**Microsoft Office SharePoint Server (MOSS) for Developers**

**(Part 1)**

**Topic 2: Data Lists for Developers**

[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

**Introduction**

A SharePoint data list is somewhat analogous to a database table, from the standpoint of a list has a set of columns, just like a database table has a set of columns; has a set of rows, i.e. the items in the list, just like a database table has items in a list. However unlike a database table a data list starts to break down provides more functionality than just rows and columns.

For instance, a list inherently has a set of forms that are associated with it for entering new data and for displaying data and for changing data. Those are inherent to a list in SharePoint. The user interface is already built in.

It also has a view to look at the data in the list. In a traditional database table, if you think about that, you don't get any of the user interface experience. One of the great things about SharePoint and using lists for your data sources is a lot of that user interface you'd have to build for yourself is already built.

In addition, the schema is changeable by the user. For example if need to give the user enough permission to make a change to the schema of the list, they can do that on their own.

Another thing that's very valuable with the way that SharePoint manages list data is you have these Web parts, the list view Web part and the data view Web part. The list view is the one that you'll see in a Web part as you go in to add a Web part; it's the one that's up at the top that matches all your lists.

Another important feature is the notion of list types. List types are kind of this prebuilt schema and provide prebuilt functionality. When you look at a list instance, that instance was created based on a “template” described in SharePoint from that type. One way I like to think about it is it's a rubber stamp. A the list type is a rubber stamp that maybe has an approval block on it, and you put it on a form and you stamp it on, and it leaves behind its mark. If you add new fields to the template; lists created from that template will inherit the new fields.

We can manage a list after we've created its instance from a template. We create a template, we get a bunch of fields, we get a bunch of default behaviors, and then we can go and change that: We can remove a field, we can add a field, we can change the forms if we've got SharePoint client.

Lists also support attachments, which is important. So, it's not just a row of data but it's a row of data that can support multiple attachments. So, if I want to add to this vacation request content type or this list item, if I add a document, which is somebody's signed signature that, yes, I can go on vacation and they're paying for me to go to Disneyworld or whatever it is, I can do that, because I've got the ability to add attachments to a list. So, it's another place where although you can think about lists as kind of the database table, it doesn't quite work out that way in terms of there's a lot of the advanced functionality that you don't get with a database table that you do get with a SharePoint list.

Another type of list is called document libraries. From a code perspective a document library is a simply a special type of list that always has at least one document content type.

One advantage document libraries offer is integration with the Office client, all of your Office tools, your Word and Excel and PowerPoint etc, know how to talk directly to these document libraries. You can do File, Save As and go directly into a SharePoint document library. That integration is great for building your applications. You don't have to build a way for users to upload documents into the system; all of that is infrastructure that's built in.

Document libraries also support versions, just like any other list. The document libraries do, by default, put their data into the SharePoint content database, and there is one content database for the site collection contains all the documents. If you upload a Word document, it becomes a record in the database with a binary large object, your file, attached to it.

**Folders in Lists**

Folders—one of the popular features from document libraries in Windows SharePoint Services 2.0—is now also available in lists. Folders allow more segregation of content within a single list, adding another level of versatility.

A new feature of folders is the ability to assign metadata to a specific folder. This creates a rich content holder that contains child content, while serving as more than a container. Essentially, this allows the folder to function as a separate, but fully functional, item type. An example of this can be seen in Windows SharePoint Services discussion boards. Each top-level discussion is a folder with metadata, containing one to *n* child objects. By modifying the appearance of the folder, a user can visualize the folder as another list item rather than as a container. You can use this model in custom lists to create a dynamic parent/child list.

Many business applications require rich container objects, for which a parent object includes both important data concerning itself, and one to *n* children with similar data. Normally, you handle this challenge by using lists in which a parent—or master—list selection drives the display of a child—or detail—list. Because you can now assign metadata to folders in a SharePoint list, you can achieve similar functionality with a single list.

For example, consider an Orders list. You can provision a folder as an order, and then assign metadata to it, such as *order number*, *customer*, and *shipment* *method*. By doing so, you can make the folder mimic the master list functionality. You can then fill the folder with *order* *items* list items, each with its own metadata, such as *item* *number*, *description*, and *price*, thus providing the rich functionality of a master/detail list within a single list.

**List Indexing**

Integral to Windows SharePoint Services as a platform is the ability to use its storage containers, specifically lists and document libraries, for application storage. Windows SharePoint Services 3.0 expands the capacity to use lists and libraries in this way by providing a rich storage model with performance capable of supporting applications, both external and those built upon the platform. One of the key ways that Windows SharePoint Services 3.0 enhances its storage containers is to include indexing on columns. By allowing a column to be indexed and its data stored as a simple name-value pair in a separate database, access to specific items in large lists is significantly improved.

List indexing allows SharePoint lists to become true data stores, capable of supporting external applications as well as simple team sites. For example, in the the Orders list previously described, indexing the *order number* column significantly improves access to specific items in the list in external requests. In the case of an external application with hundreds of thousands of records, indexing on columns can greatly improve performance and make it easier to use a SharePoint list as a storage device. Standard features of the platform, such as views, alerts, and RSS feeds, can be used as a data store explorer for internal use in simple team sites.

**Cross-List Queries**

Another improvement made with the idea of using Windows SharePoint Services storage more effectively is cross-list queries. Cross-list queries allow you to use the [SPQuery](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spquery.aspx) object to query all the lists within a Web site or site collection. In Windows SharePoint Services 2.0 you had to iterate through the parent objects to obtain a collection of list objects, query the lists to return the items, and then build your own collection of list items from the multiple lists. The [SPSiteDataQuery](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spsitedataquery.aspx) class allows you to perform cross-list queries quickly and efficiently, providing a rich querying capability, similar to SQL queries, within Windows SharePoint Services.

**List Item Versions**

Several improvements in Windows SharePoint Services 3.0 are implemented on the item level rather than on the entire list. Following is a list of several important new features.

* Per-Item Security   Lists and document libraries provide item-level security and the ability to set roles on each item, rather than simply at list level. If you combine per-item security with other features, such as list events or extensible field types, you can use per-item security to change who has access to an item, based on an entry in another field. For example, a change to the status of an item could trigger a change in who has the ability to edit the item.
* Versioning   In Windows SharePoint Services 2.0, only document libraries allowed versioning, and then only a major version would be created. This basic functionality did not match the workflow normally associated with documents, namely, the creation and edit process when a document is in draft version, and then the publishing of a document to public status.

Windows SharePoint Services 3.0 offers a new, richer versioning model. This model includes both major and minor versions, and versioning within lists as well as document libraries. In addition, major and minor versions can have different security, allowing separate draft and public version workflows. Versions are associated with a rich event model that allows numerous scenarios, including custom workflows, item validation, and information propagation.

* Required Checkout   In Windows SharePoint Services 2.0, users could modify documents without checking those documents out from a document library, which allowed multiple users to edit the same document and left a complicated merge/overwrite scenario. In Windows SharePoint Services 3.0, document editing is allowed only when the document has been checked out. This forced check-out defines a clear owner model and simplifies

**Working with List Objects and Collections**

To perform actions on list data in a SharePoint Web site, you must first obtain an [SPWeb](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.aspx) object to serve as an entry point to the object model, allowing you to access lists, items, documents, users, alerts, etc. For information about how to return SharePoint Web sites, see [Getting References to Sites, Web Applications, and other Key Objects](http://msdn.microsoft.com/en-us/library/ms468609.aspx).

# **Objects**

Most classes in the [Microsoft.SharePoint](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.aspx) and [Microsoft.SharePoint.Administration](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.aspx) namespaces start with **SP**. Generally, classes in the **Microsoft.SharePoint** assembly that don't start with this prefix represent Web form controls.

Windows SharePoint Services typically does not save modifications of object properties to the database until you call the **Update** method on the given object. The following example shows how to change the title and description for the Tasks list.

# **Collections**

Just as lists are at the center of a SharePoint site, so are collections at the center of its object models. You can use each collection to add, delete, enumerate, and update a type of object. Collection classes generally share the following traits:

* Has a name that ends in "Collection."
* Implements the **System.Collections.ICollection** interface.
* Has a **Count** property of type Int32.
* Has an Int32 indexer that can be used to get the nth item in the collection.
* Has an indexer that takes an item identifier.
* Has **Add** and **Delete** methods.

Calling the **Add** method for a collection usually updates the database on the back-end server with the appropriate data, except when additional information is required in order to update data. In this case, using the **Add** method returns an object that you can use to gather the information. For example, to add an item to a list, first use the [Add](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splistitemcollection.add.aspx) method of the [Microsoft.SharePoint.SPListItemCollection](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splistitemcollection.aspx) class to return an [SPListItem](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splistitem.aspx) object, assign values to appropriate properties of the object, and then call the Updatemethod to effect changes within the content database.

# **Indexers**

Indexers provide a useful means to access individual items in collections. To return an item, use square brackets (**[]**) in C# or parentheses (**()**) in Visual Basic to contain either an index or a string that identifies the item within the collection.

For example, if *mySite* represents an instance of the **SPWeb** class, then SPList myList = mySite.Lists["Announcements"] returns the Announcements list in C#, while Dim myList As SPList = mySite.Lists("Announcements") returns the same list in Visual Basic. You can then use the [Items](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splist.items.aspx) property for the list object to return all the list items in the list (in C#, SPListItemCollection myItems = myList.Items, or in Visual Basic, Dim myItems As SPListItemCollection = myList.Items). To return only a subset of items from the list, you can call the list object's GetItems method and pass an [SPQuery](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spquery.aspx) object to specify the subset: SPListItemCollection myItems = myList.GetItems(myQuery) (in Visual Basic, Dim myItems As SPListItemCollection = myList.GetItems(myQuery)).

You can use an indexer to return a specific field from a list (for example, myList.Items["Field\_Name"] in C#, or myList.Items("Field\_Name") in Visual Basic). You can also specify a field name in an indexer and iterate through the collection of items in a list in order to return values from the field.