DHTML Object Model Support for Data Binding

In Windows Internet Explorer, elements are exposed to scripts as objects. The attributes supported by those elements are exposed as properties. Just as a user can perform actions on most elements clicking a button, for example scripts can call methods of the corresponding object. Scripts can also customize the behavior of elements by handling the events exposed by the corresponding object.

Web authors can achieve dramatic effects on their databound pages by adding simple scripts that manipulate the properties, methods, and events exposed by the DHTML Object Model. The DHTML Object Model supports the following capabilities at run time.

This section discusses how to use these features. Because the event model exposed by the DHTML Object Model for data binding is so extensive, it is treated in DHTML Event Model Support for Data Binding.

Modifying the Binding of Elements to Data

The dataSrc and dataFld attributes, introduced in About Data Binding Architecture, allow an author to bind an HTML element to data. The DHTML Object Model exposes these attributes as the **dataSrc** and **dataFld** properties, and their values can be modified at any time through scripts.

Web authors might want to perform the following operations on elements that support data binding:

- Add a binding to an unbound element
- Remove a binding from a currently bound element
- Change the current binding of an element

The following sections assume that the element is not serving as a template element in a repeated table scenario. Dynamic tabular binding is discussed below.

Adding a Binding to a Single-Valued Element at Run Time

Given an element that supports data binding, such as a span or textArea, adding a binding is as simple as setting the **dataSrc** and **dataFld** properties of that element. For example, to bind a span with ID span1 to the first name field in a data set provided by a data source object (DSO) with ID dsoComposer, use the following script:

```
span1.dataSrc = "#dsoComposer";
span1.dataFld = "compsr_first";
```

The same binding can be accomplished at design time using the following HTML code:

```
<SPAN DATASRC="#dsoComposer" DATAFLD="compsr_first"></SPAN>
```

In both cases compsr_first is the name of an actual field in the data set. The binding does not occur until both properties are set, and any prior value stored by the element is lost. If the value is required at a later

time, for example when the binding is removed, the page author must cache the value prior to binding the element.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbaddsv.htm

Removing the Binding from a Bound Element

To remove the binding from a currently bound element, simply set the **dataSrc** and **dataFld** properties to the empty string. For example:

```
span1.dataSrc = "";
span1.dataFld = "";
```

The previous Show Me button demonstrates this behavior.

Modifying the Binding of a Single-Valued Element at Run Time

Modifying an existing binding is similar to **binding an unbound element**. Two cases are described.

To bind an element to another field in the same data set, modify the **dataFld** property. For example, given a **span** with ID span1 that is currently bound to a composer's first name, the following code binds the **span** to the composer's last name.

```
span1.dataFld = "compsr_last";
```

To bind an element to a field in a different data set, set the value of the **dataSrc** property to the DSO that provides the data. Next, set the value of the **dataFld** property for the element, if the name of the field in the data set provided by the DSO is different from the previous DSO.

Binding a Table at Run Time

The section on **Binding HTML Elements to Data** explained how to **bind a table to data at design time**. Binding a table at run time requires three steps:

1. If the table is already bound to a data source, it must be unbound by setting the **dataSrc** property of the corresponding table object to the empty string.
2. Modify the bindings of the template elements contained by the table by setting the **dataFld** property of the corresponding objects.
3. Bind the table to the appropriate data source by setting the **dataSrc** property of the corresponding table object.

Note that modifying the bindings of any of the individual elements contained within a table requires that the entire table be unbound and then rebound to the DSO. This is necessary because while the table is bound, changes to elements inside the table are applied to the generated rows. Changes while the table is unbound affect the template. Click the Show Me button to see an example that modifies the binding at run time.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbdyntab.htm

```
<SCRIPT FOR=cboField EVENT=onchange>
    cDSO = tblComposers.dataSrc; // remember the DSO
    tblComposers.dataSrc = ""; // unbind the table
    // Set the binding for the contained element.
    spanField.dataFld = this.options(this.selectedIndex).value;
    tblComposers.dataSrc = cDSO; // rebind the table
</SCRIPT>

<TABLE ID=tblComposers DATASRC=#dsoComposers>
    <THEAD><TR STYLE="font-weight:bold">
        <TD>Last Name</TD>
        <TD><SELECT ID=cboField>
            <OPTION VALUE=compsr_first SELECTED>First Name</OPTION>
            <OPTION VALUE=compsr_last>Last Name</OPTION>
            <OPTION VALUE=compsr_birth>Date of Birth</OPTION>
            <OPTION VALUE=compsr_death>Date of Death</OPTION>
            <OPTION VALUE=origin>Origin
            </SELECT>
        </TD>
    </TR></THEAD>
    <TR>
        <TD><SPAN DATAFLD=compsr_Last></SPAN></TD>
        <TD><SPAN ID=spanField DATAFLD=compsr_first></SPAN></TD>
    </TR>
</TABLE>
```

This example uses the drop-down list to dynamically set the binding of the second column of the table. When a value is selected in the drop-down list, the onchange event handler stores the table's previous binding in a temporary variable. It then unbinds the table by setting its **dataSrc** property to the empty string. Finally, the code rebinds the table to the DSO by setting the **dataSrc** property to the previous value.

Dynamically Adding Bound Elements and Data Providers

In addition to modifying the bindings of existing elements on the page, the DHTML Object Model supports the dynamic addition of both databound elements and data source objects at run time. By adding these elements at run time, Web authors can reduce the initial download time of their content and can use a single page to display various sets of data to the user.

Click the Show Me button to see an example that dynamically adds a data source object and bound elements to the page.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbfly.htm

Dynamically Changing the Way Elements Render Data

The DATAFORMATAS attribute, introduced **above**, allows an author to change the way an element renders its data. The DHTML Object Model exposes the **dataFormatAs** property on **elements that support rendering** in formats other than text.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbfmtfly.htm

Dynamically Controlling the Amount of Displayed Data

The dataPageSize attribute introduced **above** allows an author to restrict the number of records displayed by a tabular data consumer. The DHTML Object Model exposes this attribute as the **dataPageSize** property, and its value can be modified at any time through scripts. In addition, the page author can control the currently viewed page of records displayed by a tabular consumer using the firstPage, lastPage, nextPage and previousPage methods.

Click the Show Me button to see how the **dataPageSize** property and accompanying methods work on a table, a tabular data consumer.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbpage.htm

Manipulating Data with Active Data Objects (ADO)

Previous sections have shown how to use HTML elements to bind and display the data provided by a DSO. Independent of the user interface, scripts can access and manipulate the data provided by a DSO in a consistent manner using the ADO recordset object. The recordset object is exposed by all data source objects through the extended recordset property.

Scripting the ADO recordset object, Web authors can add sophisticated functionality to their pages. The ADO recordset exposes a rich set of objects, which enable the following:

- Navigation

  The recordset object points to a specific record within a recordset, which is called the current record. All manipulation of recordset data is performed on the current record. Several navigation methods are provided by the ADO recordset object, enabling easy and efficient positioning of the current record. See the section on Navigating Through a Data Set for an introduction to these methods.

- Insertion, deletion, and update of records

  A comprehensive set of methods are provided by the recordset object to facilitate with practically any type of data manipulation. See the section on Adding, Deleting, and Modifying Records in a Data Set for more information.

  Note It is also possible to instantiate and populate an ADO recordset entirely using script—without any need to access an external data source.

- Set and Retrieve Properties

  The ADO recordset object exposes a Fields collection and a Properties collection. See Accessing the ADO Recordset Object Collections for some simple techniques on how to use the ADO recordset collections and their associated properties.

For complete information on ADO, see the accompanying Microsoft Active Data Objects (ADO) documentation.

Accessing the ADO Recordset Object from Script

When using ADO in conjunction with data-bound elements, there is no need to instantiate an ADO connection object because the DSO provides the **recordset** property, from which all the methods, properties, and collections of an ADO recordset are exposed.

Given a reference to a data source object, accessing the ADO recordset is easy. The following code sample assumes that dsoComposer is a valid DSO.

```
var oRecordSet = dsoComposer.recordset;
```

When using a case-sensitive scripting language such as Microsoft JScript (compatible with ECMA 262 language specification), observe that the **recordset** property is completely lowercase.

Navigating Through a Data Set

Data binding in Internet Explorer uses the notion of a current record to indicate the record in a data set to be displayed by single-valued consumers. When a data set is first loaded, the first record is typically identified as the current record. The current record is affected by any filter or sort order applied to the data. Single-valued consumers always display the current record.

To change the current record and hence the data displayed by single-valued consumers, Web authors can provide users with a set of navigation buttons like the ones shown below.

[CDATA[span " TITLE="Move to the last record"

The code behind these buttons calls the **MoveFirst**, **MovePrevious**, **MoveNext**, and **MoveLast** methods on the ADO recordset, respectively.

```
<INPUT ID=cmdNavFirst TYPE=BUTTON VALUE="<<"
    onclick="tdcComposers.recordset.MoveFirst()">
<INPUT ID=cmdNavPrev TYPE=BUTTON VALUE=" < "
    onclick="tdcComposers.recordset.MovePrevious();
    if (tdcComposers.recordset.BOF)
        tdcComposers.recordset.MoveFirst();">
<INPUT ID=cmdNavNext TYPE=BUTTON VALUE=" > "
    onclick="tdcComposers.recordset.MoveNext();
        if (tdcComposers.recordset.EOF)
            tdcComposers.recordset.MoveLast();">
<INPUT ID=cmdNavLast TYPE=BUTTON VALUE=">>"
    onclick="tdcComposers.recordset.MoveLast()">
```

Using the recordNumber Property

If a page displays both single-valued and tabular data consumers on the page, the Web author can add functionality so that when the user clicks a row in the table, the current record and single-valued consumers are set to display the selected row. Likewise, when the user clicks navigation buttons on the page, the table can be synchronized to indicate the currently selected record.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbrecnum.htm

The script behind the example uses the recordNumber property available on all repeated elements within a databound table. This includes both bound and unbound elements. The property returns the number of the record displayed by the table row. In the example, when the user clicks any element within a table row, the click-handling code sets the **AbsolutePosition** property of the ADO recordset to the value of that element's **recordNumber** property.

In turn, if the user clicks a navigation button, the appropriate **Move** method is called on the ADO recordset. Then the highlighted row in the table is updated by searching for the table row with the **recordNumber** that corresponds to the value of the **AbsolutePosition** property on the ADO recordset. Once found, the background color of the table row is set to yellow.

Adding, Deleting, and Modifying Records in a Data Set

To support the addition and deletion of records from a recordset, the ADO recordset object supports both **AddNew** and Delete methods. When **AddNew** is called from a script, the DSO adds a new record to the locally cached data set. When **Delete** is called, the current record is removed from the local data cache.

Bound elements are immediately updated to reflect the addition, removal, and modification of records through scripts. If the current record is removed, single-valued elements are updated to display the next record in the data set. If the last record in the data set is deleted, the previous record is displayed. Tabular data consumers are updated to no longer display the deleted record.

It is important to distinguish between operations that are performed on the local data cache and those that affect the back-end data set. While the TDC, for example, supports the modification of data in the local cache, it does not support committing those changes to the file from which the data was obtained. In contrast, Microsoft Remote Data Service (RDS) does support this feature.

Click the Show Me button to see an example that exercises the ADO object model to insert, delete, and update records in the local data cache.

Code example: http://samples.msdn.microsoft.com/workshop/samples/author/databind/dbupdate.htm

Accessing the ADO Recordset Object Collections

The ADO recordset exposes two collections and a brief summary of each is given here. For comprehensive information on the ADO collection see Microsoft Active Data Objects (ADO).

- Fields

A basic ADO recordset, and each record within a recordset, has a collection of fields. The specific properties available within the fields collection depend to some extent on the type of data source being accessed. The fields collection is the default collection for a recordset object.

The sample code below shows how to enumerate the fields collection. For an ADO recordset named rsAttendees, the `name` and `value` propertiers of each field in the Fields collection are written to a document.

```
<SCRIPT Language="VBScript">
For Each objFld in rsAttendees.Fields
    document.write("The field name is " & objFld.Name & "<BR>")
    document.write("The field value is " & objFld.Value & "<BR>")
Next
</SCRIPT>
```

The Fields collection has a `Count` property, which indicates the number of fields in the collection.

- Properties

As with every type of ADO object, an ADO recordset has a Properties collection. Each property within the Properties collection has a `name`, `type` and `value` and these are easily obtained through script, using a similar approach to the code sample shown above for the Fields collection.

The sample code below shows how to enumerate the Properties collection. The following code extracts each property in the Properties collection, for each field in an ADO recordset object named rsAttendees, and writes the value of each property to a document.

```
<SCRIPT Language="VBScript">
For Each objFld in rsAttendees.Fields
document.write("The field named " & objFld.Name & " has the following properties:<BR>")
     For Each objProp in objFld.Properties
     document.write("The " & objProp.Name & " property has the value: " & objProp.Value & '
     Next
Next
</SCRIPT>
```

Community Content