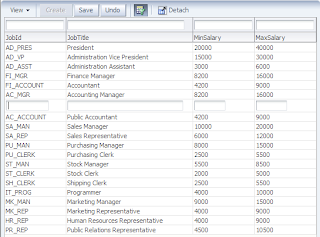
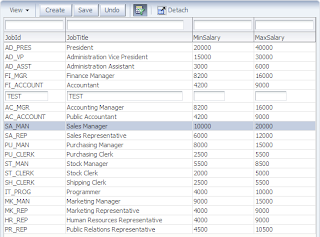
[**Dynamic Row Selection for ADF Table Component**](http://andrejusb.blogspot.com/2010/04/dynamic-row-selection-for-adf-table.html)

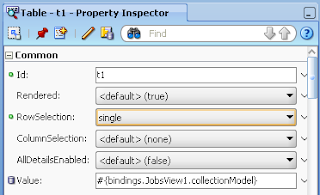
It is a common use case, when user inserts new row into table and we want to prevent him from inserting one more row, before currently inserted row is not stored into database. Some customers like to have such functionality. You can implement this requirement with ADF pretty easily, just need to specify expression language for ***RowSelection*** property. I did a bit more and implemented complete use case, I will describe it today.  
  
Download sample application - [DynamicSelection.zip](http://jdevsamples.googlecode.com/files/DynamicSelection.zip). This sample implements conditional check for Create button - it makes it disabled, when current row is a new row. I'm using ADF table ***clickToEdit*** functionality, this allows to improve table data rendering perfomance:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S7jTxMBDisI/AAAAAAAADfQ/a0tLeOdwfd0/s1600/1.png)

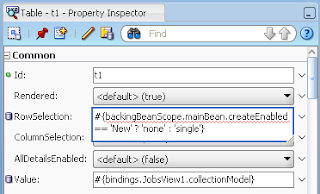
The problem with default implementation - user can select any other record and Create button will become enabled again, even user still didn't saved new record into database:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7jUoqCkEZI/AAAAAAAADfY/129YDz2v3Us/s1600/2.png)

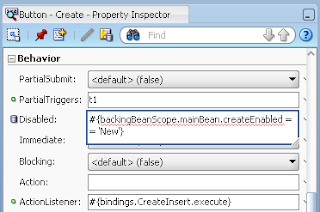
We want to prevent this and make sure Create button remains always disabled, when there is new (not yet saved) record available.  We can achieve this by substituting default ***RowSelection*** property value with expression language:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7jVn9KLq2I/AAAAAAAADfg/tSe6xIiuG3w/s1600/3.png)

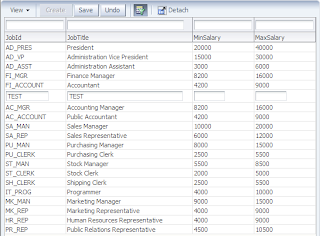
I will change ***single*** value to the following expression language statement:

[](http://4.bp.blogspot.com/_OSq71i5oy0c/S7jWPGUpetI/AAAAAAAADfo/XY6TYqh37Cw/s1600/4.png)

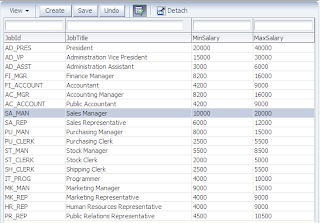
You can see that I'm accessing custom method to get information about current row status (New, Unmodified, etc.) and if current row is not newly inserted row - selection will be enabled, otherwise it is disabled. Its a key thing - it will disable row selection, when there will be new row in the table (before it is saved).  
  
Same custom method is used to disable/enable Create button itself:

[](http://4.bp.blogspot.com/_OSq71i5oy0c/S7jYdZGUcfI/AAAAAAAADfw/UVkyKpZA2Js/s1600/5.png)

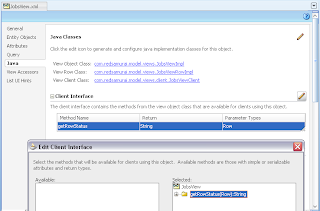
Now, when there is new row - Create button remains disabled always and user cant select any other records, because table record selection is conditionally disabled:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7jY0ps4isI/AAAAAAAADf4/i0BheGJ7juo/s1600/6.png)

When the record is saved - Create button becomes enabled and user can select records again:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7jZUC2-0gI/AAAAAAAADgA/LzkjMNiNLX0/s1600/7.png)

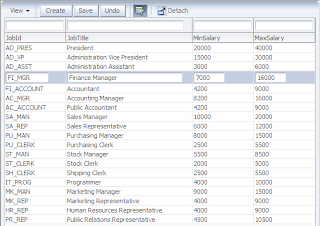
Custom method to check if current row is a new row is coming from View Object implementation class:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7jZ_bipX0I/AAAAAAAADgI/kK5D4gXlW6o/s1600/8.png)

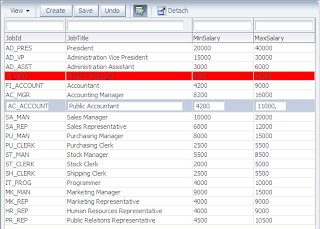
Posted by Andrejus Baranovskis at [8:27 PM](http://andrejusb.blogspot.com/2010/04/dynamic-row-selection-for-adf-table.html)

[**Changed Row Highlighting in Oracle ADF Table**](http://andrejusb.blogspot.com/2010/04/changed-row-highlighting-in-oracle-adf.html)

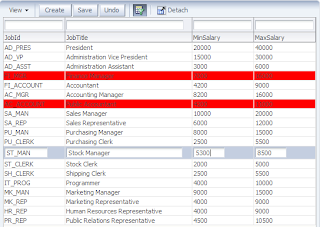
User always wants to be informed about changes he did, this means we should keep him informed - moreover with ADF 11g it can be done very easily. Let's say user is changing one row and moving to another, how he will remember what rows were changed and review them once again before submitting to database. Logical answer - by highlighting changed rows. I will describe this use case and will provide sample application.  
  
Download my sample - [RowHighlight.zip](http://jdevsamples.googlecode.com/files/RowHighlight.zip). You can run this sample on your machine and see that row color will be changed after you will leave edited row and move to the next one.  
  
I'm editing FI\_MGR row:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7uOxgAxkmI/AAAAAAAADgQ/e8LYZCZUlI4/s1600/1.png)

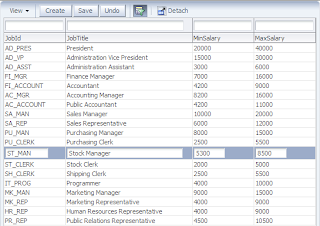
Next I'm changing data for AC\_ACCOUNT row, without committing previously done changes for FI\_MGR:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S7uPpcDwXkI/AAAAAAAADgY/INIDW7EgSjs/s1600/2.png)

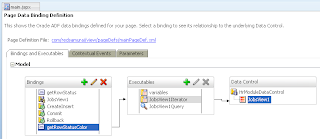
You can see - FI\_MGR row is highlighted, this informs user that he did changes in that row and those changes will be submitted to database when he will press Save button.  
  
I navigate to the next record - AC\_ACCOUNT row becomes highlighted:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S7uQU_TIf8I/AAAAAAAADgg/GPZrqQzZxtM/s1600/3.png)

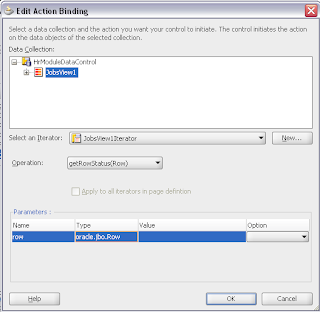
After pressing Save button, all user changes are submitted to database and row highlighting is removed:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S7uQiLJFEVI/AAAAAAAADgo/2E4ADYUWCyo/s1600/4.png)

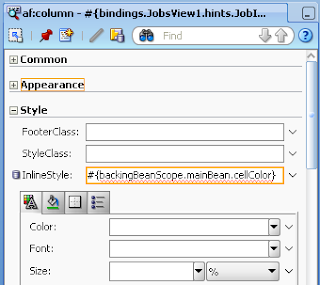
Now I will tell you, how you can implement described use case. Check my sample application, you will see there in Page Definition file, that I have defined **getRowStatusColor** method:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S7uRFSpY55I/AAAAAAAADgw/32u5WVUNaWg/s1600/5.png)

This method is calling custom method available in View Object Implementation class - to check current row status. I will need to iterate through range set of rendered rows, this means I will provide method parameter dynamically:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S7uRf6a5R3I/AAAAAAAADg4/MeLdvMWv6vA/s1600/6.png)

I'm calling this method from Backing Bean class, where I'm evaluating current row status and returning color setting for ***InlineStyle*** property of ADF column component:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S7uR5yyiBqI/AAAAAAAADhA/v4l3qKqFvtI/s1600/7.png)

This means I will highlight each cell individually.  
  
Method in Backing Bean class is accessing rendered row, passing it to **getRowStatusColor** method and returning red color in case if rendered row is ***Modified***:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S7uSZdJDjOI/AAAAAAAADhI/KTRRbmOIc04/s1600/8.png)

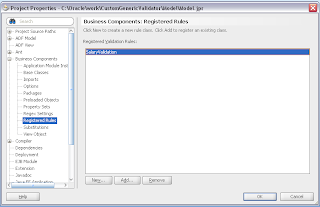
Posted by Andrejus Baranovskis at [9:58 PM](http://andrejusb.blogspot.com/2010/04/changed-row-highlighting-in-oracle-adf.html)

[Email This](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=4723612567385516902&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=4723612567385516902&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=4723612567385516902&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=4723612567385516902&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=4723612567385516902&target=pinterest)

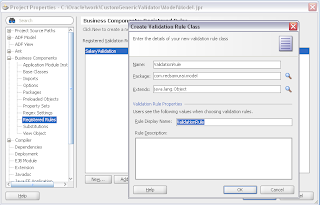
Labels: [ADF](http://andrejusb.blogspot.com/search/label/ADF), [JDeveloper 11g](http://andrejusb.blogspot.com/search/label/JDeveloper%2011g)

[**Implementing Custom Generic PL/SQL Validator in Oracle ADF**](http://andrejusb.blogspot.com/2010/04/implementing-custom-generic-plsql.html)

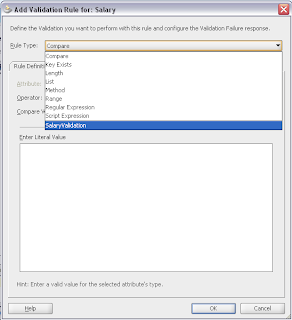
In Oracle ADF you are not constrained only by built in validation rules, its possible to add new ones. This allows to build custom generic validation rules and use them across project through standard JDeveloper wizard. Today I will explain how you can define such rules, and will use example where PL/SQL function will be invoked to check validation logic.  
  
Download sample application - [CustomGenericValidator.zip](http://jdevsamples.googlecode.com/files/CustomGenericValidator.zip). This sample implements custom validation rule, where it invokes PL/SQL validation logic for Number type attribute. In PL/SQL function value is checked to be positive, if not - FALSE is returned and validation rule fails.  
  
To define custom validation rule - open Project Properties for Model project and navigate to Registered Rules section under Business Components group. Here you will be able to define new validation rules for your project:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S8YTK0guAnI/AAAAAAAADhQ/uJMna5zA7mA/s1600/1.png)

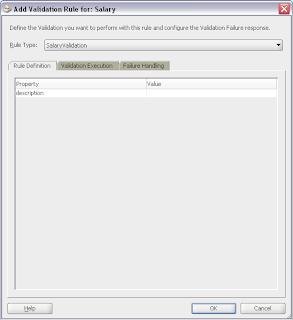
New rule is created in Java class, this class must be specified together with package name and rule display name:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S8YTpm30naI/AAAAAAAADhY/LDNoaXcj6F4/s1600/2.png)

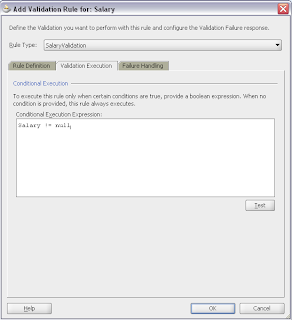
Once new rule is defined in Model project properties, this rule will become available through JDeveloper Validation Rule wizard - custom **SalaryValidation** rule is defined for Salary attribute in Employees entity:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S8YUM8-mlJI/AAAAAAAADhg/inuIokqSeaE/s1600/3.png)

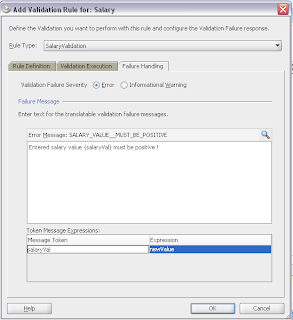
When custom rule is selected, it brings Rule Definition tab - nothing important here:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S8YUf36O2KI/AAAAAAAADho/a1Xi7yMa_Jg/s1600/4.png)

On the next tab - Validation Execution, as with built in validation rule, you can specify conditional execution expression:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S8YXgSutOQI/AAAAAAAADhw/-I6odluowAY/s1600/5.png)

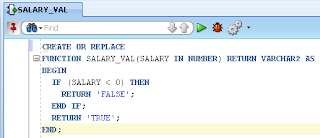
In Failure Handling tab, failure message is defined to inform user about validation failure:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S8YYBkAnv7I/AAAAAAAADh4/tjCeSqDR6CA/s1600/6.png)

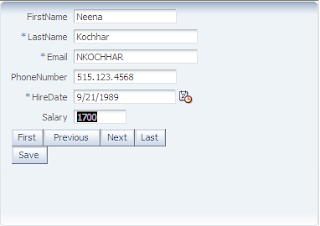
Now let's look into validator code - ***validate()*** method is invoked automatically by the framework. Using input parameter type - JboValidatorContext, I can access current transaction and call PL/SQL function. Based on the result, if it is negative - ValidationException is thrown and user is informed about failure:

[](http://1.bp.blogspot.com/_OSq71i5oy0c/S8YYwBMAyzI/AAAAAAAADiA/gQ0TmOJmUbA/s1600/7.png)

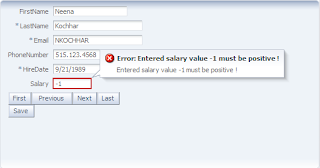
PL/SQL function I'm using in this example is really simple - it accepts NUMBER type parameter and if its value is negative, it returns FALSE:

[](http://2.bp.blogspot.com/_OSq71i5oy0c/S8YZI2juYRI/AAAAAAAADiI/DZ7K6749xvI/s1600/8.png)

On runtime, validation is successful for positive salary value:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S8YZjX7xQTI/AAAAAAAADiQ/4uEhBgw99nc/s1600/9.png)

It brings failure message for negative salary value, based on result from PL/SQL function:

[](http://3.bp.blogspot.com/_OSq71i5oy0c/S8Ybnwp7a0I/AAAAAAAADiY/dmUnE-cyx0c/s1600/10.png)

The same rule can be reused in other EO's, this means PL/SQL will be always invoked from one central place - custom validator class.