

# Menus

Component

Menu (abstract)

MainMenu

ContextMenu

MenuItem

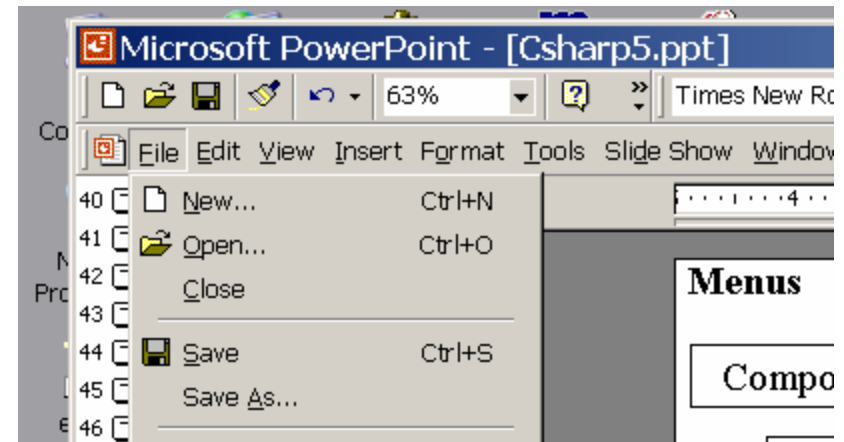
MainMenu is the menu bar typically below the Form's title.

ContextMenus are shown at the mouse point with a right click

Conceptually the MainMenu is a set of MenuItem's that each contain a set (sub menu of) MenuItem's....

Menus can be easily designed and implemented with Designer.

<< do demo >>



# Menus by hand

## Constructors

```
MainMenu ();
```

```
MainMenu (MenuItem[] menuItems);
```

```
ContextMenu ();
```

```
ContextMenu (MenuItem[] menuItems);
```

```
MenuItem ();
```

```
MenuItem (string str);
```

```
MenuItem (... , EventHandler eventHandlerName);
```

```
MenuItem (... , ShortCut shortCutEnumeration);
```

```
MenuItem (string str, MenuItem[] menuItems);
```

Form Properties      the menu for a Form

MainMenu              Menu                      get/set

Control Properties    the menu for a Control

ContextMenu           ContextMenu              get/set

## simple menu

```
// probably in a constructor ....
// create menu items for a File menu
MenuItem Open = new MenuItem("&Open...",
    new EventHandler(doFileOpen), Shortcut.CtrlO);
MenuItem Save = new MenuItem("&Save",
    new EventHandler(doFileSave), Shortcut.CtrlS);
MenuItem SaveAs = new MenuItem("Save &As...",
    new EventHandler(doFileSaveAs));
MenuItem miDash = new MenuItem("-");
MenuItem miExit = new MenuItem("E&xit",
    new EventHandler(doExit));
// create the File menu
MenuItem miFile = new MenuItem("&File",
    new MenuItem[] {miOpen, miSave, miSaveAs, miDash, miExit});
// repeat for remaining menuItems of each menu
....
```

# Geometry

## Geometry Management: Position or Constraint Hierarchy

Position: controls have Location & Size properties

in resize event handler (or override OnResize)

Use variables, arrays, expressions to set control's Location and Size

get parent's clientRect (or Location and ClientSize)

recompute component's position.

## Constraint Hierarchy

dock {Top, Left, Fill, Right, Bottom, None} how control's size responds to parent's resize.

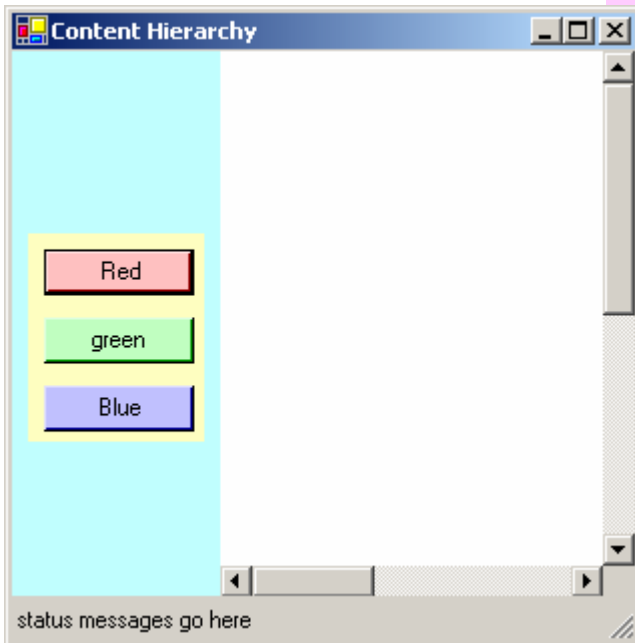
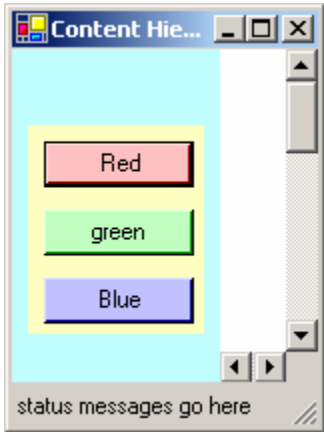
anchor {Top, Left, Right, Bottom, None } default Top|Left  
how control positions relative to sides of the form.

when anchored to opposite sides control's size changes

# Layout

## Containment Hierarchy -- contentHierarch.cs

### Form1



statusBar -  $d = B, a = T | L$

bottomPanel -  $d = F, a = T | L$

leftPanel -  $d = L, a = T | L$

buttonPanel -  $d = N, a = L | R$

redButton -  $d = N, a = T | L | R$

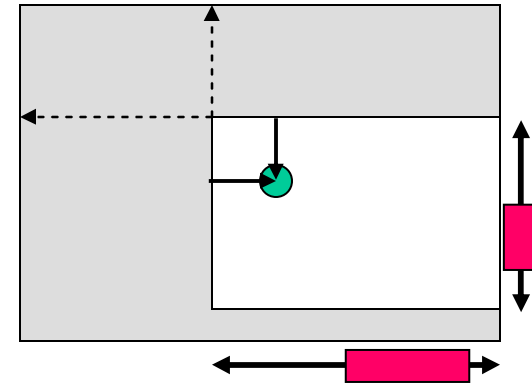
greenButton -  $d = N, a = L | R$

blueButton -  $d = N, a = B | L | R$

fillPanel -  $d = N, a = T | B | L | R$

# Control Auto Scroll Properties

bool	AutoScroll	get / set	false
Size	AutoScrollMargin	get / set	0, 0
Size	AutoScrollMinSize	get / set	0, 0
Point	AutoScrollPosition	get / set	



Margin is between contained controls and scrollable parent

MinSize of the scrollable control.

Scroll bars visible when  $ClientSize < MinSize$

AutoScrollPosition used to adjust display of child controls in scrolled parent.

As scroll thumb is moved right or down  $AutoScrollPosition.X$  or  $Y$  increases as a negative offset. 0 to  $-(AutoScrollMinSize - ClientSize)$

Also see panel in panel scroll example.

# Parent - Child

## Properties

Control.ControlCollection      Controls      get

## Methods

```
void Add (Control aControl);  
void AddRange (Control[] aControl);  
void Remove (Control aControl);  
void RemoveAt (int index);  
bool Contains (Control aControl);  
int getChildIndex (Control aControl);  
void setChildIndex (Control aControl, int index);
```

child controls are implicitly added to a form with  
`aControl.Parent = theForm;`

the form's `Control.count` property is incremented as controls are added

# Buttons

## Bitmaps and Owner-drawn

### ButtonBase Properties

Image	Image	get / set	
ImageList	imageList	get / set	// collection of images

Bitmaps are \*.bmp set in Designer or code w/

```
aButton.Image = new Bitmap(GetType(), "*.bmp");  
new Bitmap (Type extractorClass, string fileName);
```

Program Files\Microsoft Visual Studio .NET 2003\Common7\Graphics\  
subdirectories: bitmaps, cursors, icons

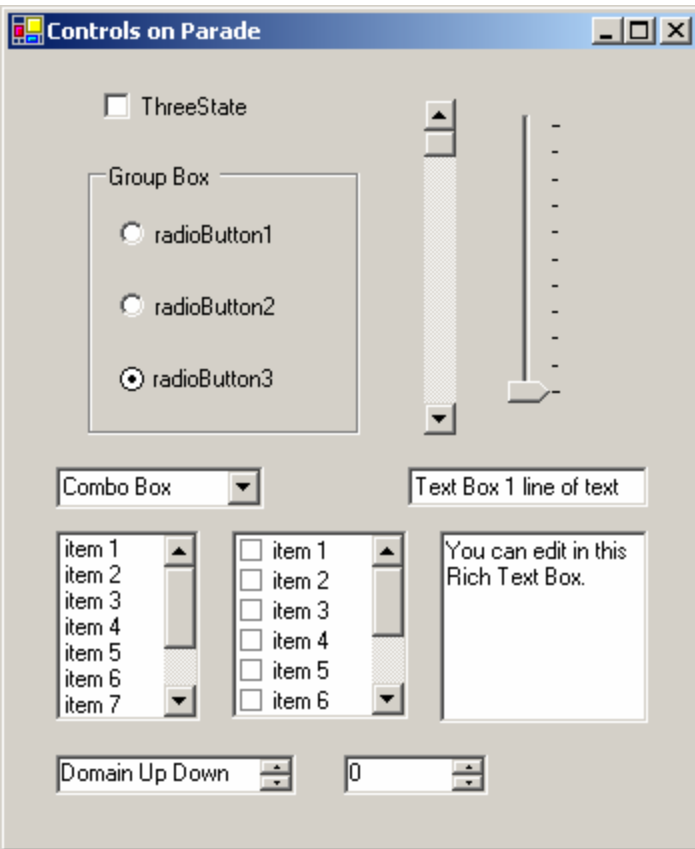
### Owner-drawn buttons

in a ButtonOnPaint(...) PaintEventHandler

use ControlPaint's overridden **DrawButton(...)** to draw borders,

**DrawFocusRectangle(...)** and graphics draw commands

# Controls on Parade



**TextBox** usually 1 line

**RichTextBox** text area w/editing

**TextBase** properties:

bool **MultiLine**, **wordwrap**, **scrollbars**

**ListControl** (abstract) display & selection

**ListBox**

**CheckedListBox**

**ComboBox** - **TextBox** with **List**

**UpDownBase** (abstract) **Spinners**:

**NumericUpDown** numbers

**DomainUpDown** strings

# CheckBox

CheckBox -- normally an on/off checkbox

## Properties

bool	Checked	get / set	false
bool	AutoCheck	get / set	true
Appearance	Appearance	get / set	{Normal, Button }
bool	ThreeState	get / set	false
CheckState	CheckState	get / set	// use instead of Checked {Unchecked, Checked, Indeterminate}

## Events

CheckedChanged	OnCheckChanged	EventHandler	EventArgs
CheckStateChanged	OnCheckStateChanged	"	"

Group Box -- a container for other controls

a title and "border" around a set of controls

used for Radio Buttons

Radio Buttons

mutually exclusive set of buttons

similar properties and events as CheckBox (w/o ThreeState).

# ScrollBars

## Properties

int	Value	get / set	Min .. Max + 1 - LargeChange
int	Minimum	get / set	0
int	Maximum	get / set	100
int	SmallChange	get / set	1
int	LargeChange	get / set	10

## Events

ValueChanged	OnValueChanged	EventHandler	EventArgs
Scroll	OnScroll	ScrollEventHandler	ScrollEventArgs

## ScrollEventArgs Properties

int	NewValue	get / set	
ScrollEventType	Type	get	// how scrolled

many ways to use Scroll and ScrollEventArgs -- see text

TrackBar alternate form of ScrollBar with similar properties

# C # Collection interfaces

Windows Forms controls that have models use C# collections:

- IEnumerable**      iterate / enumerate through collection  
using foreach statement
- ICollection**      provides CopyTo(), and Count, IReadOnly, ISynchronized and SyncRoot properties
- IComparer**          compares two objects in collection for sorting
- IList**              used by array-indexable collections
- IDictionary**      used by key / value based collections
- IDictionaryEnumerator**      allow foreach use with Dictionaries

## Models && Views revisited

Windows Forms has complex controls for viewing and manipulating collections of data.

Single Collections:     NumericUpDown, DomainUpDown,  
                                  ComboBox, ListBox

Many Collections:     TreeView, ListView and DataGrid

Views of single collections store the model in an collection of objects. The views display, and provide UIs, to the items held in the collection.

ComboBox, NumericUpDown, and DomainUpDown all have an Items property: an ObjectCollection implementation of the IList, ICollection, and IEnumerable interfaces.

ListBox model has 3 collections:

- Items (like the above),
- SelectedItems (UI event result),
- SelectedIndices (UI events result).

## Hierarchical collections

The collections used by views of related collections are used to hold the values and relations.

TreeView displays collections that have a hierarchical relationship like directories and files. TreeView is used in File Explorer's Folder pane.

A TreeView has a single TreeNode

bool	Sorted	get / set
TreeNode	string + optional image	

TreeNode class has a Nodes property that is a TreeNode Collection

TreeNodeCollection	[ ]	get / set
int	Count	get
bool	IsReadOnly	get

TreeNode Collection implements IList, ICollection and IEnumerable with TreeNodes property [ ]

Each TreeNode can contain other TreeNodes (hierarchy)

## Tabular collections

ListView displays information in a single table of rows and columns with column headings. ListView is used in File Explorer's file pane.

The first column is the list views Items.

The remaining columns are the subitems of the item.

### ListView properties

View	View Enumeration
	{ LargeIcon, Details, SmallIcon, list }
SmallImageList	ImageList
LargeImageList	ImageList
Columns	ColumnHeaderCollection
Items	ListViewItemCollection

### ColumnHeaderCollection property:

[]	ColumnHeader	get
----	--------------	-----

## ListViewItem Collection property

[] ListViewItem get / set

## ListViewItem property:

SubItems ListViewSubItemCollection get

## ListViewSubItemCollection properties

[] listViewSubItem get / set

## Other property collections (results of events):

SelectedIndices, SelectedItems

## Other controls:

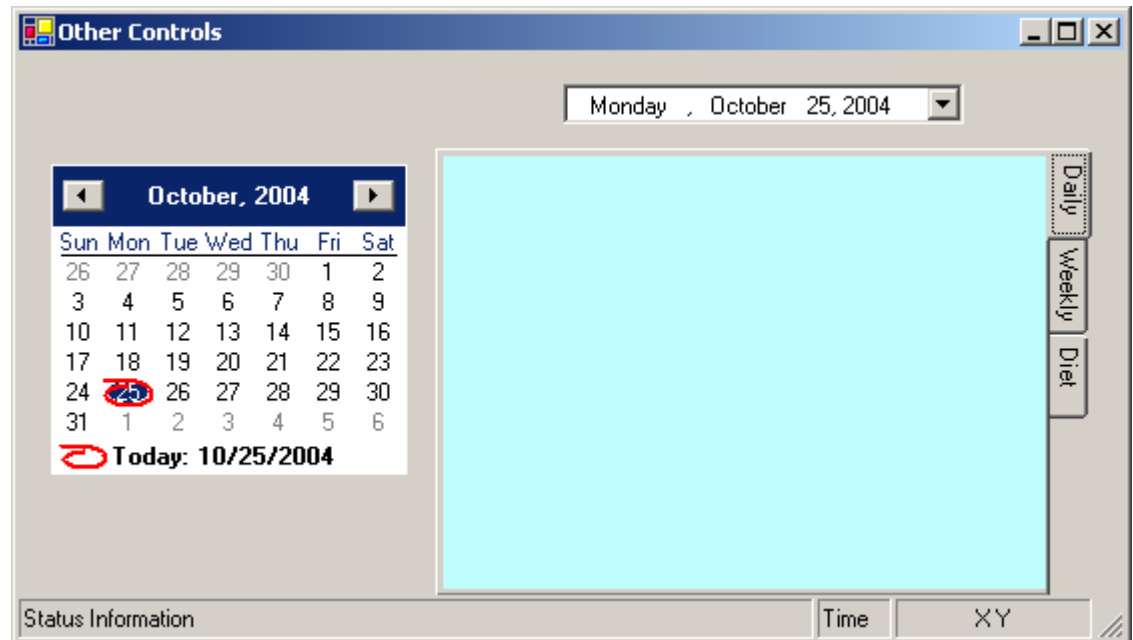
TabControl

DateTime Picker

MonthCalendar

StatusBar w/ panels

Splitter



## Database collections

Relational databases are collections of related tables (relations).

Each table is identified by a primary key.

A table's primary key can be a field (foreign key) in another table.

ADO.NET object model has: DataSet, DataTable and DataRelation objects

The DataSet is a subset of database cached on local machine.

Data connections are used to get and set the DataSet.

A DataSet has:

- Tables (collection of DataTable)

- Relations (collection of DataRelation) properties.

# Tables

Tables property returns a DataTablesCollection of all DataTable objects.

## DataTable properties

Columns	DataColumnsCollection	get
Rows	DataRowCollection	get
Constraints	ConstraintCollection	get

DataColumn represents a column in the table using property

item	[]	get / set
------	----	-----------

DataRow stores items in the row in property

ItemArray	object []	get / set
-----------	-----------	-----------

and individual column items in a row as property

Item	[]	get set
------	----	---------

# Relations

DataRelation represents the dependencies between two tables using DataColumn objects.

Relations property of DataSet returns a DataRelationCollection of DataRelations

Data Adapter is a bridge between a DataSet and the data source.

DataAdapter.Fill() retrieve data from a database and fill a data set.

DataAdapter.Update() inserts, updates, or deletes rows in the data set.

DBConnection represents a data source

DBCommand sends commands (SQL statements) to the database.

```
SQLDataAdapter myDataAdapter = new SQLDataAdapter(  
    commandString, connectionString);
```

```
DataSet myDataSet = new DataSet();
```

```
myDataAdapter.Fill(myDataSet, "DataSetName"); // tables
```

# DataGrid

The Window Form control DataGrid provides the view to DataTables

DataGrid has a DataSource property

```
DataGrid myDataGrid = new DataGrid();  
myDataGrid.DataSource = myDataSet.Tables("DataSetName");
```

When working with multiple (n) tables you need

- 1 connection object
- n SqlCommand objects for each table
- n SqlDataAdapter objects for each table

Create DataRelation object for each dependency between tables,  
add to DataSet

Create a DataViewManager for the DataSet to provide a view of the  
DataSet and set the Grid's DataSource. (use DefaultViewManager).

Set the DataMember property to the parent (base) table

# multi-tables

```
section ( snum, profID, studentID, class)
```

teaches

```
profs ( pid, name, office)
```

memberOf

```
deptFaculty ( dept, profID)
```

attends

```
students ( sid, name, major)
```

one dataset: semesterCourses

four tables: section, profs, students, deptFaculty

three relations: teaches, attends, memberOf

DataGrid shows relations facilitates navigation based on relations

Need to create relations after connections (adapters).

app needs to know semantics of the database.

# example

