.Items)
    {
        if (item.Selected)
        {
            if (citiesChosen.Length > 0)
            {
                citiesChosen += separator;
            }
            citiesChosen += item.Text;
        }
    }
    if (citiesChosen.Length == 0)
    {
        citiesChosen = "none";
    }
    CityLabel.Text = citiesChosen;
    ActiveForm = SecondForm;
}
</script>
<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form Runat="server" ID="FirstForm" Wrapping="Nowrap">
        <mobile:Label Runat="server" ID="Cities"
            StyleReference="title" Text="Choose One or More Cities" />

        <mobile:SelectionList Runat="server" ID="CityList" SelectType="CheckBox">
            <Item Text="Sydney" Value="Sydney" />
            <Item Text="Paris" Value="Paris" />
            <Item Text="New York" Value="New York" />
            <Item Text="Madrid" Value="Madrid" />
            <Item Text="Tokyo" Value="Tokyo" />
        </mobile:SelectionList>
        <mobile:Command Runat="server" ID="Go" OnClick="CityForm_OnSubmit" Text="Go" />
    </mobile:Form>
    <mobile:Form ID="SecondForm" Runat="server">
        <b>Cities chosen: </b>
        <mobile:Label Runat="server" ID="CityLabel" />
    </mobile:Form>
</body>
</html>
```

Selection List Types

Similar to a **List** control, the **SelectionList** control offers several rendering styles for devices that support these rendering styles. The rendering style, found through the **SelectType** property, also affects whether or not the control uses single or multiple selection. Valid settings for this property come from the **ListSelectType** enumeration, and include the following.

Type (ListSelectType enumeration)	Selection type	Description
DropDown	Single	The list is shown as a drop-down list box.
ListBox	Single	The list is shown as a list box.
Radio	Single	The list is shown as a set of options (radio buttons).
MultiSelectListBox	Multiple	The list is shown as a list box.
CheckBox	Multiple	The list is shown as a set of check boxes.

The following sample allows you to see all of these supported rendering styles interactively.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>

<script runat="server">
    protected void ListStyle_OnSelect(Object sender, ListCommandEventArgs e)
    {
        String text = e.ListItem.Text;
        CityList.SelectType =
            (ListSelectType)Enum.Parse(typeof(ListSelectType), text, true);
        CityFormLabel.Text = String.Format("SelectType = {0}", text);
        ActiveForm = CityForm;
    }
    protected void CityForm_OnSubmit(Object sender, EventArgs e)
    {
        ActiveForm = ChoiceForm;
    }
}
```

```
</script>
<html xmlns="http://www.w3.org/xhtml">
<body>
  <mobile:Form ID="ChoiceForm" Runat="server">
    <mobile:Label Runat="server" ID="Types" StyleReference="title"
      Text="Selection List Types" />
    <mobile:List Runat="server" ID="ListTypes" OnItemCommand="ListStyle_OnSelect">
      <Item Text="DropDown" />
      <Item Text="ListBox" />
      <Item Text="Radio" />
      <Item Text="MultiSelectListBox" />
      <Item Text="CheckBox" />
    </mobile:List>
  </mobile:Form>
  <mobile:Form ID="CityForm" Runat="server" Wrapping="NoWrap">
    <mobile:Label Runat="server" ID="CityFormLabel" StyleReference="title" />
    <mobile:SelectionList Runat="server" ID="CityList" SelectType="CheckBox">
      <Item Text="Sydney" />
      <Item Text="Paris" />
      <Item Text="New York" />
      <Item Text="Madrid" />
      <Item Text="Tokyo" />
    </mobile:SelectionList>
    <mobile:Command Runat="server" ID="BackCommand"
      OnClick="CityForm_OnSubmit" Text="Back" />
  </mobile:Form>
</body>
</html>
```

Validating Input

Microsoft ASP.NET provides a powerful validation mechanism that you can use to easily **check input forms for errors** and, if necessary, display messages to the user. Mobile Web Forms provides mobile controls that correspond to each of the validation controls available in ordinary Web Forms pages. Each of these controls **wraps the corresponding Web Forms validator** and provides an identical object model to its peer. Thus, if you are familiar with the way validation server controls work in Web Forms, you are already familiar with how to work with validation in Mobile Web Forms pages.

Mobile validation controls work with a limited subset of mobile controls. For each control, a specific property contains the value to be validated. The following table shows the input controls that may be validated.

Control	Validation property
TextBox	Text
SelectionList	SelectedIndex

The following sample demonstrates how to use the mobile equivalent of the **CompareValidator** control on your mobile Web Forms page. The sample provides a user with a text box. If you enter a number greater than 100, the validator will display an error message.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>
<script runat="server" >
    protected void Submit_OnClick(Object sender, EventArgs e)
    {
        //This.Validate();
        if (Page.IsValid)
        {
            ActiveForm = Form2;
            Result.Text = String.Format("The number you chose was {0}",
                                      NumberEdit.Text);
        }
    }
</script>
<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form ID="Form1" Runat="server">
        <mobile:CompareValidator
            Runat="server" id="Comparer"
            ControlToValidate="NumberEdit"
            ControlToCompare="OriginalEdit"
            ValueToCompare="100"
            Type="Integer"
            Operator="LessThanEqual"
            ErrorMessage="There is an error">
                The number is out of range
        </mobile:CompareValidator>
        <mobile:Label Runat="server" ID="ErrorMessage">
            Please enter an integer less than or equal to 100
        </mobile:Label>
        <mobile:TextBox ID="OriginalEdit" Runat="server" Numeric="True">
        </mobile:TextBox>
        <mobile:TextBox ID="NumberEdit" Numeric="true" Runat="server" />
        <mobile:Command Runat="server" ID="SubmitCommand">
```

```

        OnClick="Submit_OnClick"
        CausesValidation="true"
    >
        Submit
    </mobile:Command>
</mobile:Form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Result" Runat="server" />
    <mobile:Link Text="Back" NavigateUrl="#Form1" Runat="server" id="GoBack">
        Go Back to Data Entry Form
    </mobile:Link>
</mobile:Form>
</body>
</html>

```

Note that for validation to occur on a form, the control that causes the form to be submitted must have its **CausesValidation** property set to **true**. The only current control that this applies to is the **Command** control, for which the **CausesValidation** property is **true** by default.

Using a Validation Summary

Instead of showing each validation error separately, you can use the **ValidationSummary** mobile control to display a summary of validation errors. To use a **ValidationSummary** mobile control, create a separate form on your page, place the control on it, and set its **FormToValidate** property to the form that you want to validate. Similar to its Web Forms equivalent, the **ValidationSummary** mobile control polls each of the validation controls on the given form and displays a summary of the results.

The following sample shows an example of the **ValidationSummary** mobile control. The text box has three associated validators, and the **ValidationSummary** control shows a summary of validation errors in the input.

```

<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<%@ Register Assembly="System.Web.Mobile" Namespace="System.Web.UI.MobileControls"
    TagPrefix="mobile" %>
<script runat="server">
    protected void Summary_OnClick(Object sender, EventArgs e)
    {
        ActiveForm = Form2;
    }

```

```
void ServerValidate(object source, ServerValidateEventArgs args)
{
    int num;
    try
    {
        num = Int32.Parse(args.Value);
    }
    catch (FormatException)
    {
        args.IsValid = false;
        return;
    }
    if (num % 2 == 0)
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
</script>
<html xmlns="http://www.w3.org/xhtml">
<body>
    <mobile:Form ID="Form1" Runat="server">
        <mobile:CustomValidator Runat="server" ID="CustomValidator"
            ControlToValidate="number"
            OnServerValidate="ServerValidate"
            ErrorMessage="The input number should be even.">
            Please enter an even number
        </mobile:CustomValidator>
        <mobile:RangeValidator Runat="server" ID="RangeValidator"
            ControlToValidate="number"
            Type="Integer"
            MaximumValue="50"
            MinimumValue="1"
            ErrorMessage="The input number should be with in the range 1 - 50.">
            Please enter a number of value 1 - 50
        </mobile:RangeValidator>
        <mobile:RequiredFieldValidator Runat="server" id="Required"
            ControlToValidate="number"
            ErrorMessage="The input box cannot be empty.">
            Please fill in a number
        </mobile:RequiredFieldValidator>
        <mobile:Label Runat="server" ID="Label">
            Please enter an even number between 1 and 50
        </mobile:Label>
        <mobile:TextBox ID="number" Numeric="true" Runat="server" />
        <mobile:Command OnClick="Summary_OnClick" Runat="server">
```

```
        ID="Summary">
            Submit
        </mobile:Command>
    </mobile:Form>
    <mobile:Form ID="Form2" Runat="server">
        <mobile:Label Runat="server" ID="InfoLabel">
            Your input is valid if no error is shown below.
        </mobile:Label>
        <mobile:ValidationSummary
            Runat="server" ID="ErrorSummary"
            FormToValidate="Form1"
            HeaderText="Error listed below:"
            BackLabel="Return to Entry Form"/>
    </mobile:Form>
</body>
</html>
```

Finish