Editable Grid PCF Documentation

Overview

The **Editable Grid PCF** enhances a standard Dynamics 365 subgrid by enabling inline editing, notifications, and dynamic filtering on the form itself. Users can update multiple rows without opening record forms, improving data-entry speed and UX.

Features

- **Inline Editing**: Edit any cell directly in the grid.
- **Field-Level Notifications**: Show warnings/errors on individual cells.
- **Lookup Filtering**: Apply and remove presearch filters on lookup columns.
- **Event Model**: Hook into grid lifecycle events (OnLoad, OnChange, OnSave, OnNew).

Concepts

1. **BeverControls**: Global helper for locating PCF controls (e.g. getEditableGrid).
2. **EditableGrid**: root object representing the editable grid; allows event handlers to be attached and data to be refreshed.
3. **Row**: Represents one row; allows retrieval of its cells.
4. **Cell**: Represents one field in a row; supports get/set value, notifications, disable/enable, setting required level, and lookup filter.

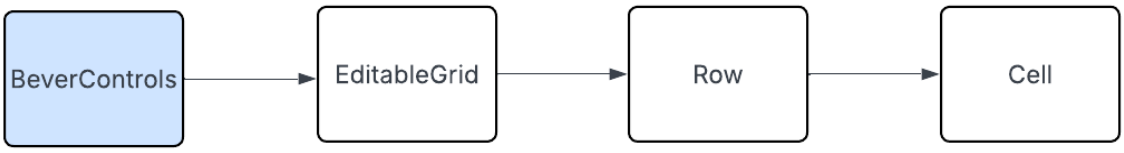


Table of Contents

BeverControls

The BeverControls object serves as the central entry point for accessing PCF controls on a Dynamics 365 form. It is primarily used to locate and return custom controls by schema name, enabling immediate event handler attachment or data manipulation.

Method	Signature	Description
getEditableGrid (name: string): Promise<EditableGrid>	(name: string) => Promise<EditableGrid>	Returns the editable grid instance matching the given name.

Code Example

```
// Using BeverControls to get the editable grid object.
async function Init() {
  const editableGrid = await BeverControls.getEditableGrid("invoice_lines");
  //Check that editableGrid object is not null
  if (editableGrid != null) {

    }

}
```

Important: The getEditableGrid function will return null on new record forms, as the editable grid is not available in that context. It may also return null if the grid is still loading. Developers must check that editableGrid is not null before using it.

EditableGrid

The **EditableGrid** object represents the PCF grid instance on the form. It offers hooks and actions for managing row cells and linking functions to grid events.

Responsibilities

- Associate event handlers with grid events such as load, save, new row, and cell value changes.
- Retrieve all rows or a specific row by ID.
- Provide function to refresh the grid.

Properties

Property Name	Description
gridId	Holds the unique identifier of the subgrid.
rows	Represents an array of row objects associated with the EditableGrid object.

Methods

Method	Description
addOnLoad**(callback: (context: GridContext) => void): void**	Registers a handler function that is triggered when the grid loads on the form.
removeOnLoad**(callback: (context: GridContext) => void): void**	Removes a handler function from the OnLoad event.
addOnChange**(fieldName: string, callback: (context: GridContext) => void): void**	Registers a handler function that will be called each time any cell in the specified column is edited.
removeOnChange**(fieldName: string, callback: (context: GridContext) => void): void**	Removes a handler function from the OnChange event.
addOnSave**(callback: (context: GridContext) => void): void**	Registers a handler function that is triggered when the grid is saved.
removeOnSave**(callback: (context: GridContext) => void): void**	Removes a handler function from the OnSave event.
addOnNew**(callback: (context: GridContext) => void): void**	Registers a handler function that is triggered when a new row is added to the grid.
removeOnNew**(callback: (context: GridContext) => void): void**	Removes a handler function from the OnNew event.
refresh**(): void**	Reload grid data from the server.
getRows()**: Row[]**	Retrieves all rows linked to the Editable Grid.

getRow**(rowId:string):Row**	Retrieves a specific row from the Editable Grid using the provided rowId, which is the unique ID of the row (same as the Dynamics 365 record ID)

addOnLoad

Description

Registers a handler function that is triggered when the grid loads on the form.

Parameters

Callback: A function that receives a GridContext object. Use context.data.table.rows to access all rows.

Syntax

```
addOnLoad(callback: (context: GridContext) => void): void
```

Code Example

```
// After retrieving the grid, use addOnLoad to set default text
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnLoad(context => {
  // Loop through each row when grid data is loaded
  context.data.table.rows.forEach(row => {
    const cell = row.getCell("new_name");
    await cell.setValue("Default Name");
  });
});
```

Method Response

```
{
   gridId: string
   eventName: string,
   data: {
      table: EditableGrid
   }
}
```

removeOnLoad

Description

Removes a handler function from the OnLoad event.

Parameters

Callback: The same function reference that was passed to addOnLoad.

Syntax

```
removeOnLoad(callback: (context: GridContext) => void): void
```

Code Example

```
// Handler definition
function GridOnLoad(ctx) {
  console.log("Grid has loaded");
}

// Attach the handler
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnLoad(GridOnLoad);

// Remove the handler
grid.removeOnLoad(GridOnLoad);
```

addOnChange

Description

Registers a handler function that will be called each time any cell in the specified column is edited.

Parameters:

- fieldName: schema name of the column to watch
- callback: function receiving GridContext when any cell in the specified column is edited

Syntax

```
addOnChange(fieldName: string, callback: (context: GridContext) => void):
void
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnChange("new_name", context => {
  const row = context.data.row;
  const cell = row.getCell("new_name");
  if (cell.getValue() == null) {
    cell.setNotification("Cannot be empty", 201);
  }
});
```

Method Response

```
{
   gridId: string
   eventName: string,
   data: {
      table: EditableGrid
      row: Row
      cell: Cell
   }
}
```

removeOnchange

Description

Removes a handler function from the OnChange event.

Parameters:

- fieldName: schema name of the column for which the change handler should be removed.
- callback: the same function reference that was passed to addOnChange.

Syntax

```
removeOnChange(fieldName: string, callback: (context: GridContext) => void):
void
```

Code Example

```
// Define the handler
function onNameChange(context) {
const row = context.data.row;
  const cell = row.getCell("new_name");
  if (cell.getValue() == null) {
    cell.setNotification("Cannot be empty", 201);
  }
}

// Attach the handler
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnChange("new_name", onNameChange);

// Remove the handler
grid.removeOnChange("new_name", onNameChange);
```

addOnSave

Description

Registers a handler function that is triggered when the grid is saved.

Parameters

Callback: function receiving a GridContext object when the grid's Save event occurs.

Syntax

```
addOnSave(callback: (context: GridContext) => void): void
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnSave(ctx => {
  ctx.data.table.rows.forEach(row => {
    const cell = row.getCell("new_name");
    // Other records update logic by using API
  });
});
```

Method Response

```
{
   gridId: string
   eventName: string,
   data: {
      table: EditableGrid
   }
}
```

removeOnSave

Description

Removes a handler function from the OnSave event.

Parameters

Callback: the same function reference that was passed into addOnSave

Syntax

```
removeOnSave(callback: (context: GridContext) => void): void
```

Code Example

```
function onSave(ctx) {
  console.log("Grid saved");
}

// Attach the handler
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnSave(onSave);

// Remove the handler
grid.removeOnSave(onSave);
```

addOnNew

Description

Registers a handler function that is triggered when a new row is added to the grid.

Parameters

Callback: function receiving a GridContext when the user inserts a new row.

Syntax

```
addOnNew(callback: (context: GridContext) => void): void
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnNew(ctx => {
  const newRow = ctx.data.row;
  const cell = newRow.getCell("new_name");
  if (cell) {
    await cell.setValue("Auto-init");
  }
});
```

Method Response

```
{
   gridId: string
   eventName: string,
   data: {
      table: EditableGrid
      row: Row
   }
}
```

removeOnNew

Description

Removes a handler function from the OnNew event.

Parameters

Callback: the same function reference that was passed into addOnNew.

Syntax

```
removeOnNew(callback: (context: GridContext) => void): void
```

Code Example

```
function onNewRow(ctx) {
  console.log("A new row was added");
}

// Attach the handler
const grid = await BeverControls.getEditableGrid("editable_grid");
grid.addOnNew(onNewRow);

// Remove the handler
grid.removeOnNew(onNewRow);
```

refresh

Description

Reloads the grid's data from the server, discarding any unsaved in-memory changes.

Syntax

```
refresh(): void
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
// Refresh the grid
grid.refresh();
```

getRows

Description

Retrieves all rows linked to the Editable Grid.

Syntax

```
getRows(): Row[]
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
let rows = editableGrid.getRows();
```

getRow

Description

Retrieves a specific row from the Editable Grid using the provided rowId, which is the unique ID of the row (same as the Dynamics 365 record ID)

Syntax

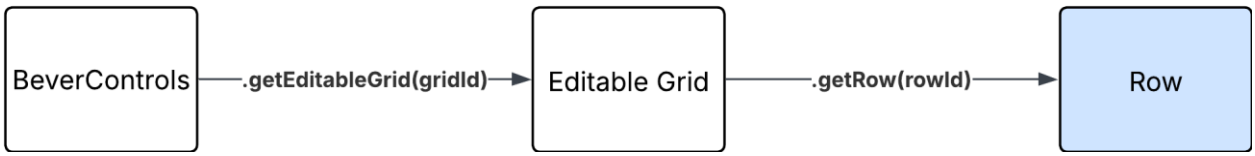
```
getRow(rowId: string): Row | null
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
let row = editableGrid.getRow("d74ffe77-084d-f011-877a-7c1e5277a5f9");
```

Row

Represents a single record in the editable grid. A row provides access to its cells and metadata, allowing inspection or modification of values on a per-record basis.



Properties

Property Name	Description
rowId	Represents the unique ID of the row (same as the Dynamics 365 record ID).
cells	Represents an array of cell objects associated with the row object.

Methods

Method	Description
getCell**(fieldName: string): Cell | null**	Gets a specific cell by its schema name.

getCell

Description

Returns the cell object for the given column in the row, or null if that field isn't present on the grid.

Parameters

FieldName: schema name of the column to be accessed.

Syntax

```
getCell(fieldName: string): Cell | null
```

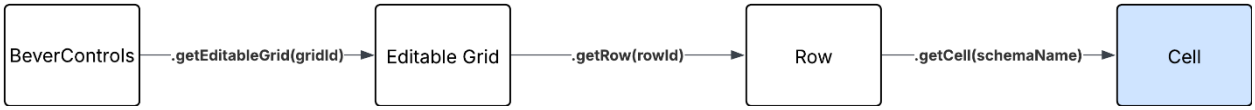
Code Example

```
grid.addOnLoad(ctx => {
  ctx.data.table.rows.forEach(row => {
    const nameCell = row.getCell("new_name");
    if (nameCell) {
      nameCell.setNotification("Custom Error", 500);
    }
```

```
  });
});
```

Cell

Represents a single field in a row.



Properties

Property Name	Description
gridId	Holds the unique identifier of the subgrid.
isDisabled	Indicates whether the cell is disabled for editing.
isRequired	Indicates whether the cell is mandatory.
rowId	Represents the unique ID of the cell's related row, which is the same as the Dynamics 365 record ID.
schemaName	Represents the unique name of the cell (schema name of the column)
type	Indicates the Dynamics 365 field type of the cell, such as 'SingleLine.Text', 'OptionSet', or 'Currency'.
value	Stores the value of the cell field.

Methods

Method	Description
setValue**(value: any): Promise<void>**	Sets the value in the cell.
getValue**(): any**	Gets the value from the cell.
setDisabled**(isDisabled: boolean): void**	Specifies whether or not the cell should be disabled.
getDisabled**(): boolean**	Gets the information on whether or not the cell is disabled.

setRequiredLevel**(level: "none " | "required"): void**	Specifies whether or not the cell should be mandatory.
getRequiredLevel**(): "none" | "required"**	G ets the information on whether or not the cell is mandatory.
addPresearch**(fetchXml: string): void**	Applies a filter to the lookup cell.
removePresearch**(): void**	Removes a filter from the lookup cell.
setNotification**(message: string): void**	Sets the message notification for the cell.
clearNotification**(): void**	Clears the message notification from the cell.
getType**(): string**	Gets the Dynamics 365 field type of the cell, such as 'SingleLine.Text', 'OptionSet', or 'Currency'.
getEditableGridId**(): string**	Gets the unique identifier of the subgrid.

setValue

Description

Asynchronously sets the value in the cell.

Parameters

Value: any — the new value to assign to the cell.

Note: setValue() returns a Promise—cell.setValue(...) must be awaited to reliably retrieve the updated value using cell.getValue()."

Syntax

```
setValue(value: any): Promise<void>
```

Code Example

```
const cell = row.getCell("new_name");
await cell.setValue("New text");
console.log(cell.getValue());
```

getValue

Description

Returns the cell's current value.

Syntax

```
getValue(): any
```

Code Example

```
const cell = row.getCell("new_name");
const current = cell.getValue();
console.log(current);  // logs whatever the cell holds
```

setDisabled

Description

Toggles the cell's editable state. When disabled, the user cannot change its value.

Parameters

IsDisabled: boolean — pass true to make the cell read-only, false to allow editing.

Syntax

```
setDisabled(isDisabled: boolean): void
```

Code Example

```
const cell = row.getCell("new_name");
// Disable editing
cell.setDisabled(true);
// Enable editing
cell.setDisabled(false);
```

getDisabled

Description

Returns true if the cell is currently disabled (read-only), or false if it is editable.

Syntax

```
getDisabled(): boolean
```

Code Example

```
const cell = row.getCell("new_name");
if (cell.getDisabled()) {
  console.log("This cell cannot be edited");
}
```

setRequiredLevel

Description

Sets the cell's requirement status, updating its visual indicator and enforcing form-level validation rules accordingly.

Parameters

Level: required/none

Syntax

```
setRequiredLevel(level: "none " | "required"): void
```

Code Example

```
const cell = row.getCell("new_name");
// Mark as required
cell.setRequiredLevel("required");
// Mark as optional
cell.setRequiredLevel("none");
```

getRequiredLevel

Description

Retrieves the cell's current requirement level, indicating if it's optional ("none"), suggested ("recommended"), or mandatory ("required").

Syntax

```
getRequiredLevel(): "none" | "required"
```

Code Example

```
const cell = row.getCell("new_name");
const level = cell.getRequiredLevel();
console.log(level);  // e.g., "required"
```

addPresearch

Description

Applies a presearch filter to a lookup cell. The next time the user opens the lookup dialog, only records matching the filter will appear.

Parameters

FetchXml: string — an XML <filter> fragment defining the lookup filter to apply (e.g. conditions, values).

Note: If providing OData string, "&$filter=" should be included.

If providing FetchXml string, "<filter type='and/or'>" and "</filter>" tags should be included.

Syntax

```
addPresearch(fetchXml: string): void
```

Code Example

```
const cell = row.getCell("lookup_field");
cell.addPresearch(
  `<filter type="and">
    <condition attribute="new_name" operator="eq" value="Example Name" />
  </filter>`
);
```

removePresearch

Description

Clears any lookup filter previously applied via addPresearch, restoring the default record list in the lookup dialog.

Syntax

```
removePresearch(): void
```

Code Example

```
const cell = row.getCell("lookup_field");
cell.removePresearch();
```

setNotification

Description

Displays a validation or informational message attached to the cell.

Parameters

- message: string — the text to display on the cell.

Syntax

```
setNotification(message: string): void
```

Code Example

```
const cell = row.getCell("new_name");
// Show a warning
cell.setNotification("Value must be greater than zero");
```

clearNotification

Description

Removes the notification from the cell, clearing its message and any visual indicator.

Syntax

```
clearNotification(): void
```

Code Example

```
const cell = row.getCell("new_name");
// Show a notification
cell.setNotification("Please enter a valid value");
// Later—clear that notification
cell.clearNotification();
```

getType

Description:

Returns the cell's underlying data type (e.g. 'SingleLine.Text', 'OptionSet', or 'Currency').

Syntax

```
getType(): string
```

Code Example

```
const cell = row.getCell("new_name");
console.log(cell.getType());  // e.g. "String"
```

getEditablegridid

Description:

Retrieves the unique identifier of the editable-grid control instance on the form.

Syntax

```
getEditableGridId(): string
```

Code Example

```
const grid = await BeverControls.getEditableGrid("editable_grid");
const gridId = grid.getEditableGridId();
console.log("Grid control ID:", gridId);
```