

## Super Models: Introducing the Underlying Data Model

by Bobby Orndorff, Sr. Software Architect

### Introduction

Spread .NET products make full use of the object-based nature of .NET by providing underlying models that give you the ability to extend Spread and customize it for your application. The data model is the backbone of the component, and is the most powerful of the models. This article introduces the use of the data model in Spread for .NET products.

### What's a Data Model

There are several underlying models in Spread. The data model in Spread is the object that stores all the data in the sheet, and provides all of the data-related core functionality. This data includes all values in the cells, notes (or comments) in cells, and application-defined tag values for the cells. It is also the object that controls the dimensions of the sheet by determining the number of columns and rows. When rows or columns are inserted or removed, or when the RowCount or ColumnCount is changed directly, it is the data model where these changes occur first, and then the other sheet models (style, span, and selection) are updated with the new dimensions.

### Interfaces in the Data Model

The core interface for a data model, ISheetDataModel, defines the properties, methods and events which are required of all data model objects for the above described functionality.

The data model also supplies the data binding, calculated formulas, and hierarchical data support for the sheet, although none of those is required of the data model. Each is optionally implemented through additional interfaces which may be defined on the data model. Support for data binding requires implementation of the

IDataSourceSupport interface, and if unbound rows in the sheet are required, you can also implement the IUnboundRowSupport interface. Support for calculation requires implementation of the IExpressionSupport interface, and if more advanced calculation support is required, you can also implement the ICalculationSupport interface for automatic and manual recalculation of formulas, the IIterationSupport interface for formulas which make recursive circular references between two or more cells, the ICustomFunctionSupport interface for application-defined formula functions, and the ICustomNameSupport interface for application-defined named expressions which can be used in formula expressions. Support for hierarchical data requires implementation of the IChildModelSupport interface, which creates parent-child relationships between two or more data model objects.

The data model can also implement the IArraySupport interface for getting and setting arrays of values in the model, which can be more efficient than looping through each cell. The IRangeSupport interface allows for inserting and removing columns and rows, and for range operations such as copying, moving, or swapping the contents of one range of cells to another range of cells, or clearing the contents of a range of cells. For XML serialization, the data model can also implement the ISerializationSupport, ICanSerializeXml, and ISerializeData interfaces.

These interfaces are implemented in the DefaultSheetDataModel class, which is the default implementation of the data model object that is initially created by each sheet. If you simply wish to use the default implementation then it is not necessary to create a DefaultSheetDataModel and assign it to a sheet, since there already is one there.

## Custom Data Model

While the `DefaultSheetDataModel` is designed to fulfill the requirements of a wide range of applications, there may be some scenarios where a custom data model would work better. For example, suppose the application has a very large table of read only values where each individual value can be quickly calculated. It may be more efficient to calculate the values on the fly as they are displayed, rather than calculating and storing all the values ahead of time. The following code implements a custom data model that consists of one million rows by one million columns where the values are calculated on the fly. The code takes advantage of the `BaseSheetDataModel` class, which is an abstract class that provides default implementations of many of the methods in the `ISheetDataModel` interface.

```
class MyDataModel : BaseSheetDataModel
{
    public override int RowCount
    {
        get { return 1000000; }
    }
    public override int ColumnCount
    {
        get { return 1000000; }
    }
    public override object GetValue(int row, int
column)
    {
        return row + column;
    }
}
```

This custom data model could then be assigned to the sheet using the following code.

```
spread.Sheets[0].Models.Data = new
MyDataModel();
```

## Conclusion

This is just a brief introduction to the models in Spread in general, and the data model in particular. For more information, read the chapter in the Developer's Guide and keep an eye out for a follow-up article on the other models.

While not everyone uses models in their application, the object-oriented development of Spread and the flexibility that the underlying models afford allow you to customize and extend Spread in so many ways. For more information, refer to product documentation or contact FarPoint. Find us online at [www.fpoint.com](http://www.fpoint.com).

"This is an excellent product, well worth the money spent. Your developers have made it possible for us to produce an excellent system in a time frame which far exceeded our initial goals. In no way would we have been able to develop a grid which so perfectly suited our needs with the same cost."

*-Guy Munton Jackson, Software Developer*