When developing complex form controls consisting of multiple input elements, you can use values in XML format to save the resulting data into form fields. XML allows you to store structured data that can be processed by APIs and is also readable by humans.

Kentico provides the following recommendations and API methods to help you correctly create form controls with XML values:

- When adding XML elements manually (for example via the XmlDocument.CreateElement, XmlElement.AppendChild or PrependChild methods), make sure that the ***element order is always consistent*** in the resulting XML document.
- To add attributes to elements, call the **XmlElement.AddAttributes** extension method, which ensures that all attributes are always in alphabetical order (based on the attribute names).
- When creating child elements based on dynamic data, prepare an *IDictionary* collection containing the required values, and call the **XmlElement.AddChildElements** extension method. The method ensures that the child elements are ordered alphabetically based on their element names.
- Call the **XmlDocument.ToFormattedXmlString** extension method on the resulting XML document to convert the XML data to a string that can be saved as the form control's value. The method provides whitespace formatting, including line breaking and indentation for each XML element.
- Once the form control is developed and registered in the system, use the **Long text** data type for form fields where you plan to use the control.

**Note**: The *XmlElement.AddAttributes*, *XmlElement.AddChildElements* and *XmlDocument.ToFormattedXmlString* extension methods are provided within the **CMS.Helpers** library.

The steps above ensure that the XML data is formatted correctly and has consistent order. Consistency is important to allow easy comparison and change tracking in object data, particularly when objects containing fields with values provided by the form control are serialized (for example during Export, Staging or Object versioning).

**Other data formats**

If your form control saves its value in a different structured data format than XML (for example JSON), you need to ensure the consistency of the data manually.

## Example

The following example demonstrates how to create a form control that consists of multiple internal controls and saves the resulting data into an XML value. The sample control allows users to:

- Select an animal type from a list of predefined options
- Enter a species name
- Mark continents that the animal inhabits

**Form control implementation options**

This example implements the custom form control as a Web user control (.ascx file) within the Kentico web project. You can alternatively create form controls by adding a control class to a separate assembly (the class must also inherit from the *FormEngineUserControl* base class).

**Note**: For the sake of simplicity, the example only provides the most basic implementation (without sufficient input validation, error handling, etc.).

1. Open your web project in Visual Studio (using the **WebSite.sln** or **WebApp.sln** file).
2. Right-click the **CMSFormControls** folder and choose **Add -> Add New Item**.

3. Create a new **Web User Control**, for example named *CustomAnimals.ascx*.
4. Add the following input controls into the user control's markup:

```
<asp:DropDownList ID="drpAnimalType" runat="server"></asp:DropDownList>
<asp:TextBox ID="txtSpecies" runat="server"></asp:TextBox>
<asp:CheckBoxList ID="checkContinents" runat="server"></asp:CheckBoxList>
```

5. Switch to the code behind, add the following *using* statements, and adjust the class declaration so that the control inherits from **FormEngineUserControl**:

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Xml;
using System.Web.UI;
using System.Web.UI.WebControls;

using CMS.FormEngine.Web.UI;
using CMS.Helpers;

public partial class CMSFormControls_CustomAnimals : FormEngineUserControl
{
}
```

6. Add the following members into the class:

```
/// <summary>
/// Gets or sets the value of the field, in this case XML data stored in string
format.
/// </summary>
public override object Value
{
        get
        {
                // Gets XML data in string format based on the values of the
internal controls
                return GetXmlString();
        }
        set
        {
                // Gets the field's value as a string
                string strValue = ValidationHelper.GetString(value, string.Empty);

                // Sets up the drop-down and checkbox items of the control
(occurs for new forms when attempting to set the default field value)
                EnsureItems();

                if (!string.IsNullOrEmpty(strValue))
                {
                        // Creates an XmlDocument based on the field's value
                        XmlDocument xmlValue = new XmlDocument();
                        xmlValue.LoadXml(strValue);

                        // Sets values for the internal controls based on the XML
data
                        SetValuesFromXml(xmlValue);
                }
        }
```

```
}

/// <summary>
/// Sets up the items of the internal drop-down list and checkboxes.
/// </summary>
private void EnsureItems()
{
        if (drpAnimalType.Items.Count == 0)
        {
                drpAnimalType.Items.Add(new ListItem("Birds", "Birds"));
                drpAnimalType.Items.Add(new ListItem("Bees", "Bees"));
                drpAnimalType.Items.Add(new ListItem("Marsupials", "Marsupials"));
        }

        if (checkContinents.Items.Count == 0)
        {
                checkContinents.Items.Add(new ListItem("Africa"));
                checkContinents.Items.Add(new ListItem("North America"));
                checkContinents.Items.Add(new ListItem("South America"));
                checkContinents.Items.Add(new ListItem("Asia"));
                checkContinents.Items.Add(new ListItem("Europe"));
                checkContinents.Items.Add(new ListItem("Australia"));
                checkContinents.Items.Add(new ListItem("Antarctica"));
        }
}

/// <summary>
/// Returns a string containing XML data based on the values of the internal
controls.
/// </summary>
private string GetXmlString()
{
        // Creates a new XML document
        XmlDocument xmlDoc = new XmlDocument();

        // Creates an animal XML element
        XmlElement xmlAnimal = xmlDoc.CreateElement("animal");

        // Prepares attributes for the animal element
        var animalAttributes = new Dictionary<string, string>
        {
                {"type", drpAnimalType.SelectedValue},
                {"species", txtSpecies.Text}
        };

        // Adds the attributes to the animal element (ensures consistent
alphabetical order)
        xmlAnimal.AddAttributes(animalAttributes);

        // Adds the animal element as the root of the XML document
        xmlDoc.AppendChild(xmlAnimal);

        // Prepares a dictionary of child elements
        var continentElements = new Dictionary<string, bool>();
        foreach (ListItem item in checkContinents.Items)
        {
                continentElements.Add(item.Text.Replace(" ", string.Empty), item.
Selected);
        }
```

```
        // Creates child elements under the animal element
        xmlAnimal.AddChildElements(continentElements);

        // Ensures line breaks and indentation formatting for the XML text, omits
the XML declaration
        return xmlDoc.ToFormattedXmlString(true);
}

/// <summary>
/// Sets up the internal controls based on data from an XmlDocument.
/// </summary>
private void SetValuesFromXml(XmlDocument xmlValue)
{
        // Sets the animal type value based on the type attribute of the animal
element
        drpAnimalType.SelectedValue = System.Convert.ToString(xmlValue["animal"].
Attributes["type"].Value);

        // Sets the species name based on the species attribute of the animal
element (if it has a value)
        if (xmlValue["animal"].HasAttribute("species"))
        {
                txtSpecies.Text = System.Convert.ToString(xmlValue["animal"].
Attributes["species"].Value);
        }

        // Sets the continent checkboxes based on the child elements under the
animal element
        foreach (ListItem item in checkContinents.Items)
        {
                item.Selected = System.Convert.ToBoolean(xmlValue["animal"][item.
Text.Replace(" ", string.Empty)].InnerText);
        }
}
```

7. Save the control's files. **Build** your project if it is installed as a web application.

To register the form control in the system:

1. Sign in to the Kentico administration interface.
2. Open the **Form controls** application.
3. Click **New form control**.
4. Enter the following values:
   - **Control source**: Web user control
   - **Display name**: Animals XML
   - **Code name**: Leave the *(automatic)* value
   - **File name**: ~/CMSFormControls/CustomAnimals.ascx (you can click **Select** to choose the file)
5. Click **Save**.
6. On the **General** tab, set the **Control scope**:
   - Use control for: **Long text**
   - Show control in: Any required options
7. Click **Save**.

You can now assign the form control to form fields with the *Long text* data type.


The control saves the user input into XML data, and ensures consistent alphabetical order of attributes and child elements. For example, a field using the *Animals XML* control would have the following value for the input shown in the screenshot above:

```
<animal species="Swallow" type="Birds">
  <Africa>True</Africa>
  <Antarctica>False</Antarctica>
  <Asia>True</Asia>
  <Australia>True</Australia>
  <Europe>True</Europe>
  <NorthAmerica>True</NorthAmerica>
  <SouthAmerica>True</SouthAmerica>
</animal>
```