in Telerik's Kendo Grid for server binding. Instead, you need to use the transport option to send additional data to the server, such as the RebuiltStockNum.

In this case, you should use the transport's ParameterMap to pass additional query string parameters or custom client-side logic to attach parameters.

**Correct Approach to Pass RebuiltStockNum Using ParameterMap**

Here's the correct way to configure the Update, Read, and Destroy actions with query string parameters using ParameterMap.

**Updated Grid Configuration in Index.cshtml**

csharp

CopyEdit

@(Html.Kendo().Grid<RebuiltPartsViewModel>()

.Name("RebuiltPartsGrid")

.Columns(columns =>

{

columns.Bound(c => c.RebuiltStockNum)

.Title("Rebuilt Stock Code")

.Filterable(ftb => ftb.Cell(cell => cell.Operator("contains").SuggestionOperator(FilterType.Contains)));

columns.Bound(c => c.ListOfBus)

.Title("Vehicle Series")

.ClientTemplate("# if (ListOfBus && ListOfBus.length > 0) { # #= ListOfBus.map(bus => bus.Description).join(', ') # # } else { # - # } #");

columns.Command(command =>

{

command.Edit().Text("Edit");

command.Destroy().Text("Delete");

}).Title("&nbsp;").Width(150);

})

.Editable(editable => editable.Mode(GridEditMode.InLine)) // Inline editing

.Pageable()

.Sortable()

.Scrollable()

.Filterable()

.DataSource(dataSource => dataSource

.Ajax()

.Read(read => read.Url(Url.Action("GetRebuiltPartsView", "RebuiltParts")).Type(HttpVerbs.Get))

.Update(update => update.Url(Url.Action("UpdateRebuiltPart", "RebuiltParts")).Type(HttpVerbs.Post))

.Destroy(destroy => destroy.Url(Url.Action("DeleteRebuiltPart", "RebuiltParts")).Type(HttpVerbs.Post))

.Model(model =>

{

model.Id(p => p.RebuiltStockNum); // Specify the primary key

model.Field(p => p.ListOfBusIds).DefaultValue(new List<int?>());

})

.Transport(transport =>

{

transport.ParameterMap("addRebuiltStockNum"); // Attach the custom parameter map function

})

)

)

<script>

function addRebuiltStockNum(data) {

// Attach RebuiltStockNum from the selected row

if (data.models && data.models.length > 0) {

const rebuiltStockNum = data.models[0].RebuiltStockNum;

data.RebuiltStockNum = rebuiltStockNum; // Pass it in the request

}

return data;

}

</script>

**Explanation of Changes**

1. **Custom ParameterMap**:
   * The .Transport(transport => transport.ParameterMap("addRebuiltStockNum")) is used to pass additional parameters from the client to the server for Update, Destroy, and other CRUD operations.
2. **addRebuiltStockNum Function**:
   * This JavaScript function extracts the RebuiltStockNum from the currently edited row and attaches it to the request payload.
3. **Inline Editing**:
   * For inline editing, we directly extract the RebuiltStockNum from the data.models array (since it represents the row being updated).

**Updated Controller: RebuiltPartsController**

You need to update the UpdateRebuiltPart and DeleteRebuiltPart actions to read the RebuiltStockNum from the request payload.

**Controller Code:**

csharp

CopyEdit

[HttpPost]

public async Task<IActionResult> UpdateRebuiltPart(string RebuiltStockNum, [FromBody] RebuiltPartsViewModel model)

{

try

{

if (string.IsNullOrEmpty(RebuiltStockNum) || model == null)

{

return BadRequest("Invalid data.");

}

// Update the RebuiltPart in the database

var updatePartSql = @"

UPDATE SBCES.RbMasterlist

SET ... -- Add fields to update here

WHERE RebuiltStockNum = @RebuiltStockNum";

await \_dbConnection.ExecuteAsync(updatePartSql, new { RebuiltStockNum });

// Update associated buses

var deleteOldBusesSql = "DELETE FROM SBCES.RBLISTOFBUSES WHERE RebuiltStockNum = @RebuiltStockNum";

await \_dbConnection.ExecuteAsync(deleteOldBusesSql, new { RebuiltStockNum });

if (model.ListOfBusIds?.Any() == true)

{

var insertNewBusesSql = @"

INSERT INTO SBCES.RBLISTOFBUSES (RebuiltStockNum, ListId)

VALUES (@RebuiltStockNum, @ListId)";

foreach (var listId in model.ListOfBusIds.Where(id => id.HasValue))

{

await \_dbConnection.ExecuteAsync(insertNewBusesSql, new { RebuiltStockNum, ListId = listId.Value });

}

}

return Json(new { success = true });

}

catch (Exception ex)

{

return StatusCode(500, $"Internal server error: {ex.Message}");

}

}

[HttpPost]

public async Task<IActionResult> DeleteRebuiltPart(string RebuiltStockNum)

{

try

{

if (string.IsNullOrEmpty(RebuiltStockNum))

{

return BadRequest("Invalid data.");

}

var deletePartSql = "DELETE FROM SBCES.RbMasterlist WHERE RebuiltStockNum = @RebuiltStockNum";

var deleteBusesSql = "DELETE FROM SBCES.RBLISTOFBUSES WHERE RebuiltStockNum = @RebuiltStockNum";

await \_dbConnection.ExecuteAsync(deleteBusesSql, new { RebuiltStockNum });

await \_dbConnection.ExecuteAsync(deletePartSql, new { RebuiltStockNum });

return Json(new { success = true });

}

catch (Exception ex)

{

return StatusCode(500, $"Internal server error: {ex.Message}");

}

}

**How It Works:**

1. **ParameterMap**:
   * The addRebuiltStockNum function extracts the RebuiltStockNum for the selected row being updated or deleted.
   * This function attaches RebuiltStockNum to the request payload, ensuring it is sent to the server as part of the request.
2. **Inline Editing**:
   * When you edit a row, the RebuiltStockNum for that row is automatically passed to the server along with other updated values.
3. **Exception Handling**:
   * Proper exception handling ensures meaningful error responses are sent to the client if something goes wrong.

**Summary of Files:**

1. **Model**: Includes RebuiltPartsViewModel and BusesModel.
2. **Controller**: Includes the GetRebuiltPartsView, UpdateRebuiltPart, and DeleteRebuiltPart actions with proper query string handling.
3. **Grid Configuration**:
   * Uses ParameterMap to attach RebuiltStockNum to the request.
   * Includes inline editing with update and delete commands.
4. **JavaScript**: Implements the addRebuiltStockNum function to dynamically add query string parameters.

This should fix the issue and allow the RebuiltStockNum to be passed correctly in the Read, Update, and Delete operations for inline editing. Let me know if you encounter further issue