|  |  |
| --- | --- |
|  | **Unidev** |
|  | **Unified Development, Inc.**  16690 Swingley Ridge Road  Suite 260  Chesterfield, MO 63017  (636) 532-4424 |

|  |
| --- |
|  |
|  |

**Duncan Solutions, Inc**.

Grid Management

December 2013

Revision 1.00

**Prepared by:**

Ron Howard

Caleb Miller

Unified Development

**Table of Contents**

[1. Overview 4](#_Toc374445382)

[1.1 Goal 4](#_Toc374445383)

[2. Target Audience 5](#_Toc374445384)

[3. Database 6](#_Toc374445385)

[3.1 Tables 6](#_Toc374445386)

[3.2 Stored Procs 9](#_Toc374445387)

[4. Adding Grid Columns 11](#_Toc374445388)

[4.1 Steps 11](#_Toc374445389)

[4.1.1 Update Master Grid list 11](#_Toc374445390)

[4.1.2 Update the Entity 12](#_Toc374445391)

[4.1.3 Update the MVC View 13](#_Toc374445392)

[4.1.4 Update DB View 14](#_Toc374445393)

[4.1.5 Update Customer Grid Options 15](#_Toc374445394)

[4.1.6 Testing 16](#_Toc374445395)

[5. Adding Grid Filters 18](#_Toc374445396)

[5.1 Steps 18](#_Toc374445397)

[5.1.1 Update the Entity 18](#_Toc374445398)

[5.1.2 Update DB View / Stored Procedure 18](#_Toc374445399)

[5.1.3 Update the MVC View 20](#_Toc374445400)

[5.1.4 Test 23](#_Toc374445401)

**Revision** History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Author | Description | Revision Date |
| 1.00 | Caleb Miller | Initial version | 12/10/13 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Overview

## Goal

The purpose of this document is to explain the concepts and methodologies used when adding or removing columns from the grids within the system. Since the grids have such complicated functionality, adding a column is not as simple as just adding the column to the grid. Re-ordering, re-titling, and hiding columns on a customer specific basis has to be taken into account. For more information on the rules and how that is accomplished, see the Kendo Grids document. This document will cover the following:

1. Database - All of the associated tables and stored procedures that are involved.
   1. Tables
   2. Stored Procs
2. How to add a column –
   1. Rules to follow
   2. Where data is stored
   3. How to update the data retrieval process
3. How to add a filter..

**NOTE: Section 1 & 2 have been made obsolete due an updated table structure and business logic flow for the custom Grids. Refer to the Custom Grid Options document for more information. Section 3 is still valid.**

# Target Audience

The target audience of this document is person or persons who have:

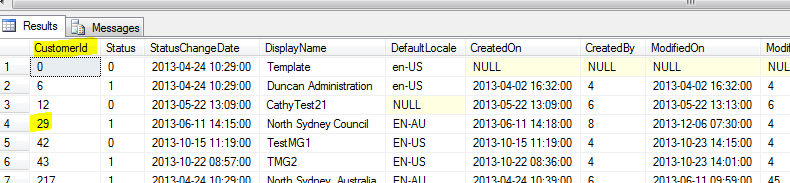
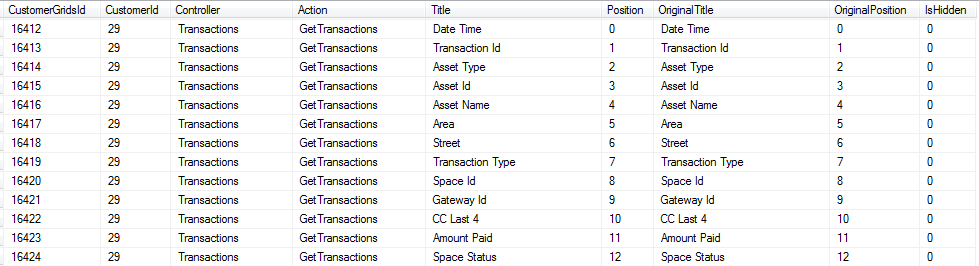
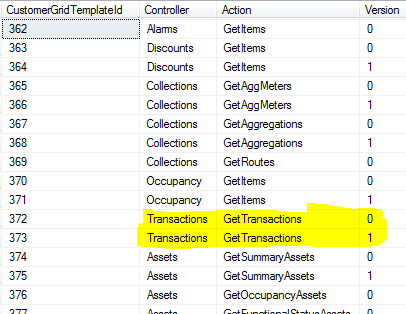
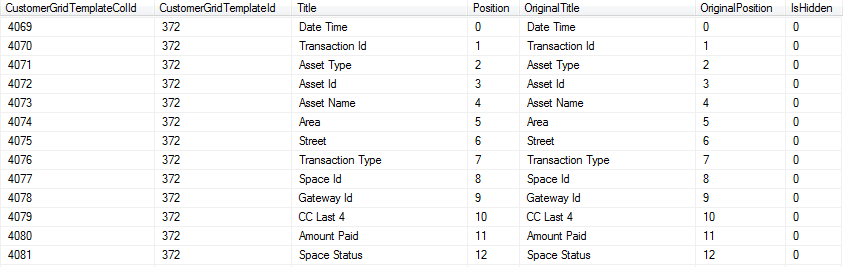
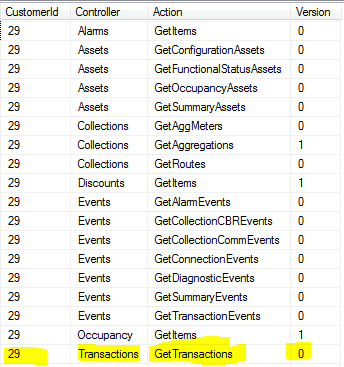
1. Experience in the following technologies:
   1. Javascript / Jquery. This is used heavily on each grid.
   2. C# /.Net / MVC
   3. Working knowledge of Telerik and Kendo MVC
   4. T-SQL, SSMS, SqlServer, Stored Procedures and Views
2. The user of this document has a full understanding of the Duncan PEMS project. This includes database and system architecture knowledge, desired business rules of the application, etc.
3. Microsoft SQL Server administration and understand rights, database creation and administration, and are able to use either SSMS or SQL command line interface. The user will need the ability to create and maintain stored procedures and views.
4. Visual Studio 2012

# Database

The PEMS RBAC database is used for storing all of the data needed. Here is the database diagram:

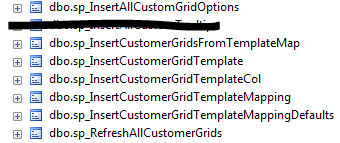


## Tables

1. **Customer Profile**: the table that holds the customers profile information. The grids only use the Customer ID field of this table
   1. 
2. **Customer Grids**: **IMPORTANT**: the table that holds the finalized grid data used for each grid on the site. The grids in the web application will use this table to pull the visibility, title, and order of the columns for each grid in the system. This table is the only table the Kendo grids use. The goal is to input the correct data to this table so the application works correctly. All other tables and stored procs are support items to facilitate keeping this table up to date with the correct data
   1. 
3. **Customer Grid Template**: Each version of the grid consists of a unique controller / action / version. This table holds all of the different versions for each grid in the system.
   1. 
4. **Customer Grid Template Col**: this table represents the columns for a particular template in the Customer Grid Template table. This will be the table that needs to be maintained. When adding columns, this table will be updated and then the data will be re-persisted to all the existing customers with the change based on the templates they are using. Any customer that is using the updated template will have their Customer Grid data updated based on this Template Column table. For Grid Template Id of 372, the columns are:
   1. 
5. **Customer Grid Template Map**: this table maps the Customer Grid Templates to the Customer Profile. When re-generating the Customer Grids for a customer, this table determines what Customer Grid Template should be used for each grid on the site. For North Sydney Council, here is the mapping:
   1. 

## Stored Procs

The following is a list of stored procedures the application uses to maintain all of the grid data in the system.



1. Insert all custom grid options: this is the master list of all grid templates and columns in the system. This will be the main file that needs to be updated when adding version, updating version, adding columns, removing columns, etc. All of the data in the Customer Grid Templates and Template Column tables is removed then fresh data is added based on this stored proc. Any time there is a change to the grid, this is the first file updated, as it is the master file.
2. Insert Customer Grids From Template Map: This removes all of the data from the Customer Grids table for a specific customer and re-inserts new data based on what is in the Customer Grid Template Map table. It is used when saving the custom grids page for a customer, creating a new customer, or adding a column to the grid. The Customer Template Map is retrieved, then for each mapping item, it inserts the associated data from the Custom Grid Template and Columns.
3. Insert Customer Grid Template: Inserts a Customer Grid Template and returns the ID of the newly created grid template. This is called from the “Insert all custom grid options” stored procedure.
4. Insert Customer Grid Template Col: Inserts a template column for a specific grid template. This is called from the “Insert all custom grid options” stored procedure.
5. Insert Customer Grid Template Mapping: Inserts a map between the customer and the template for a specific action / controller / version. It deletes all existing data for that customer / action / controller, and then re-inserts the data with the updated version passed in. This is used when saving the custom grids for a customer. This allows the application to update the customer templates for each grid in the system with a new version.
6. Insert Customer Grid Template Mapping Defaults: Deletes all of the customer template maps for a specific customer, and then gets a distinct list of action / controllers from the Customer Grid Template table. For each distinct set, the Insert Customer Grid Template Mapping stored proc is called using Version = 0. This essentially inserts all of the default data for a customer. This can be used when resetting a customer to the default and when creating a new customer.
7. Refresh all customer grids: This stored proc will run the Insert All Custom Grid Options, and once all of those are setup correctly, will roll through the customers in the system and re-insert the appropriate data in the customer template mapping. If they do not have any template mapping setup for a specific action / controller, it will create one using the Version 0. This allows us to easily add custom grid options to the system and push that data to all of the existing customers. Once all customers have a map for each controller / action, the Customer Grid data is updated

# Adding Grid Columns

This section will use the Transactions grid as an example and a field called “Time Paid” is being added directly after the “AmountPaid” field. Since the system has to perform certain functional requirements, the following process must be used to add a column to a grid. This applies to Complex grids (see the Kendo Grids document) that have re-ordering, re-titling, or hidden columns.

## Steps

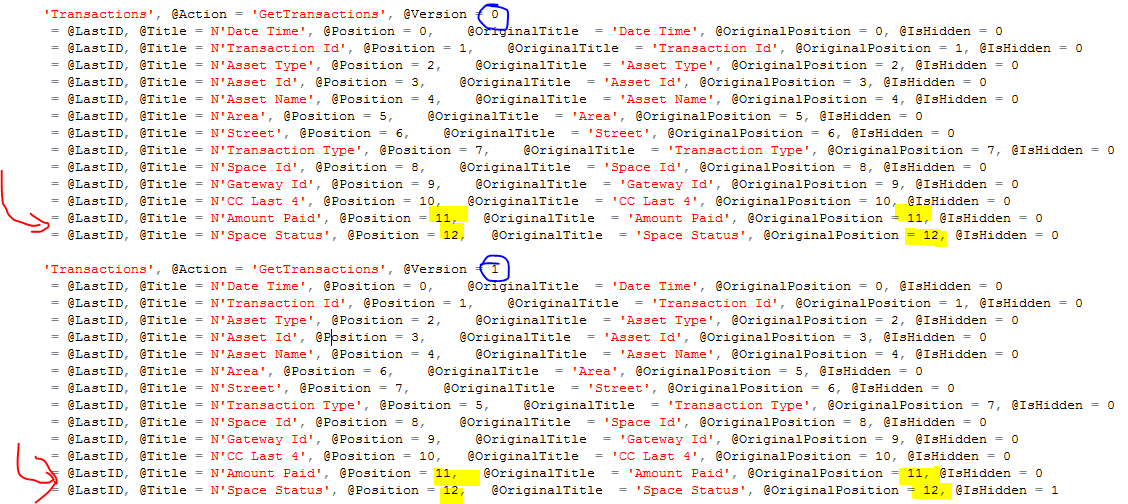
These are the steps that will need to be taken in order to maintain data consistency and successfully add a column to an existing grid.

### Update Master Grid list



The first step is to update all versions within the master list to include the new column. This list resides within the sp\_InsertAllCustomGridOptions stored procedure. The field has to be added in the correct spot with the valid Original Position and Position values.

Before:



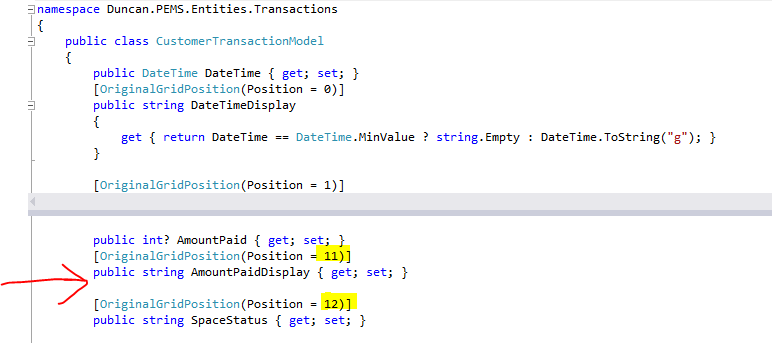
After:



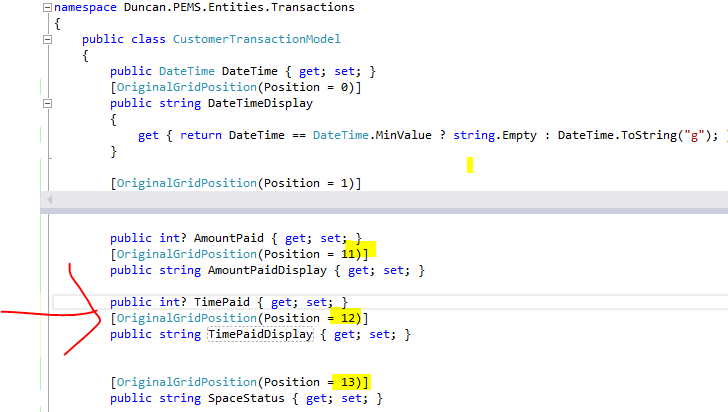
### Update the Entity

The CustomerTransactionModel is the class that needs to be updated that represents the Grid data on the Transactions Index page. A field called “TimePaid” will be added. A Display version of this field will be added in order to correctly format the content on the site. The new field will be added right after the “AmountPaid” field. When adding a field, it is important to make sure the OriginalGridPosition attribute of the property correctly lines up with where we add it to all versions in the MasterGridList. In this example, we will be adding the new field, and updating Space Status’s OriginalGridPosition to be 13, in order to match the changes made above.

Before:



After:



### Update the MVC View

The Kendo grid will need to be updated to display the value to the user:

Before:

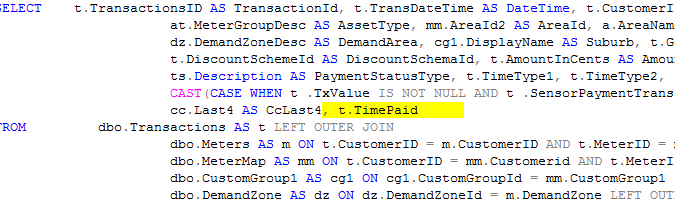


After:

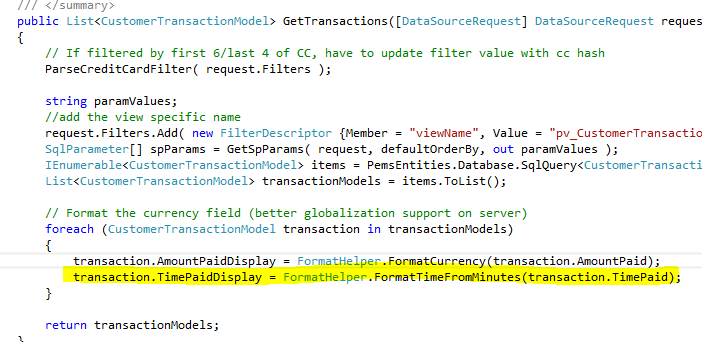


### Update DB View

The data access method must be updated to populate our new property correctly (TimePaid). This example uses the stored procedure / view method, so the view in this case is pv\_CustomerTransactions. This will need to be updated to return the data requested. In this case t.TimePaid was added



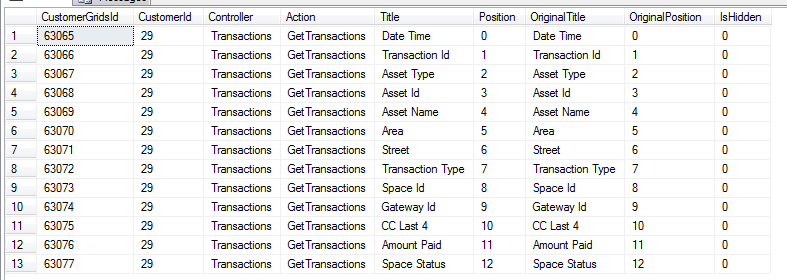
Since the data returned needs to be formatted before it is displayed to the user, it is formatted as well:



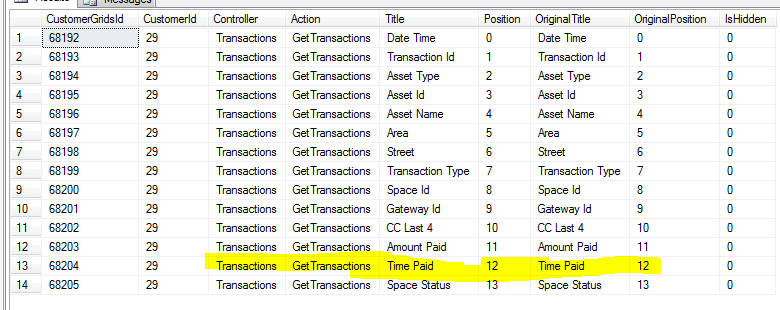
### Update Customer Grid Options

Now that the grids and their associated data access are updated, the data within the Customer Grids table needs to be updated to reflect the new column. This is done by running the sp\_RefreshAllCustomerGrids stored procedure within the RBAC DB. This will update each customers Customer Grid data to include the new field in the appropriate location for display on the site.

Here is a snapshot of the Transactions data for customer 29 before running the sp\_RefreshAllCustomerGrids:



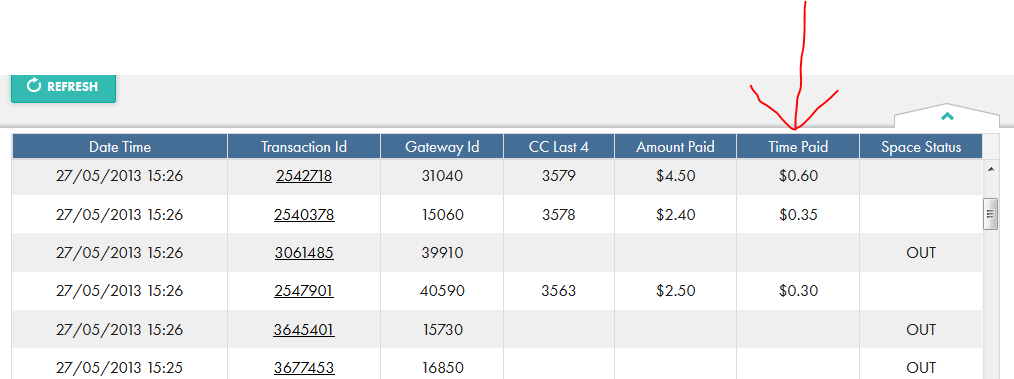
After:



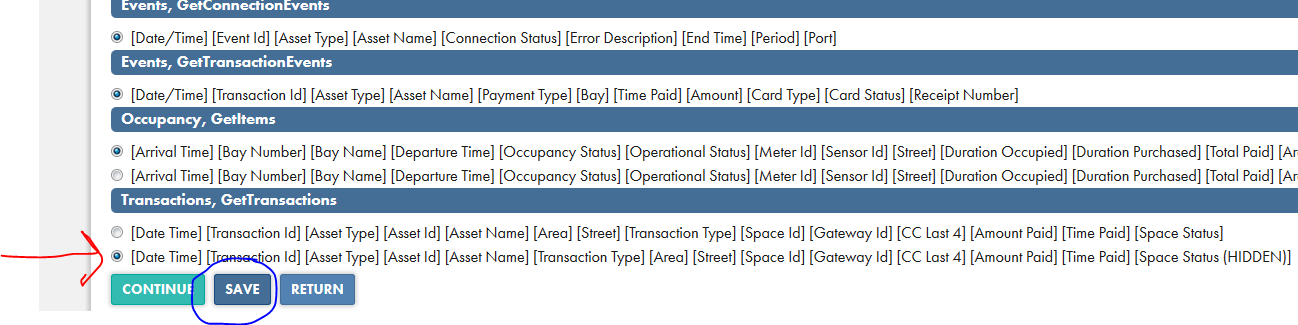
### Testing

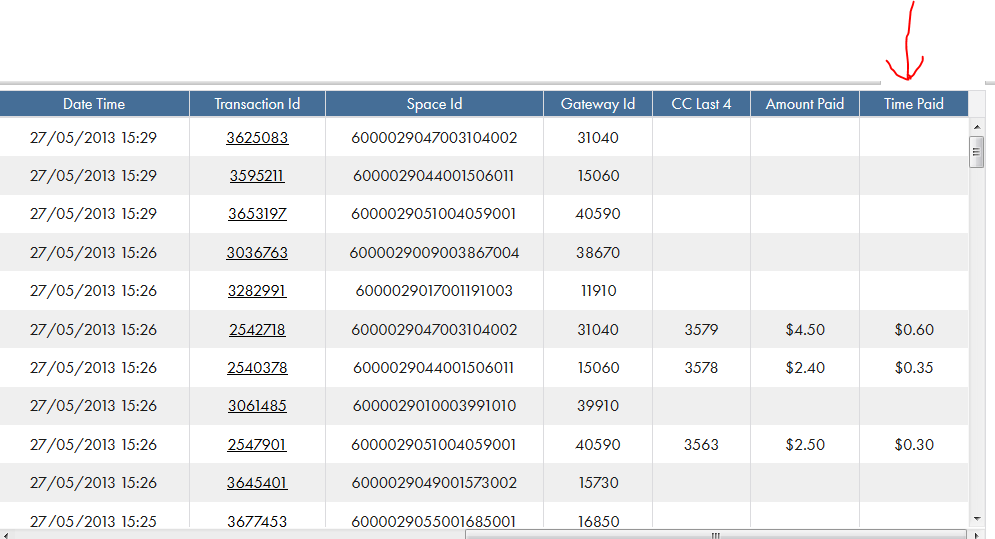
The final step is to verify the data is coming back correctly and is displaying in the grid:

Version 0:



Version 1:





# Adding Grid Filters

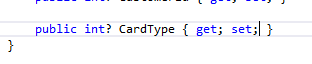
This section will use the Transactions grid as an example and the filter “Card Type” will be added. Since the system has to perform certain functional requirements, the following process must be used to add a filter to a grid. This assumes that the field being added is only a filter and not a column on the grid.

## Steps

These are the steps that will need to be taken in order to maintain data consistency and successfully add a filter to an existing grid.

### Update the Entity

The CustomerTransactionModel is the class that needs to be updated that represents the data on the Transactions Index page. A field called “CardType” will be added. Since this is just another property of the class and NOT a property that represents a grid column, it is important to NOT include the OriginalGridPosition attribute.

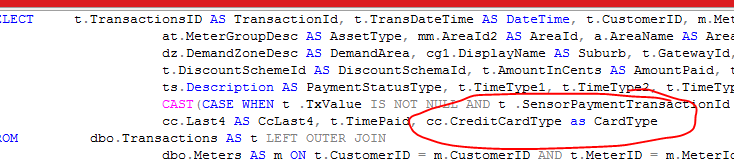


### Update DB View / Stored Procedure

This section only applies if the grid in question uses the stored procedure / view method for retrieving data. Otherwise, the data access being used will have to correctly populate the entity with the valid data; “CardType” in this scenario.

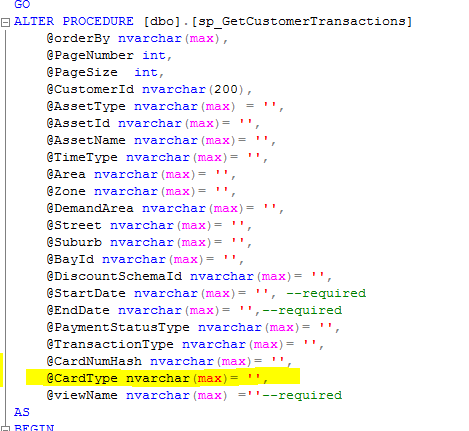
#### Update View

The view needs to be updated to return the data. The name of the field returned “CardType” has to be the same as the property that was added to the entity.

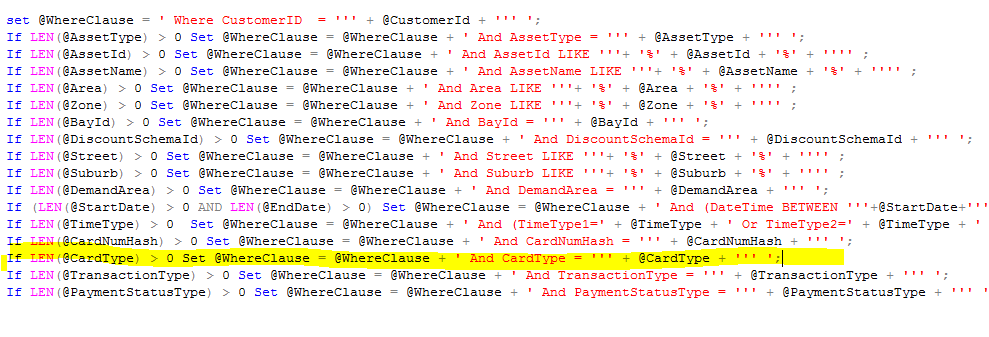


#### Update Stored Proc

The new filter parameter needs to be passed to the stored procedure to allow it to filter against the data brought back by the view. It is important that the filter values being passed in are in the same order as the parameters for the stored procedure. In this example, we will put our CardType filter right after the “CardNumHash” parameter:



The generation of the Where clause needs to take this new field into consideration as well:

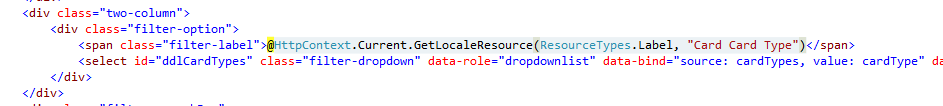


### Update the MVC View

The following steps need to be taken to correctly update the view.

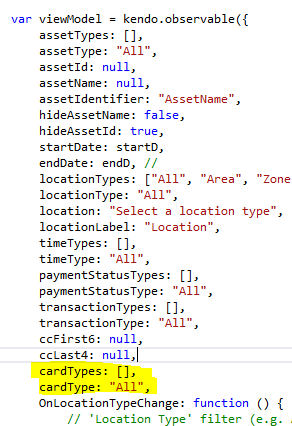
#### Add the filter

On the index page, the filter needs to be added to give the user an option to select it. This will allow the application to pass this filter value to the data access method and filter the result set correctly.

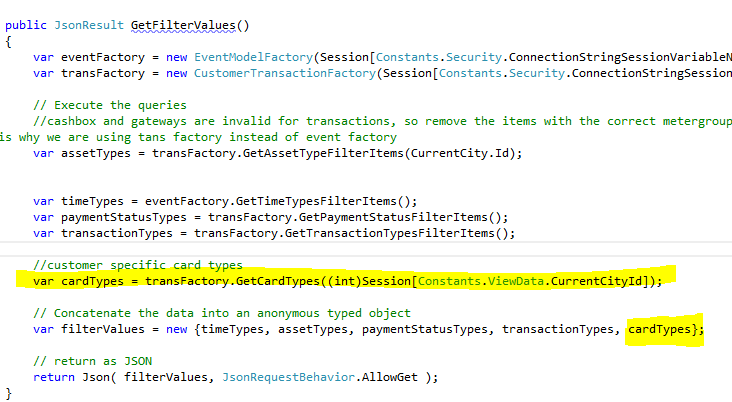


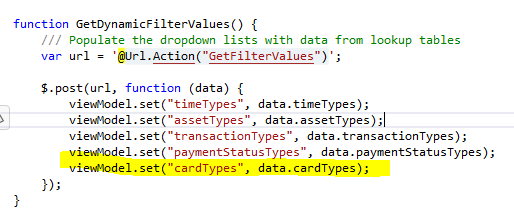
#### Update the View Model

The Kendo viewModel will need to be updated as well:

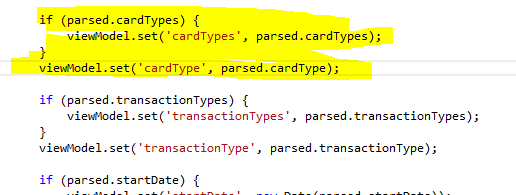


#### Update the data retrieval for the filter (if it is a drop down list)



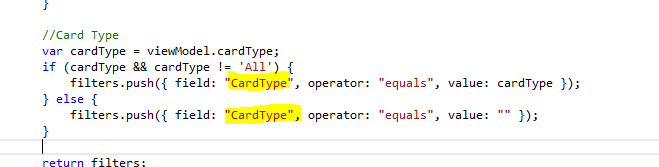


#### Update saved view state on return to the index page:



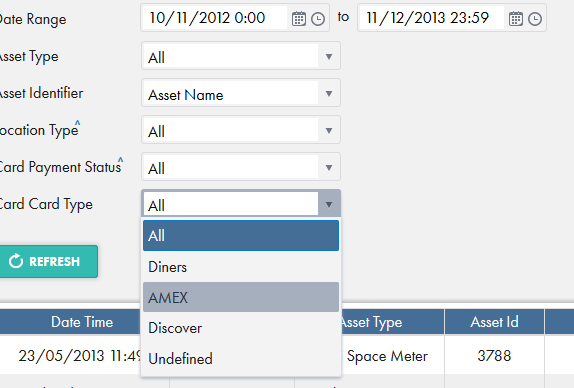
#### Update the LoadGridData to pass the selected filter in correctly

The order is important here, since the order of the parameters passed in must match the order of the stored procedure parameters. Also, the “field” passed in as a filter has to be cased and named the same as the property “CardType”:



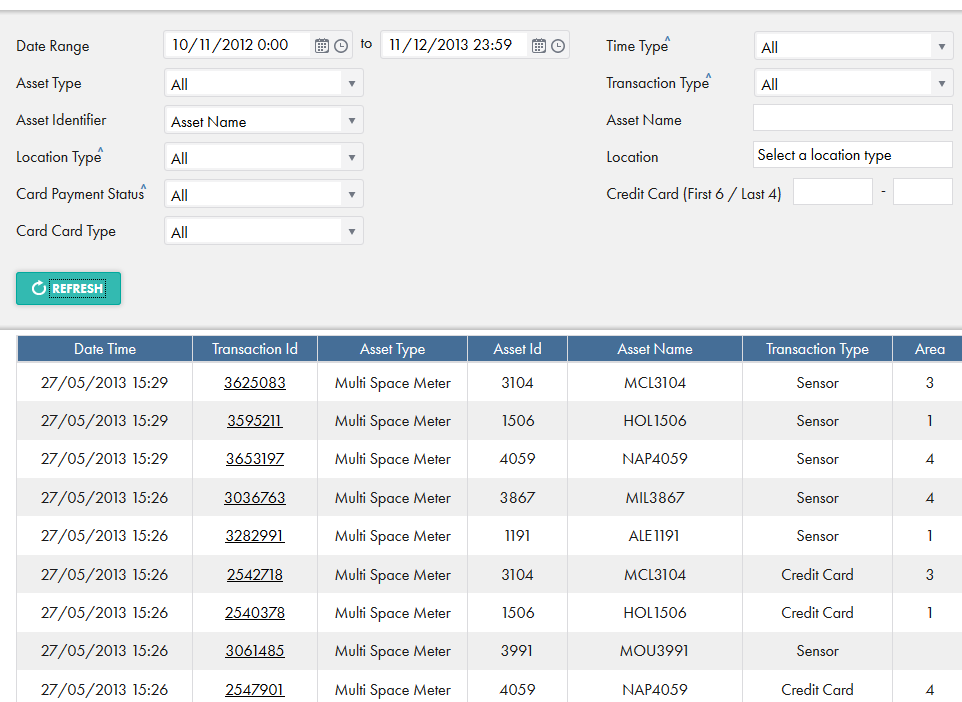
### Test

Make sure the filter is loading correctly:



Verify the data is being filtered correctly:

No Filter:



With Filter:

