



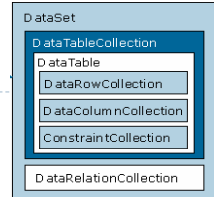
# COMPSCI 280 S1 2011 Enterprise Software Development

ADO.NET  
DataTables, DataColumns & DataRows



## Agenda & Reading

- Agenda:
  - DataTables
    - Creating DataTables
    - DataTable Properties
  - DataColumns
    - Properties
    - Creating DataColumns
  - DataRows
    - Properties
    - Methods
  - DataTables Methods
  - Validation using ColumnChanging and RowChanging event handlers
- Recommended Reading:
  - Microsoft ADO.NET 2.0 step by step, Rebecca M. Riordan
    - Chapter 7: Using Data Tables
  - MSDN Library
    - <http://msdn.microsoft.com/library/en-us/vbcon/html/vbtskAddingUntypedDatasetsToFormOrComponent.asp>
    - <http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcreatingdatatables.asp>
    - <http://msdn.microsoft.com/library/en-us/cpguide/html/cpconmanipulatingdatainadonetdatatables.asp>



### Hands-on Lab:

- Lecture22Lab: Validating data using DataTable events



## Understanding DataTables

Example: DataTableDemo

- DataTable
  - DataSet is an in-memory representation of relational data.
  - DataTable
    - Contains the actual data within the DataSet.
    - Contains columns, rows, constraints and relations

### Creating DataTables

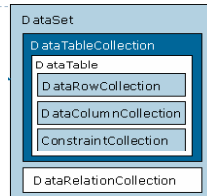
- To create an independent DataTable
 

```
DataTable dtTable = new DataTable("TEMP");
```
- To create a DataSet Table
  - Using the DataTable constructor or by passing constructor argument to the Add method of the Tables property of the DataSet
 

```
DataTable dtStudents = new DataTable("Students");
northwindDataSet.Tables.Add(dtStudents);
```
  - Using the Fill method of the TableAdaptor object
 

```
customersTableAdapter.Fill(northwindDataSet.Customers);
```
- To refer an existing DataSet Table
 

```
DataTable dtTable1 = northwindDataSet.Customers;
```

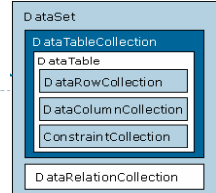


- You can create and use a DataTable independently or as a member of a DataSet.
- You can access the collection of tables in a DataSet through the Tables property of the DataSet object.
- You are not required to supply a value for the TableName property when you create a DataTable; you can specify the property at another time, or you can leave it empty. However, when you add a table without a TableName value to a DataSet, the table will be given an incremental default name of TableN, starting with Table1.



## DataTables Properties

- TableName
  - Gets or sets a virtual table name
  - Unique
- Columns
  - Read-only
  - Retrieves a DataColumnCollection object containing a collection of all the columns forming a table
  - .Count returns the total number of columns
- Rows
  - Retrieves a DataRowCollection object containing each row comprising a DataTable record
  - Manages every row, represented by a DataRow object, contained in the DataTable
  - Adds new, deletes and modifies rows
  - .Count returns the total number of rows



TableName: Customers  
Number of columns: 11  
Number of rows: 91

```
txtLog.AppendText("TableName:" + northwindDataSet.Customers.TableName);
txtLog.AppendText("Number of columns:" + northwindDataSet.Customers.Columns.Count);
txtLog.AppendText("Number of rows:" + northwindDataSet.Customers.Rows.Count);
```

- The schema, or structure of a table is represented by columns and constraints. You define the schema of a DataTable using DataColumn objects as well as ForeignKeyConstraint and UniqueConstraint objects.
- A DataTable must also have rows in which to contain and order data.



# Understanding DataColumnns

- ▶ **DataColumn**
  - ▶ Fundamental building block of a DataTable schema (contained in Columns collection)
  - ▶ Define the structure of the table.
- ▶ To refer an existing DataColumn from a DataSet DataTable
 

```
DataColumn dcCol = northwindDataSet.Customers.Columns[0];
```
- ▶ **DataColumn Properties**
  - ▶ ColumnName
    - ▶ Gets or sets the column name of the DataColumn object
  - ▶ DefaultValue
    - ▶ Sets the default value when a new record is created
  - ▶ DataType
    - ▶ Gets or sets a column type
  - ▶ Expression
    - ▶ Gets or sets the expression used to calculate the values in a column, or create an aggregate column

```
txtLog.AppendText("The first Column is " + dcCol.ColumnName.ToString());

DataColumn dcCountry = northwindDataSet.Customers.CountryColumn;
dcCountry.DefaultValue = "New Zealand";
```

The columns in a table can map to columns in a data source, contain calculated values from expressions, automatically increment their values, or contain primary key values.



# DataColumns (con't)

Example: DataTableApp

- ▶ **Creating DataColumnns**
  - ▶ Create a new DataColumn using a constructor
    - ▶ DataColumn(ColumnName)
    - ▶ DataColumn(ColumnName, dataType)
    - ▶ DataColumn(ColumnName, dataType, expression)
  - ▶ Add the new DataColumn to the Columns collection of a DataTable
 

```
DataColumn dc1 = northwindDataSet.Customers.Columns.Add();
dc1.DefaultValue = "1";
DataColumn dc2 = new DataColumn("FullAddress", Type.GetType("System.String"),
"Address + ' ' + City");
northwindDataSet.Customers.Columns.Add(dc2);
```

Expression-based column
- ▶ **Type.GetType**
  - ▶ Gets a Type object that represents the specified type.
  - ▶ String: Type.GetType("System.String")
  - ▶ Integer: Type.GetType("System.Int32")
- ▶ **Expression**
  - ▶ Gets or sets the expression used to filter rows, calculate the values in a column, or create an aggregate column

- ◆ You create DataColumn objects within a table by using the DataColumn constructor, or by calling the Add method of the columns property of the table, which is a DataColumnCollection. The Add method will add the DataColumn to the collection, and will return a reference to the added DataColumn if requested.
- ◆ If a column name is not supplied for a column, the column is given an incremental default name of ColumnN, starting with Column1.



# Expression-based DataColumnns

- ▶ **Understanding expression-based DataColumnns**
  - ▶ Read-only. Values are stored in memory all the time
  - ▶ Automatically refresh the computed values as the involved column is updated
- ▶ **Example**
  - ▶ Calculate a value by combining multiple columns of the same table using operators
 

```
dc2.Expression = "Address + ' ' + City";
dc3.Expression = "FirstName = 'John'";
dc4.Expression = "Birthdate < #1/31/82#";
dc5.Expression = "UnitPrice * 0.086";
dc6.Expression = "Year >= 1999 AND LastName LIKE 'A%'";
```

## Simple Functions and aggregates Functions

```
... = "Len(ItemName)";
... = "IsNull(price, -1)";
... = "IIF(total>1000, 'expensive', 'dear')";
... = "SUBSTRING(ContactName, 2, 3)";
... = "Convert(total, 'System.Int32')";

... = "Sum(Freight)";
... = "Avg(Freight)";
... = "Min(Freight)";
... = "Max(Freight)";
... = "Count(Freight)";
```

IsNull function: Return price; if the price is null, return -1

Substring(columnName, index, length), Index start from 1  
E.g. "Michael" => "ich"

- ◆ String values should be enclosed within single quotes and Date values should be enclosed within pound signs (#)
- ◆ If a column name contains special characters (#, =, <, >, %, &, etc), the name must be wrapped in brackets. For example: "UnitPrice \* [Column#]"
- ◆ Both the \* and & can be used interchangeably for wildcards in a LIKE comparison. Wildcards are not allowed in the middle of a pattern. For example, 'te\*xt' is not allowed.
- ◆ IsNull: Checks an expression and either returns the checked expression or a replacement value.
- ◆ IIF: Gets one of two values depending on the result of a logical expression.
- ◆ Convert: Converts given expression to specified .NET Framework type



# DataRows

```
NorthwindDataSet.CustomersRow drRow = northwindDataSet.Customers[0];
txtLog.AppendText(drRow.CompanyName);
```

- ▶ **The DataRow object**
  - ▶ Represent data in a DataTable
  - ▶ Conforms to schema defined by DataColumnns
- ▶ **Properties**
  - ▶ Item
 

```
DataRow drRow = northwindDataSet.Customers.Rows[0];
```

    - ▶ Gets or sets a column value contained in the DataRow
    - ▶ To refer to an existing row from a DataSet DataTable

Use indexer
  - ▶ ItemArray
    - ▶ Gets or sets an array of values associated with the columns contained in the DataRow object
  - ▶ RowError
 

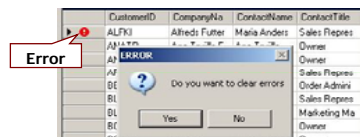
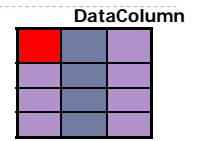
```
Object[] items = drRow.ItemArray;
txtLog.AppendText(items.Length);
```

    - ▶ Gets or sets the custom error description of a row
  - ▶ HasError
 

```
drRow.RowError = "A custom error description!";
```

    - ▶ Gets a boolean value indicating whether there are errors in a row
- ▶ **To iterate through all rows**

```
foreach (NorthwindDataSet.CustomersRow drRow in northwindDataSet.Customers)
txtLog.AppendText(drRow.CompanyName.ToString());
```



- ◆ The DataRow class represents the actual data contained in a table. You use the DataRow and its properties and methods to retrieve, evaluate, and manipulate the data in a table. As you access and change the data within a row, the DataRow object maintains both its current and original state.



# DataRow Properties (con't)

## RowState

- Gets a DataRowState Enumeration value for the specific row contained in the DataRowCollection inside the DataTable
- Reflects the actions that have been taken since the DataTable was created or since the last time the AcceptChanges method was called

Unchanged	No pending changes
Detached	The row has been created. The row has not been added to the DataRowCollection
Added	The row has been added
Modified	The row contains pending changes. (AcceptChanges has not been called.)
Deleted	The row is pending deletion

## RowVersion

- The version of the DataRow objects
  - You can look at the current value in a row, or the original value
  - Once you have committed changes, the proposed changes are the **current** row value

Example: (the first row, column: Country)  
 DataRowState: Unchanged  
 RowVersion:  
 current: Germany,  
 default: Germany,  
 original: Germany

Current	Row contains current values
Default	Row is versioned according to the DataRowState
Original	Row contains its original values
Proposed	Row contains proposed values

A row state indicates the status of a row. Row versions maintain the values stored in a row as it is modified including current, original. For example, you have made a modification to a column in a row, the row will have a row state of modified, and two row version exist: Current, which contains the current row values, and Original, which contains the row values before the column was modified.



# DataRow Methods

## HasVersion(DataRowVersion)

- Gets a value that indicates whether a specified version exists.

```
if (dr.HasVersion(DataRowVersion.Current)) {
    txtLog.AppendText(dr[0, DataRowVersion.Current].ToString());
}
```

## SetColumnError(String, Error\_String)

- Sets the error description for a column specified by name.

## IsNull(DataColumn)

- Gets a value that indicates whether the specified DataColumn contains a null value.

```
foreach (DataRow dr in northwindDataSet.Customers.Rows)
    foreach (DataColumn dc in northwindDataSet.Customers.Columns)
        if (dr.IsNull(dc))
            txtLog.AppendText("A null value has been found!");
```

## BeginEdit

- Begins an edit operation. Disables the raising of events and exceptions (make sure no other operation will be done from any other objects)

## EndEdit

- Ends an edit operation. Enables the raising of events and exceptions

## CancelEdit

- Cancels the edit operation on the row, rolling back all the data to their original values

When you modify column values in a DataRow directly, the DataRow manages the column values using the Current, Default, and Original row versions. In addition to these row versions, the BeginEdit, EndEdit, and CancelEdit methods use a fourth row version: Proposed.

Note: EndEdit does not save your changes to your actual data source.



# Updating a DataRow

## BeginEdit() & EndEdit()

```
DataRow dr = northwindDataSet.Customers.Rows[0];
// DataRowState:Unchanged, c:Germany, d:Germany, o:Germany
dr.BeginEdit();
// DataRowState:Unchanged, c:Germany, d:Germany, o:Germany, p:Germany
dr["Country"] = "New Zealand";
// DataRowState:Unchanged, c:Germany, d:New Zealand, o:Germany, p:New Zealand
dr.EndEdit();
// DataRowState:Modified, c:New Zealand, d:New Zealand, o:Germany
```

## BeginEdit() & CancelEdit()

```
DataRow dr = northwindDataSet.Customers.Rows[0];
dr.BeginEdit();
dr["Country"] = "New Zealand"
// DataRowState:Unchanged, c:Germany, d:New Zealand, o:Germany, p:New Zealand
dr.CancelEdit();
// DataRowState:Unchanged, c:Germany, d:Germany, o:Germany
```

The Proposed row version exists during an edit operation that is begun by calling BeginEdit and that is ended by using either EndEdit or CancelEdit.



# Inserting a new DataRow

## To create a new DataRow

- Use the NewRow method to create a new row
  - Newly created row has the same schema as the underlying DataTable
- Set values
- Use the Add method of the rows collection to add
  - Note: When created, this new row does not yet belong to the underlying DataTable

```
DataRow drNew = northwindDataSet.Customers.NewRow();
// DataRowState:Detached, d:, p:
drNew["CustomerID"] = "ABCDE";
// DataRowState:Detached, d:ABCDE, p:ABCDE
northwindDataSet.Customers.Rows.Add(drNew);
// DataRowState:Added, c:ABCDE, d:ABCDE
// No Original version
```

- Or, call the Add method to add a new row by passing an array of values, typed as Object.

## Rows.Add(Object())

- Adds a DataRow to the DataRowCollection.

```
northwindDataSet.Customers.Rows.Add(new Object[] { "EFGH" });
```



# Removing/Deleting a DataRow

- Remove
  - Completely **removes** the row from the **collection**

```
DataRow dr = northwindDataSet.Customers.Rows[0];
northwindDataSet.Customers.Remove(dr);
```

Annotations: DataRowState: Unchanged, c: ALFKI, d: ALFKI, o: ALFKI; DataRowState: Detached; Total number of customers: 90

- Delete
  - Deletes a row
  - Marks the row as deleted
  - Hidden, but still accessible if necessary

```
DataRow dr = northwindDataSet.Customers.Rows[0];
dr.Delete();
```

Annotations: DataRowState: Deleted, o: ALFKI; No Current version; Total number of customers: 91

The Delete method marks the row as Deleted in the DataSet but does not remove it. Instead, when you call the Update method of a DataAdapter, it executes its DeleteCommand to delete the row at the data source. The row can be permanently removed. If you use Remove to delete the row, the row will be removed entirely from the table but the DataAdapter will not delete the row at the data source.



# DataTable Methods

- Clear
  - Clears all the data contained in the DataTable object
- Clone
  - Creates a new DataTable object
  - Generates only an empty object with the same DataTable object structure

```
DataTable dtTable = northwindDataSet.Customers.Clone();
```

- Copy
  - Creates a new DataTable object
  - Produces an object with the same table schema, relations, constraints and data.

```
DataTable dtTable = northwindDataSet.Customers.Copy();
```

- Select
  - Used to filter and sort the rows of a DataTable
    - Three optional parameters
      - Filter expression
      - Sort
      - DataRowState
  - Returns an array of DataRows that match the criteria you specify

More examples:  
"CustomerID in ('ALFKI', 'ANATR')"  
"CustomerID='WILMK'"

```
DataRow[] drFound;
string strFilter = "CustomerID LIKE 'A*';
drFound = northwindDataSet.Customers.Select(strFilter, "Country DESC",
DataRowState.Unchanged);
foreach (DataRow dr in drFound)
txtLog.AppendText(dr["CustomerID"] + " ");
```



# DataTable Events

- Events are fired by .NET framework when specific circumstances are verified

ColumnChanged	Occurs after a value <b>has been changed</b> for the specified DataColumn in a DataRow.
ColumnChanging	Occurs when a value <b>is being changed</b> for the specified DataColumn in a DataRow.
RowChanged	Occurs after a DataRow <b>has been changed</b> successfully.
RowChanging	Occurs when a DataRow <b>is changing</b> .
RowDeleted	Occurs after a row in the table <b>has been deleted</b> .
RowDeleting	Occurs before a row in the table <b>is about to be deleted</b> .

- Validating Data During Column Changes

- To validate data changes
  - Write the ColumnChanging event handler
  - Connect events with the event handler methods

```
northwindDataSet.Customers.ColumnChanged += OnColumnChanged;
...
```

When records are updated, the DataTable objects raise events that you can respond to as changes are occurring and after changes are made. The ColumnChanging, RowChanging, and RowDeleting events are raised during the update process. You can use these events to validate data or perform other types of processing. The ColumnChanged, RowChanged, and RowDeleted events are notification events that are raised when the update has been completed successfully. These events are useful when you want to take further action based on a successful update. Any procedure can serve as an event handler as long as it supports the correct argument for the event being handled.



# DataTable Events (con't)

- DataColumnChangeEventArgs parameter contains:
  - Column: the DataColumn object containing the **changed** value
  - Row: the DataRow object containing the **changed** value
  - ProposedValue: get or set the ProposedValue object
- ColumnChanging
  - Occurs when a value **is being changed** for the specified DataColumn in a DataRow.

```
private void OnColumnChanging(Object sender, DataColumnChangeEventArgs args) {
PrintDataRow(args.Row, args.Column.ColumnName);
txtLog.AppendText("ProposedValue=" + args.ProposedValue);
...
DataRowState = Unchanged, c: Berlin, d: Berlin, o: Berlin, p: Berlin
```

- ColumnChanged
  - Occurs after a value **has been changed** for the specified DataColumn in a DataRow.

```
private void OnColumnChanged(Object sender, DataColumnChangeEventArgs args) {
PrintDataRow(args.Row, args.Column.ColumnName);
...
DataRowState = Unchanged, c: Berlin, d: Berlin, o: Berlin, p: Auckland
```

The ColumnChanging and ColumnChanged events are raised during and after each change to an individual column. It is useful when you want to validate changes in specific columns.



## DataTable Events (con't)

- ▶ **DataRowChangeEventArgs** parameter contains:
  - ▶ **Action**: the action description for the **changed** row
    - ▶ Add, Change, Commit, Delete, Nothing etc..
  - ▶ **Row**: the DataRow object containing the **changed** value

### RowChanging

- ▶ Occurs when a DataRow **is changing**.

```
private void OnRowChanging(Object sender, DataRowChangeEventArgs args) {
    if (args.Action == DataRowAction.Change) {
        txtLog.AppendText("RowChanging:" + args.Row.RowState.ToString());
        ...
    }
}
```

RowChanging: Action = Change  
DataRowState = Unchanged

### RowChanged

- ▶ Occurs after a DataRow **has been changed** successfully.

```
private void OnRowChanged(Object sender, DataRowChangeEventArgs args) {
    if (args.Action == DataRowAction.Change) {
        txtLog.AppendText("RowChanged:" + args.Row.RowState.ToString());
        ...
    }
}
```

RowChanged  
DataRowState = Modified

- ▶ The RowChanging and RowChanged events are raised during and after any change in a row.
- ▶ By default, each change to a column therefore raises four events: first the ColumnChanging and ColumnChanged, and then the RowChanging and RowChanged events. If multiple changes are being made to the row, the events will be raised for each change. Note: the data row's BeginEdit method turns off the RowChanging and RowChanged events after each individual column change. In that case, the event is not raised until the EndEdit method has been called, when the RowChanging and RowChanged events are raised just once.

17

COMPSCI280

Handout22



## DataTable Events (con't)

### RowDeleting

- ▶ Occurs before a row in the table **is about to be deleted**.

```
private void OnRowDeleting(Object sender, DataRowChangeEventArgs args) {
    txtLog.AppendText(args.Row.RowState.ToString());
    ...
}
```

RowDeleting: Action = Delete  
DataRowState = Unchanged

### RowDeleted

- ▶ Occurs after a row in the table **has been deleted**.

```
private void OnRowDeleted(Object sender, DataRowChangeEventArgs args) {
    txtLog.AppendText(args.Row.RowState.ToString());
    ...
}
```

RowDeleting: Action = Delete  
DataRowState = Deleted

18

COMPSCI280

Handout22



## Validation

```
private void OnColumnChanging(Object sender,
    DataColumnChangeEventArgs args) {
    if (args.Column.ColumnName == "City") {
        ...
    }
}
```

### ColumnChanging Event handler

- ▶ Evaluate the ProposedValue
- ▶ If the value is invalid,
  - ▶ Modify the proposedValue, or
  - ▶ Restore the current value, or
  - ▶ Set the ColumnError

```
if (args.ProposedValue == null)
    args.ProposedValue = "Auckland";
```

```
if (args.ProposedValue == null)
    args.ProposedValue = args.Row["City", DataRowVersion.Current];
```

```
if (args.ProposedValue == null)
    args.Row.SetColumnError(args.Column.ColumnName, "City can't be blank.");
else
    args.Row.SetColumnError(args.Column.ColumnName, "");
```

- ▶ During the edit operation you can apply validation logic to individual columns by evaluating the ProposedValue in the ColumnChanging event of the DataTable. The DataColumnChangeEventArgs keep a reference to the column that is changing and the ProposedValue. After you evaluate the proposed value, you can either modify it or cancel the edit. When the edit is ended, the row moves out of the Proposed state.

19

COMPSCI280

Handout22



## Summary

- ▶ **DataColumns**
  - ▶ Properties: ColumnName, DataType, etc
  - ▶ Expression-based
- ▶ **DataRows**
  - ▶ Properties: Item, RowError, HasError
    - ▶ RowState: Unchanged, Detached, Added, Modified & Deleted
    - ▶ RowVersion: Current, Original, Proposed & Default
  - ▶ Methods: BeginEdit, EndEdit, Delete, Remove, IsNull, HasVersion, SetColumnError etc
- ▶ **DataTables**
  - ▶ Methods: Clear, Clone, Copy, etc
  - ▶ Events: ColumnChanging, ColumnChanged, RowChanging, RowChanged, RowDeleting & RowDeleted
- ▶ **Validation**
  - ▶ Validate individual columns
    - ▶ Evaluate the ProposedValue in the **ColumnChanging** event of the DataTable

20

COMPSCI280

Handout22