Web Controls Document

Version 2.0

iInterchange – Technology Group

Note: This documentation is preliminary and is subject to change.

iTextBox Control

Features:

It is a custom control, which inherits the .Net TextBox control. Validation features are inbuilt with this control. We can have values to be entered only in Uppercase, Lowercase or Numeric.

Properties:

iCase: Values to be displayed in Upper, Lower or None. The Default value is **None**.

Numeric allows only Numeric values.

OnClientTextChange: Specify the Function name that will be used to do some operation when the value is changed in the control. This function will be called when the value changes.

Validator:

ErrorText: The Default value is **'*'.** This message is displayed in the validation control when validation fails.

Validate: To perform validations, this value should be true. **:** The Default value is False.

ValidationGroup: This will be useful when you need to validate only some controls in the page not for all. Validation will be done only for a particular group.

RequiredFieldValidation:

IsRequired: The Default value is False. To perform RequiredField validations, this value should be true.

ReqErrorMessage: The error message to display in the ValidationSummary control if RequiredField Validation fails.

InitialValue: The Default value against which the RequiredField validation Performs instead of empty value.

RangeValidation:

RangeValidation: The Default value is False. To perform Range validations, this value should be true. This validation tests whether the value of an input control is within a specified range.

RngMaximumValue: Gets or sets the value of the control that you are validating, which must be less than or equal to the value of this property.

RngMinimumValue: Gets or sets the value of the control that you are validating, which must be greater than or equal to the value of this property.

RngErrorMessage: The error message to display in the ValidationSummary control if Range Validation fails.

Type: This property used to specify the data type of the values to compare

CompareValidation:

CompareValidation: This validation will be used to compare the value entered by the user into an input control with the value entered into another input control or a constant value. The Default value is False. To perform Compare Validation, this value should be true.

CmpErrorMessage: The error message to display in the ValidationSummary control if Compare Validation fails.

ControlToCompare: Gets or sets the input control to compare with the input control being validated.

ValueToCompare: Gets or sets a constant value to compare with the value entered by the user into the input control being validated.

Type: Gets or sets the data type of the two values being compared. The default value is **string**.

Operator: Gets or sets the comparison operation used in validation. The default value is **Equal**.

SetFocusonError: This value should be false in order to avoid focus on the control. By Default, the value is True.

It is always advisable to set the compare validation in the second control only instead of both controls.

RegularExpressionValidation:

RegexValidation: This validation will be used to validate the value against the regular expression. The Default value is False. To perform Compare Validation, this value should be true.

RegularExpression: The Regular expression against which the control will be validated.

RegErrorMessage: The error message to display in the ValidationSummary control if Regular Expression Validation fails.

LookUpValidation:

LookUpValidation: This validation will be used to validate against the values in the database. The Default value is True. To perform Lookup Validation, this value should be true.

LkpErrMessage: The error message to display in the ValidationSummary control if Lookup Validation fails.

CustomValidation:

CustomValidation: This validation will be used to Perform user-defined validation on an input control. The Default value is False. To perform custom Validation, this value should be true.

CustomValidationFunction: Gets or Sets the Function name to be called that will perform user-defined validation on an input control.

CsvErrorMessage: The error message to display in the ValidationSummary control if Custom Validation fails.

Note: Lookup validation should be false in iTextBox control. By Default, lookup validation is True.

iLookup control

Objective:

It is a custom control that inherits iTextBox control .The main purpose of this control is used for retrieving records from the table. Search starts only when the Down Arrow key (by Default) is pressed. We can change the default key value to be pressed to start search operations. On further down Arrow key to be pressed, it moves t other records in the grid. The value will also be selected by clicking the left mouse button on a row in the Grid.

Features:

- To filter the values based on other lookup controls(Dependency).
- To bind the values to other lookups.
- Validation features are inbuilt with this control.

Properties:

LookupColumnCollection:

This collection is used for setting the values related to lookupcolumns.

ColumnName: Column value to be displayed from the lookup table. We need to set the columnName.

ControlToBind: To bind the related column to some other lookup control. We need to specify the lookup Control Id.

TobeHidden: Default value is **False**. To hide the column to be displayed in the grid, we need to set it as **True**.

DependentColumnCollection:

ColumnName: Specify the columnName. This column name is related to the child table.

DependentColumnName: Specify the columnName. This column name is related to the dependent parent table.

LookupName: Specify the Dependent lookup Control ID.

DependentChildControls: Specify the Dependent child controls (Lookup Control ID) with separated by commas.

Datakey: Set the column Name whose value will be displayed in the grid.

DoSearch: Default value is True. To perform Search operations.

LookupGrid: Grid that will be shown during search operations. You need to set the style for lookup Grid.

CssClass: CssClass name for Grid.

Grid : This style will be applied to the Grid.

SelectedItemStyle : This style will be applied during mouse over on the row.

ItemStyle: This style will be applied to each row

LookupIcon: The icon that will be used to start search operations when it is clicked.

CssClass: CssClass name for Lookup Icon.

IsVisible: Lookup Icon to be visible. The Default value is True.

Src: Source of the image.

MoreInfoIcon: The icon that will be used to display some help information when it is clicked.

CssClass: CssClass name for MoreInfo Icon.

IsVisible: MoreInfo Icon to be visible. The Default value is True.

Src: Source of the image.

ImgClick: specify the Function Name, which will be called when it is clicked.

For Ex: MoreInfo_Click().

AddNewIcon: The icon that will be used to add new items into the lookup control when it is clicked.

CssClass: CssClass name for AddNew Icon.

IsVisible: AddNew Icon to be visible. The Default value is True.

Src: Source of the image.

ImgClick: specify the Function Name, which will be called when it is clicked.

For Ex: AddNew_Click().

ClientFilterFunction: Specify the Function name that will be used to add custom filter condition for lookup control. This function will be called when the lookup starts search operation.

For Ex: GetFilter

OnClientTextChange: Specify the Function name that will be used to do some operation when the value is changed in the control. This function will be called when the lookup value changes.

iCase: Values to be displayed in Upper, Lower, Numeric or None. The Default value is **None**. **Numeric** allows only Numeric values.

TableName: The table code in the choose table from which the values will be retrieved.

Client Side Variables:

SelectedValues: This variable will be used to get the values of the Selected row. This is an array collection.

For Ex: We can access the selected value of the second column in the lookup control, we can use it as:

Var selVal = ILookup1.SelectedValues[0];

DependentControls: This variable will be used to get the collection of Dependent Lookup controls.

Validator: These features are same as iTextBox Control.

Note: Lookup validation should be true to validate the values entered against the database.

IDate control

Objective:

It is a custom control that inherits iTextBox control. This control will be used for Entering date.

Properties:

DateIcon: The icon that will be used to display calendar when it is clicked.

CssClass: CssClass name for Date Icon.

IsVisible: Date Icon to be visible. The Default value is True.

Src: Source of the image.

Top: To set the top position of the calendar display. It should be in pixels.

Left: To set the Left position of the calendar display. It should be in pixels.

Validator: These features are same as iTextBox Control.

Note: Lookup validation should be true to validate the values entered against the Date Format.

Valid Format: The various formats that can be acceptable in the control are:

ddMMyyyy, dd/MM/yyyy, ddMMyy, dd/MM/yy, dd-MM-yy, dd-MM-yyyy,

dd-MON-yyyy, dd-MON-yy

The default format will be shown as **dd-MON-yyyy.**

iContainer Control

Objective:

It is a custom control that inherits iTextBox control. This control will be used for entering container number, which will be validated.

Properties:

Checkdigit: By Default, Value is False. This is used to enter the 11th digit number automatically when the focus is moved from the control.

Note: Lookup validation should be true to validate the values entered against the container Number Format.

iFlexGrid Control

Purpose:

This is useful for Single Cell Editable, Updating records row wise, Deleting and Inserting Records. Validations can be done for every cell. It is also possible for placing Grid inside another Grid. The nested Grid also has the same feature of the parent Grid.

Features:

- Single Cell Editable. This can be done by clicking the cell in the grid
- Update will be done by row wise when you move onto different row by changing values in one row.
- Delete record by selecting a particular record and clicking delete button in the footer row
- Add a Record either by pressing the tab key on last cell or by clicking the Add button in the footer row.
- Validations will be done once focus moves from the cell.
- Paging is possible.
- Grid in Grid.
- Single cell editable feature in Nested Grid also. The above feature will be possible in the nested Grid also.
- Merge able Header columns

Single Cell Editable:

When you click the cell, The appropriate control will be rendered in the cell based on Column type.

Allow Edit – This property should be true in order to make the cell editable in the grid. If the value is false, then it will be like a read-only grid.

Navigation Keys:

Tab: It moves to the next cell which will be an editable one . If that cell is read-only, then it moves to the next to that read only cell. Similarly if the next cell is Hyperlink, it moves to the next to that cell.

Shift + Tab: It moves to the previous cell which will be an editable one. If that cell is read-only, then it moves to the previous to that read only cell. Similarly if the next cell is Hyperlink, it moves to the previous to that cell.

Database Operations:

Add Record:

A new row is added in the grid in the same page Index when you click the Add button in the footer row or pressing the tab key on the last cell of the last row in the grid. By clicking the Add button, a control will be rendered to the first cell of the new row.

The added new rows will be moved to last row in the last page of the grid only when me move to some other page or while saving or deleting records in the grid.

AllowAdd – This property should be true in order to add rows in the grid. If the value is false, then Add button won't be shown in the grid and also not possible to add any rows to the grid.

Update Record:

Updation will be done row by row. When the focus moves from one row to another row, it checks for any changes in the grid and data will be saved in the datasource. Before updating, data will be validated.

Delete Record:

First you need to select a row and then click the Delete Button. The row will be deleted in the data source. The deleted row will be immediately deleted from the grid.

Allow Delete – This property should be true in order to delete rows in the grid. If the value is false, then Delete button won't be shown in the grid and also not possible to delete any rows to the grid. At a time, Single row will be deleted.

Validation:

When the focus moves from cell to another cell in the same row or different row, validation occurs. If the validation fails, focus will remain in the cell and it is not possible to edit any other cell in the grid.

Grid in Grid Feature:

By clicking the '+' icon in the column of the grid, the child grid will be shown. Then the "-" icon will be shown. By again clicking the "-" icon in the same row or '+' icon in different row, the child grid shown will be collapsed.

The child grid also has the same feature as parent grid. At a time ,single child grid will be shown. We can have any number of nested grids in the grid .

When the child grid is displayed, it is not possible to do any operation in the parent grid until the child grid is collapsed.

Search in Grid:

We need to set the property **allowSearch** be true. If **AutoSearch** is true, Search operations will be handled in control level itself. Otherwise, You need to set the datasource and bind it in the search event.

When you click the search icon in the header row, a new search row will be created after the header row. Cells in the search row will have the search control (whose cell has **allowSearch** property is true) based on column type. Each search control in the cell has html input control as well as Filter icon. You have also have the option of selecting the FilterOperator using the filter icon in each cell.

When you click the icon in the search row, the filtered datasource will be bind it to the grid and there wont be any search row in the grid.

Properties:

AllowAdd: To perform Add operation in the Grid control, this value should be True. By Default, this value is False.

AllowEdit: To modify the values in the Grid control, this value should be True. By Default, this value is False.

AllowDelete: To perform Delete operation in the Grid control, this value should be True. By Default, this value is False.

AllowPaging: To display the rows in page-by-page, this value should be true. By Default, this value is False.

AllowStaticHeader: This value should be true for static header rows.By Default, this value is False.

AllowSearch: To perform Search operation in the Grid control, this value should be True. By Default, this value is False.

AutoSearch: If this value is true, Search operations will be handled in control level itself. Otherwise, You need to set the datasource and bind it in the search event.

StaticHeaderHeight: The value should be in pixels. This is used to specify the height of the grid.

DataKeyNames: This is used to specify the Datakey values for the grid. The Values will be separated by commas. This property is **mandatory**.

DataMember: The table Name, whose values will be binded to the grid.

HeaderRows: Set or Get the number of Header Rows in the Grid, The Default value is zero. This property is **mandatory**.

OnAfterCallback: Specify the function name that will be called after the callback operation is over. This will be useful for passing parameters from the server side by setting the **e.outputparameters** in the server side events like RowInserting, RowUpdating and RowDeleting Events, which will be accessible in the client side function.

OnBeforeCallback: Specify the function name that will be called before the callback operation takes place. This will be useful for passing parameters from the client side, which will be accessible in the server side events like RowInserting, RowUpdating and RowDeleting Events by **e.inputparameters**.

OnClientExpand: Specify the function name that will be called before the callback operation for Expanded Grid takes place. This function will be used to pass the parameters from the client side to the server side , which will be accessible in the Expanded Event by using the **e.Parameters** arguments.

OnClientCollapse: Specify the function name that will be called before the callback operation of collapsing the Grid takes place. This function will be used to set the values of the column in the parent Grid using the child grid column values while collapsing.

Type: By default, this value is Normal. For Nested grid, this should be Mergeable.

UseCachedDatasource: By Default,the value is False. This value should be true to maintain the datasource in the session in the control level itself.

ValidationGroup: This will be useful when you need to validate only some controls in the page not for all. Validation will be done only for a particular group.

IflexGridFieldCollection:

IFlexGridField:

DataField: Column Name in the Datasource whose value will be display in the cell.

HeaderText: Sets or Gets the text to display in the Header row for every column in the grid.

ItemStyle: You need to set the width of the fields so that it should be compatible with the editable control width.

IsEditable: Gets or Sets a value indicating whether the iFlexgrid Columns is editable or not.

Value - true if the column should be allowed to edit: otherwise, false. The default is true.

Note: iFlexgridField.Readonly shall override this property, if the ReadOnly property has the value true.

Visible: By Default, the value is False. This is used for the column to be visible in the grid.

ReadOnly: By Default, the value is False. This value should be false in order to edit the columns during runtime.

1. **iTextBoxField**
2. **iLookupField**
3. **iDateField**
4. **iHyperLinkField**
5. **iContainerField**
6. **iCheckBoxField**
7. **iExpandedField**

The properties need to set for every editable control in the grid in the same way as for normal input control except some of the properties that will be different for editable control in the grid are listed here:

iLookupField:

LookupColumnCollection:

This collection is used for setting the values related to lookupcolumns.

ColumName: Column value to be displayed from the lookup table. We need to set the columnName.

ControlToBind: To bind the related column to some other lookup control. We need to specify the lookup Control Id.

TobeHidden: Default value is **False**. To hide the column to be displayed in the grid, we need to set it as **True**.

DependentColumnCollection:

ColumnName: Specify the columnName. This column name is related to the child table.

DependentColumnName: Specify the columnName. This column name is related to the dependent parent table.

LookupName: Specify the Dependent CellIndex.

DependentChildControls: Specify the Dependent child controls (CellIndex) with separated by commas.

PrimaryDataField: Gets or sets the primary column name of the parent table.

ForeignDataField: Gets or sets the foreign column name of the child table.

Validator:

CompareValidation:

ControlToCompare: Gets or sets the cell Index to compare with the input control being validated.

iDateField:

HTMLEncode: By Default, Value is True. This value should be false in order to set the date in a particular format.

DateFormatString: Gets or sets the date Format of the value to be displayed in the grid. The value should be **{0:dd-MMM-yyyy},**

iHyperLinkfield:

DataTextField: Column Name in the Datasource whose value will be displayed in the cell.

iExpandedField:

GridId: Set the iFlexGrid control Id that will be nested with this grid.

Client Side Methods:

Get Current RowIndex of the Grid:

Var rIndex = GridObj.CurrentRowIndex();

This method is used to return the current row Index.

Get Current VirtualRowIndex of the Grid:

Var rIndex = GridObj.*VirtualCurrentRowIndex *();

This method is used to return the virtual row Index of the current row.

Get the Column values from the Datasource:

Var cols = GridObj.Rows(RIndex).Columns()

To access the values in the Datasource, access it by **cols[colIndex]**

RIndex – *refers to the row Index.*
colIndex – refers to the cell Index.

In case of Lookups,

For Ex: *Fourth column in the Grid has Lookup control, then to get the values of lookup control will be*
Cols [3] – *refers to the Id of the lookup.*
Cols [4] – *refers to the Name of the lookup.*
Cols [5] – *refers to the value in the Fifth column of the Grid.*

Note: Here, it is only possible by get the values only by its column Index not by Names.

Get the Client Column values from the Grid:

Var cols = GridObj.Rows(RIndex).GetClientColumns()

RIndex – refers to the row Index.

To access the first value in the Grid, access it by **cols[0].**

Note: Here, it is only possible by get the values only by its column Index not by Names.

Set the Client Columns of the Grid:

GridObj.Rows(RIndex).*SetColumnValuesByIndex *(colIndex,value)

RIndex – refers to the row Index.
colIndex – refers to the cell Index.

This method is used to set the client columns in the grid.

Set ReadOnly for Grid fields from Client side:

SetReadOnlyColumn : This method is used to set readonly to client columns in the grid which accepts two parameters , cellIndex and Boolean value.

Code Snippet:

GridObj.Rows(RIndex).*SetReadOnlyColumn *(colIndex,bolValue)

RIndex – refers to the row Index.
colIndex – refers to the cell Index.

Get the RowState of the client side in the Grid:

Var rowState = GridObj.ClientRowState();

*This method returns '**Inserted'** if the mode is in Add, '**Modified'** if the mode is in 'Edit' and '**Deleted'** if it is Delete mode.*

Set the Parameters for Expanded Grid:

Set the Function Name to the property '**OnclientExpand'** of the grid to be called in order to pass the parameters from the client side to the expanded Grid.

For Ex: The property '**OnclientExpand'** is set as **ExpandParameter**

You need to set the parameters in the client side method as follows:

```
function ExpandParameter(rIndex,param)
{
        var cols = IFlexGrid1.Rows(rIndex).Columns();
        param.Add("TypeId",cols[1]);
}
```

You can access the parameters in the server side events '**Expanded'** by using **Parameters** of the Event Arguments.

For Ex: ***strCNo = e. Parameters("TypeId")***

Pass the parameters from client side to Server-Side:

Set the Function Name to the property '**OnBeforeCallback'** of the grid, which will be called before the callback operation takes place.

For Ex: The property '**OnBeforeCallback'** is set as **Before*CB***

This Function '**Before*CB'*** *will be called with* ***parameters as*** **param** object and **mode.**

Code Snippet:

```
function BeforeCB(mode, param)
{
        if(mode == 'Insert')
        {
                param.add("ContainerNumber","CRUX2000005");
}
}
```

You can access the parametes in the server side events **like rowInserting, rowUpdating, rowDeleting** Events by using **InputParameters** of the Event Arguments.

For Ex: ***strCNo = e.InputParameters("ContainerNumber")***

Pass the parameters from Server-side to Client-Side :

Set the Function Name to the property '**OnAfterCallback'** of the grid, which will be called after the callback operation takes place.

You need to set the parameters in the server side events **like rowInserting, rowUpdating, rowDeleting** Events by using **OutputParameters** of the Event Arguments.

For Ex: ***e.OutputParameters("ContainerSize") = '20'***

This parameter will be passed to the function as arguments, which will be called after the callback operation takes place.

For Ex: The property '**OnAfterCallback'** is set as ***AfterCB***

You can access the parameters in this way:

```
function AfterCB(param)
{
var Size = param["ContainerSize"];
}
```

Submit Method:

This method is used to save the data in the datasource before committing the data in the database. This will return the status of the grid. This method should be called first in the Submit button click event.

For Ex: var status = GridObj.Submit();

Client Bind Grid:

Set the property 'Type' as Normal.

You need to set the parameters in any client side method as follows:

```
function ExpandParameter()
{
        var param = new ClientiFlexGrid ("iFlexGrid1");
        param.parameters.add(obj.id,obj.value);

param.DataBind();
}
```

You can access the parametes in the server side events '**ClientBind** by using **Parameters** of the Event Arguments.

For Ex: ***strCNo = e. Parameters("iLookup1")***

You need to set the datasource property to the client bind grid as follows:

```
e.DataSource = ds.Tables(0)
```

Server side Events:

Row Inserting Event :

This event will be fired before the insertion takes place. You can access the field values using **e.values(column Name) .**

Before insetion, you need to set the datakey(primary Key) value in this event itself.

Row Inserted Event :

This event will be fired after the insertion takes place. If any exception occurs during the insertion, You can get the exception using **e.Exception** in this event. If you want to cancel the insertion of records, You need to set the **e.Cancel** as **True.**

Row Updating Event :

This event will be fired before the updation takes place. You can access the field values using **e.NewValues(column Name).** If you want to cancel the updation of records, You need to set the **e.Cancel** as **True and e.RequiresRebind** as **true.**

Row Updated Event :

This event will be fired after the insertion takes place. If any exception occurs during the insertion, You can get the exception using **e.Exception** in this event.

Row Deleting Event :

This event will be fired before the Deletion takes place.

Row Deleted Event :

This event will be fired after the Deletion takes place. If any exception occurs during the Deletion, You can get the exception using **e.Exception** in this event.

iTab Control

Features:

It is a custom control, that displays set of controls.

Properties:

TabPageClientId: It should have id of the Div control which contains set of controls.

DoValidate: To perform validations, this value should be true. **:** The Default value is False.

OnAfterSelect: Function name that will be called when the tab is selected.

OnBeforeSelect: Function name that will be called before the tab is selected.

Configuration File:

Values to be included in configuration file in appsettings section:

ConnectionString – ConnectionString value, which will be used to connect to the database.

LkpChooseTable - To get the data using choosetable logic . This value should be true. The Default value is True.

LkpKeyCode - keyCode to start search operations . Default is 40(Down Arrow Key)

ScriptsFolder – To set the path of the script folder in the application.

ImagesFolder - To set the path of the Images folder in the application.

LgCSS – Css Class Name of the Grid.

LgHdrCss - Css Class Name of the Grid Header style.

LgItmCss - Css Class Name of the Grid Item style.

LgSelItmCss - Css Class Name of the Grid SelectedItem style.

LgPgCss – Css Class Name for the Grid Pager style.

TraceReport – This value should be true. This is used for trace Report.

ApplyDefaultStyle - This value should be true. This is used to apply the Default style.

Setup Files:

Once you had installed the setup files, the needed script files will be got from the scripts folder of the installed directory. You have to include these files to make the controls to work.