

Expanding Data Display Capabilities with Custom Cell Types

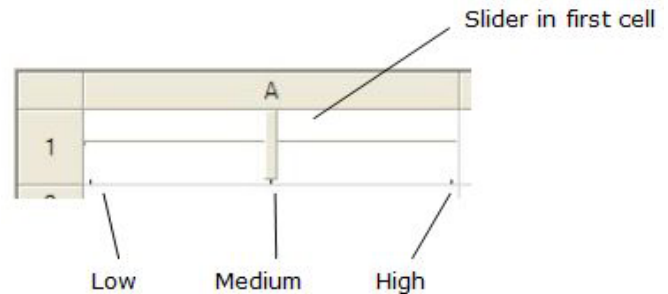
by Bobby Orndorff, Technical Contributor

Introduction

Spreadsheet controls use cell type classes to edit and display cell values. In Spread for Windows Forms, there are over a dozen built-in cell type classes including button, check box, combo box, currency, date/time, number, percent, slider, text, and more. Each of these built-in cell type classes comes with numerous properties to allow such actions as restricting the range of values while editing, formatting number values while displaying, and so on. But built-in cell types are only the beginning. To handle the increasingly complex applications that you must develop and the growing number of ways to display data, FarPoint has made it easy to customize cell types by allowing you to either extend one of the built-in cell types or to create your own cell type class from scratch. You can create custom cell types as one way of extending the power and flexibility of Spread for Windows Forms in order to handle data display challenges.

Custom Cell Type

You can create your own cell type to allow custom handling of data. For example, suppose your application has a requirement to edit and display values from an enumeration similar to the `TrackBar` control with these three values: `Low`, `Medium`, and `High`. The built-in slider cell type in Spread for Windows Forms comes very close to meeting this requirement. The only problem is that the built-in slider cell type edits and displays integer values, not strings. To work around this limitation of the built-in cell type, you could create a custom slider cell type that extends the built-in slider cell type. The custom slider cell type would provide a means of translating between enumeration values and integer values. The custom slider cell type would then hand off most of the work to the built-in cell type.



Note that cell values are passed between the spreadsheet control and the cell type class as objects. To avoid unnecessary boxing and unboxing operations it might be useful for the custom slider cell type class to hold onto boxed versions of the enumeration values and corresponding integer values. This is shown in the example C# code below.

[C#]

```
// boxed MyEnum values
private static object lowBoxed =
MyEnum.Low;
private static object mediumBoxed =
MyEnum.Medium;
private static object highBoxed =
MyEnum.High;
// boxed integer values
private static object zeroBoxed = 0;
private static object oneBoxed = 1;
private static object twoBoxed = 2;
```

The custom slider cell type would need a means of converting between boxed enumeration values and boxed integer values. This is shown in the example C# code that follows.

[C#]

```
// converts boxed MyEnum value to boxed
integer value
private static object FromMyEnum(object
value)
{
    if (value is MyEnum)
    {
        switch ((MyEnum)value)
        {
            case MyEnum.Low: return zeroBoxed;
            case MyEnum.Medium: return oneBoxed;
            case MyEnum.High: return twoBoxed;
        }
    }
    return value;
}
```

```
// converts boxed integer value to boxed
MyEnum value
private static object
    ToMyEnum(object value)
{
    if (value is int)
    {
        switch ((int)value)
        {
            case 0: return lowBoxed;
            case 1: return mediumBoxed;
            case 2: return highBoxed;
        }
    }
    return value;
}
```

Once you have the conversion methods in place, it is simply a matter of overriding the cell type methods that deal directly with cell values. This includes the `GetEditorValue`, `SetEditorValue`, `IsReservedLocation`, `IsValid`, `GetPreferredSize`, `PaintCell`, `Format`, and `Parse` methods.

For overridden methods that take a cell value, the method would call the `FromMyEnum` conversion method and then call the base method.

[C#]

```
// put value into cell editor control
public override void
SetEditorValue(object value)
{
    base.SetEditorValue(FromMyEnum(value));
}
```

For overridden methods that return a cell value, the method would call the base method and then call the `ToMyEnum` conversion method.

[C#]

```
// get value out of cell editor control
public override object GetEditorValue()
{
    return ToMyEnum(base.GetEditorValue());
}
```

That's about all there is to it.

Conclusion

Using the above techniques, you could create other custom cell types that work with other sets of values. Refer to the sample project on our Web site to get the complete source code for the custom slider cell type.

This is just a brief introduction to the customizations that are possible with the custom cell types in Spread. For more information, read the chapter on cell types in the *Developer's Guide* and keep an eye on the online forum where questions are answered about a range of customer issues. Find us online at www.fpoint.com.