

I am working on an article about flexfields and flexfield validation.

Even though the article is not yet finished, I thought the part about 'SPECIAL' and 'PAIR' validation types might be interesting enough. Many people seem to think they can only use the seeded validation sets. However, you can also create your own validation sets. And their options are very powerful. So I wanted to publish this part of the article as a prelude to the full story.

## ***Special Validation***

Special validation is used to provide flexfield functionality for a single value. What that means is that you can have for example a concurrent program parameter that will be filled with a Key flexfield value, or a range of flexfield values.

Let's go back to the Key Flexfield. We know that they are combinations of different segment values that are stored in a separate combination table.

When you want to submit a key-flexfield combination as a parameter to a concurrent program, you can code your own validation for the separate values. But you'll be missing the nice functionality that gives you pop-ups, a validation over the resulting combination and if needed the ID-value for the flexfield combination.

That is possible with a 'Special' validation type.

The special validation uses a number of user exits to enter, validate and query keyflex segments. With special validation, you will be able to enter one or more segment values for a key flexfield. To enter these segment values, 3 user exits can be used. They are: 'POPID', 'VALID' and 'LOADID'.

POPID is used to enable the user to enter the flexfield segment value. It is called when the users cursor enters the segment value field. With this user exit, you decide which segment values should be shown, and how they should be shown.

VALID is called when the user exits the segment value, or confirms the chosen flexfield combination. It validates the entered value against the values existing in the key flexfield table.

LOADID is optional, and it can be used to choose which information will be returned as flexfield value. This can be the concatenated segments, or the id-value for the flexfield combination or segment values.

These 3 user exits can be assigned to 3 'events'. There are more events possible, but they are either not yet in use, or their use is not yet supported. So we will only use 'Validate', 'Edit' and 'Load'.

Sounds complicated, so far? Don't worry; this is not an easy validation. But we'll build some examples to give you an idea. First we start with building a very easy special validation. This will be built on our Code Combination key flexfield. We'll be using a concurrent program 'Test Flex Validation' program to see our different options.

This program is based on the following procedure:

```
CREATE OR REPLACE PROCEDURE XXX_TEST_FLEXFIELD_PARAMS
```

```

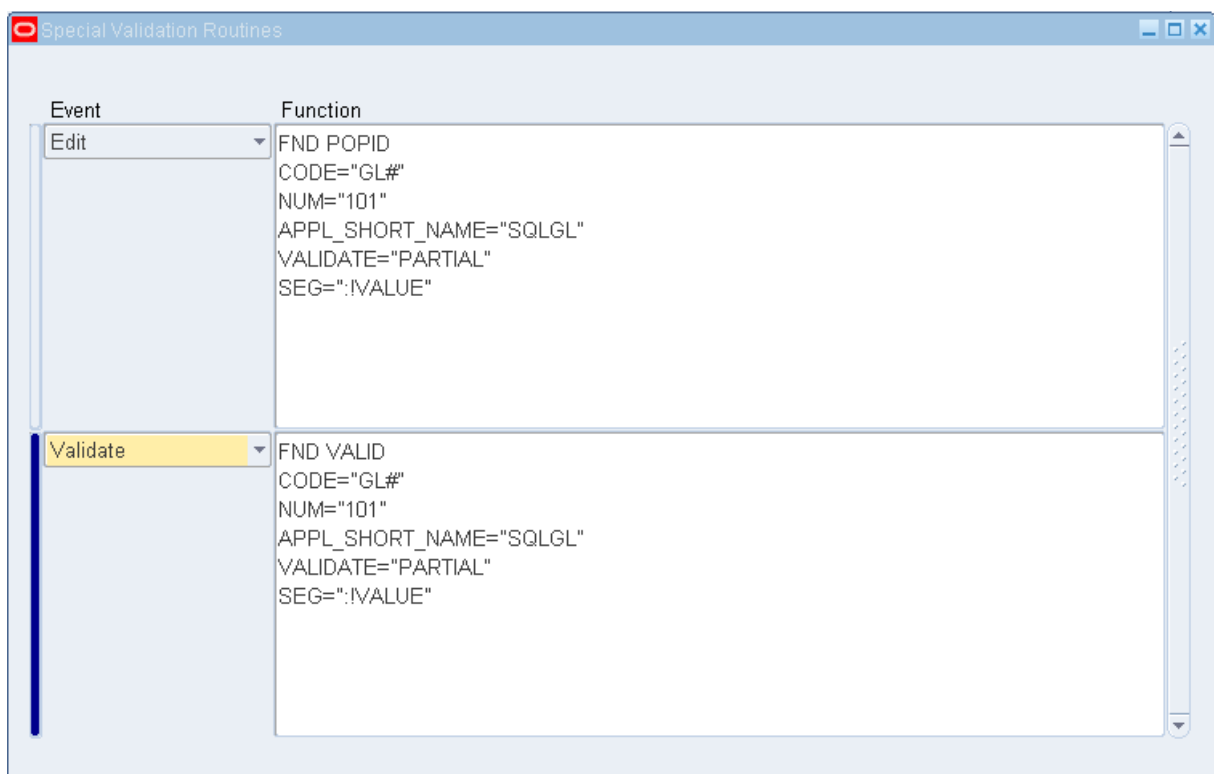
( errbuf out varchar2
, retcode out varchar2
, p_flex in varchar2
, p_flex2 in varchar2 := 'XXX'
, p_flex3 in varchar2 := 'XXX'
, p_flex4 in varchar2 := 'XXX'
, p_flex5 in varchar2 := 'XXX'
) IS
BEGIN
    FND_FILE.PUT_LINE(FND_FILE.OUTPUT,p_flex);
    FND_FILE.PUT_LINE(FND_FILE.OUTPUT,p_flex2);
    FND_FILE.PUT_LINE(FND_FILE.OUTPUT,p_flex3);
    FND_FILE.PUT_LINE(FND_FILE.OUTPUT,p_flex4);
    FND_FILE.PUT_LINE(FND_FILE.OUTPUT,p_flex5);
END;

```

This will only write the parameter value to the output of the request. To use flexfields as parameters for concurrent programs, we need to define a value set based on them.

We will start with the barest setup to enter a key-flexfield combination. For this article, we use the accounting flexfield, with code 'GL#' and id-num '101'.

In this case, we have the following definition:



So what does this mean?

The first box is for the edit event. This will be triggered when the user enters the cursor into the field with this value set.

**FND POPID** This is the user exit to pop up a flexfield screen, and let the user enter the flexfield values.

CODE="GL#" This is the flexfield code for the key flexfield that we will be using.

APPL\_SHORT\_NAME="SQLGL" The short name for the application the flexfield belongs too. Together with 'Code', this will identify the flexfield itself.

NUM="101" The id-number for the flexfield structure. If you have only a single structure flexfield, it is optional. For flexfields enabled for multiple structures, you need to enter the id-number.

VALIDATE="PARTIAL" Validate can be 'None', 'Partial' or 'Full'. None means the combination is not validated. Partial means that the separate segments are validated, there is no validation if the combination exists. Full means that segments and combination will be checked, and if a new value is entered, this will be inserted into the key flexfield table.

SEG=":!VALUE" This is the forms field that will be used to store the value of the segments.

The second box is for the 'Validation' event. This code will be called when the user navigates out of the field, or submits the entire combination.

Now when we set this value set as a parameter for our concurrent program, we can see how the validation works:

The screenshot shows the 'Concurrent Program Parameters' window. The 'Program' is 'Test Flexfield validation' and the 'Application' is 'Customizations'. The 'Conflicts Domain' and 'Security Group' are empty. A table lists parameters, with the first one 'Code combination' selected and its 'Enabled' checkbox checked. Below the table, the 'Validation' section is expanded, showing 'Value Set' as 'XXX\_SPECIAL\_VAL', 'Description' as 'Test Special Flexfield Validat', and 'Default Type' as empty. There are checkboxes for 'Required' and 'Enable Security', both unchecked. The 'Range' is a dropdown menu. The 'Display' section is also expanded, showing 'Display Size' as 50, 'Concatenated Description Size' as 25, 'Description Size' as 50, and 'Prompt' as 'Code Combination'. A 'Token' field is empty at the bottom.

Seq	Parameter	Description	Enabled
10	Code combination		<input checked="" type="checkbox"/>
			<input type="checkbox"/>
			<input type="checkbox"/>
			<input type="checkbox"/>

Validation

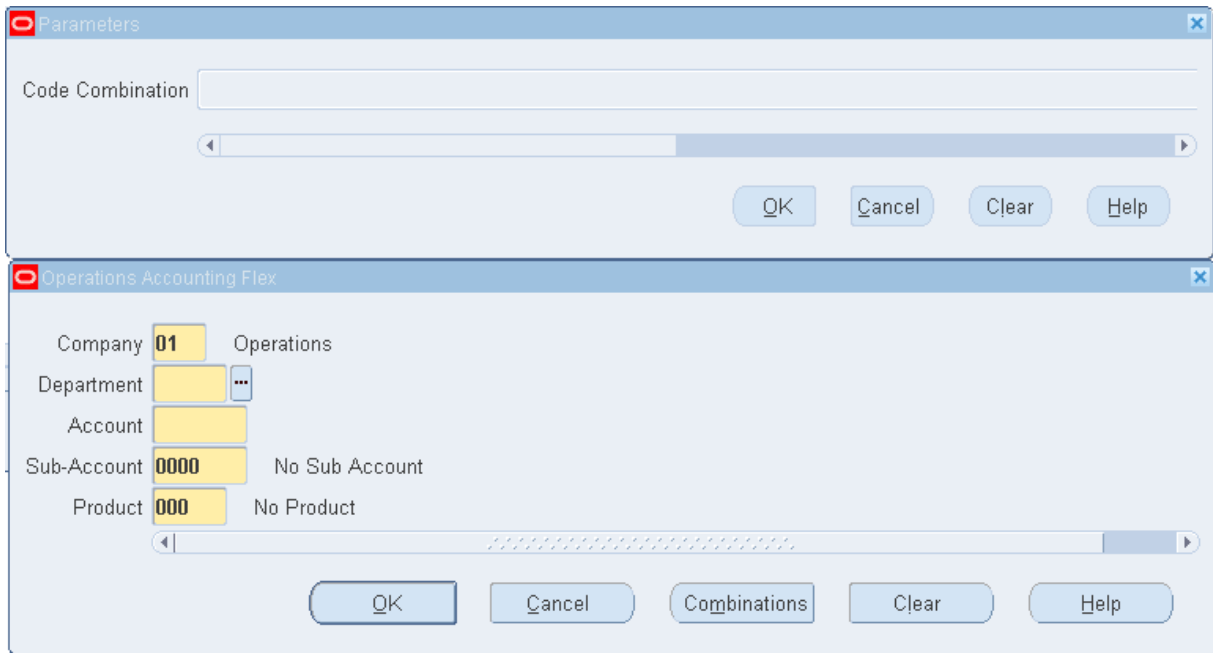
Value Set: XXX\_SPECIAL\_VAL  
Description: Test Special Flexfield Validat  
Default Type:   
 Required  Enable Security  
Range:

Display

Display Size: 50  
Concatenated Description Size: 25  
Description Size: 50  
Prompt: Code Combination

Token:

Now when we run the program, we get this pop-up:



We have all the functionality of the key flexfield. We can use the 'Combinations' button to search for existing combinations, and all separate segments will be validated, as will be the final combination.

When we submit a value to our program, it will show the concatenated segments as the value of our parameter:

Request ID	Name	Parent	Phase	Status	Parameters
5804580	Test Flexfield validation		Completed	Normal	01-000-1220-0000-000

Now let's see some more features of this validation. For example, we'd like to have the value of the combination id. (CODE\_COMBINATION\_ID in our case, since we use the Accounting Flexfield).

To get that, we need to add the LOADID user exit:

Event	Function
Edit	FND POPID CODE="GL#" NUM="101" APPL_SHORT_NAME="SQLGL" VALIDATE="FULL" SEG=":IVALUE" ID=":IID" DINSERT="N"
Load	FND LOADID CODE="GL#" NUM="101" APPL_SHORT_NAME="SQLGL" VALIDATE="FULL" SEG=":IVALUE" ID=":IID"
Validate	FND VALID CODE="GL#" NUM="101" APPL_SHORT_NAME="SQLGL" VALIDATE="FULL" SEG=":IVALUE" DINSERT="N" ID=":IID"

The 'Load' event will get the combination-id from the flexfield table. This is only possible for the 'VALIDATE="FULL", since it will validate the whole combination. Also we need to set the ID=":IID". This will populate the :IID column with the ID value of the combination.

Finally, I added the 'DINSERT="NO" ', because we don't want to allow insertion of new code combinations from this value set. (And Validation="FULL" by default inserts new combinations into the flexfield column).

Now when we run the concurrent request, we see that the parameter value is the code\_combination\_id instead of the concatenated segments:

Request ID	Name	Parent	Phase	Status	Parameters
5804582	Test Flexfield validation		Completed	Normal	13393

With these user exits it is also possible to select just a number of segments, instead of the whole combination. For this we remove the 'Load' / 'LOADID' part again.

Then we add a 'DISPLAY="x" ' to the 'Edit' and 'Validate' user exits. The "display" parameter is defaulting to 'ALL'. But you can also specify separate segments by their sequence number or names. In our case, we display the first 2 segments:

Event	Function
Edit	FND POPID CODE="GL#" NUM="101" APPL_SHORT_NAME="SQLGL" VALIDATE="PARTIAL" SEG=":IVALUE" DESC=:IMEANING DINSERT=N DISPLAY="1" DISPLAY="2"
Validate	FND VALID CODE="GL#" NUM="101" APPL_SHORT_NAME="SQLGL" VALIDATE="PARTIAL" SEG=":IVALUE" DINSERT="N" DISPLAY="1" DISPLAY="2"

Now when we run the concurrent program, we get a pop-up for only the first 2 values:

Special Validation Key

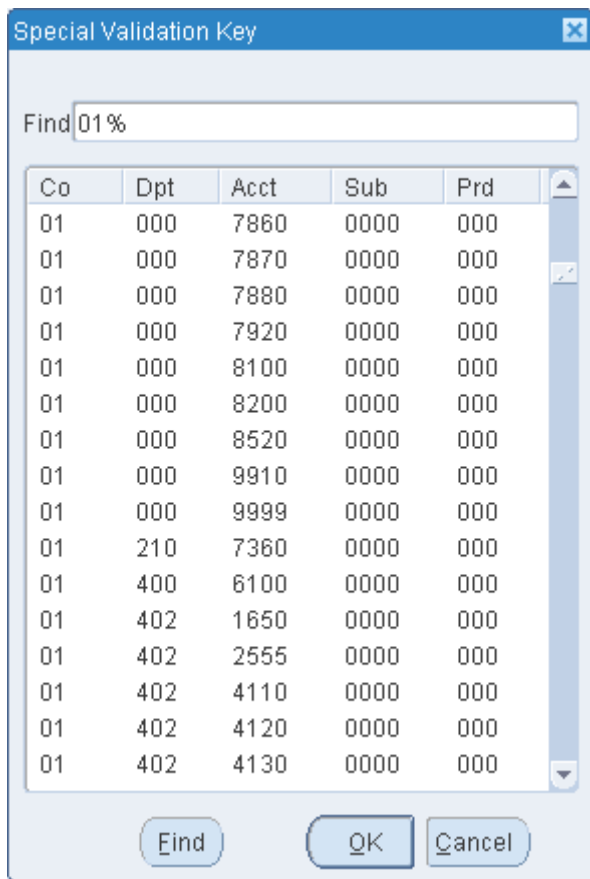
Company  Operations

Department

A very nice feature (at least as far as I'm concerned) is the use of a where clause on the combination values. Consider the following 'Enter' code:

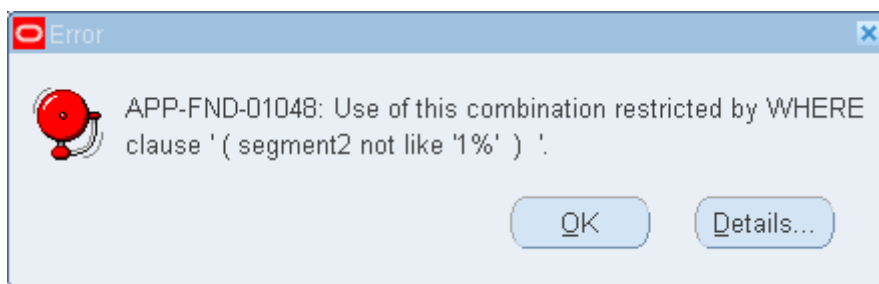
```
FND POPID
CODE="GL#"
NUM="101"
APPL_SHORT_NAME="SQLGL"
VALIDATE="FULL"
TITLE="Special Validation Key"
ID=":IID"
SEG=":IVALUE"
DESC=":IMEANING"
WHERE="segment2 not like '1%' "
```

The "WHERE" clause prevents us from choosing combinations that have a segment2 starting with '1'. When we run our concurrent program with this, and choose the combinations:



There is no Dpt starting with 1.

When we add the “WHERE”-clause to the validation event too, it will prevent us from entering the values manually:



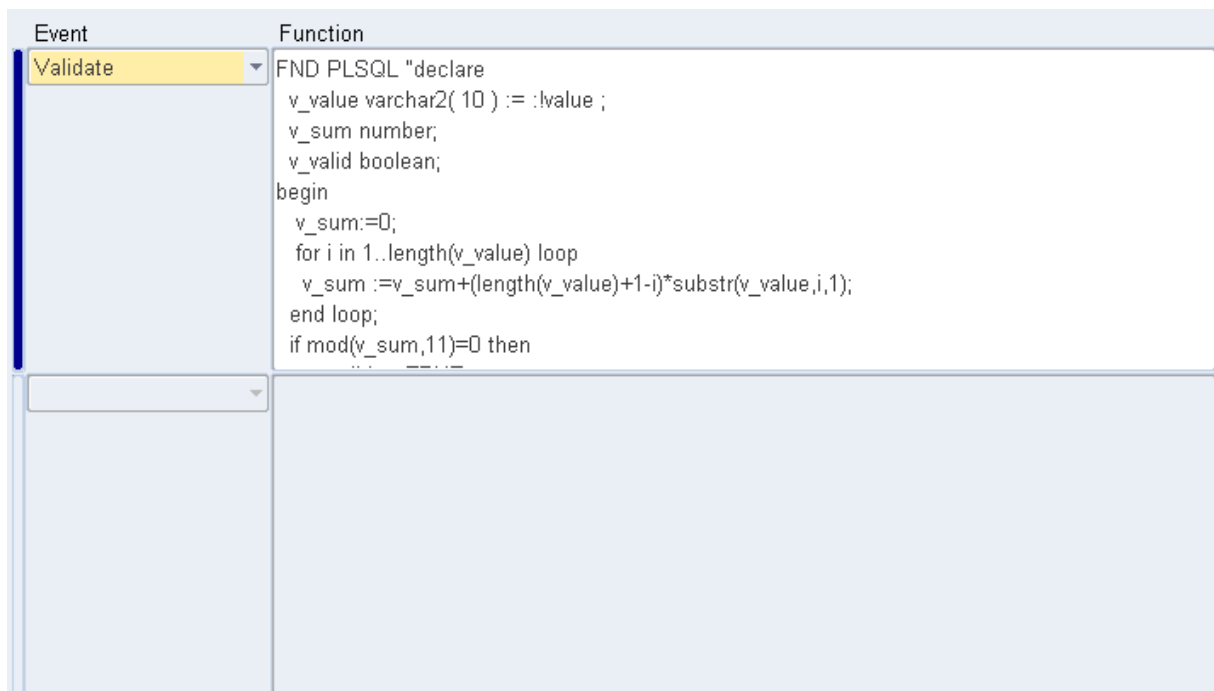
The last feature that we'll look into is the use of a pl/sql validation through the special validation routines. By using the user-exit PLSQL, we can call an anonymous PL/SQL block in our 'Validation' event. I created a value set with the following function for the 'Validation' event:

```
FND PLSQL " declare
  v_value varchar2( 10 ) := :!value ;
  v_sum number;
  v_valid boolean;
begin
  v_sum:=0;
  for i in 1..length(v_value) loop
    v_sum :=v_sum+(length(v_value)+1-i)*substr(v_value,i,1);
```

```

end loop;
if mod(v_sum,11)=0 then
    v_valid := TRUE;
else
    v_valid:=FALSE;
end if;
if not v_valid then
    fnd_message.set_name('FND','FND_GENERIC_MESSAGE' );
    fnd_message.set_token('MESSAGE','This is not a valid bank account');
    fnd_message.raise_error;
end if;
END; "

```



This PL/SQL procedure validates a (Dutch) bank account number. If it does need pass the test, a message will be displayed. This gives you almost unlimited possibilities for validating entered data. As you can see, it is only a 'Validate' event. Because we don't need any special functionality for entering the data. We can limit the entry to numbers only on the 'Validation Set' main page.

Now when we use this value set for our concurrent program, we can only enter valid dutch bank accounts:



And

Request ID	Name	Parent	Phase	Status	Parameters
5804594	Test Flexfield validation		Completed	Normal	326599509

The list of parameters for the user exits is longer than this. So we won't be going through all the possibilities. You can check the Developers Guide and the Flexfield guide for a complete listing of options. (Did you notice the flexfield title that I sneaked into the pop-up? Try and find the option for that!)

Please try the different options for yourself, and realize the possibilities of the special validation.

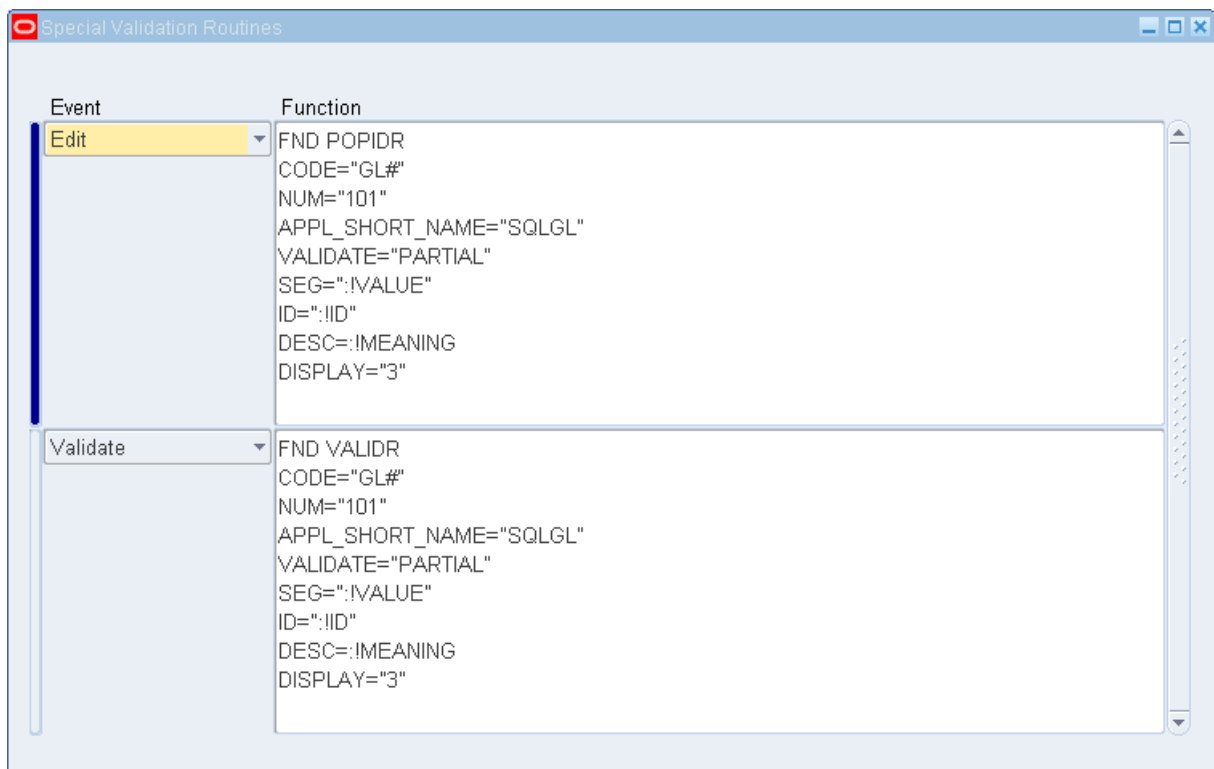
## Pair Validation

Meanwhile, we'll continue to the 'Pair' validation. The pair validation is very much like the 'special' validation. It uses the same kind of user exits, but this time, a range of segment values or combinations is selected.

Let's first create a range of the account segment. Instead of using POPID and VALID, we use POPIDR and VALIDR. The R-version of the user-exits automatically create a range.

Of course we need 2 parameters to set the range. However, we need only one validation set.

I created the validation set 'XXX\_PAIR\_VAL'. I entered only the edit and validate events:



The next step is to set the parameters for both the low and high value. Both parameters have the validation set 'XXX\_PAIR\_VAL'.

Seq	Parameter	Description	Enabled
10	Low value		<input checked="" type="checkbox"/>
20	High value		<input checked="" type="checkbox"/>
			<input type="checkbox"/>
			<input type="checkbox"/>

**Validation**

Value Set:  Description:

Default Type:  Default Value:

Required  Enable Security Range:

**Display**

Display Size:  Description Size:

Concatenated Description Size:  Prompt:

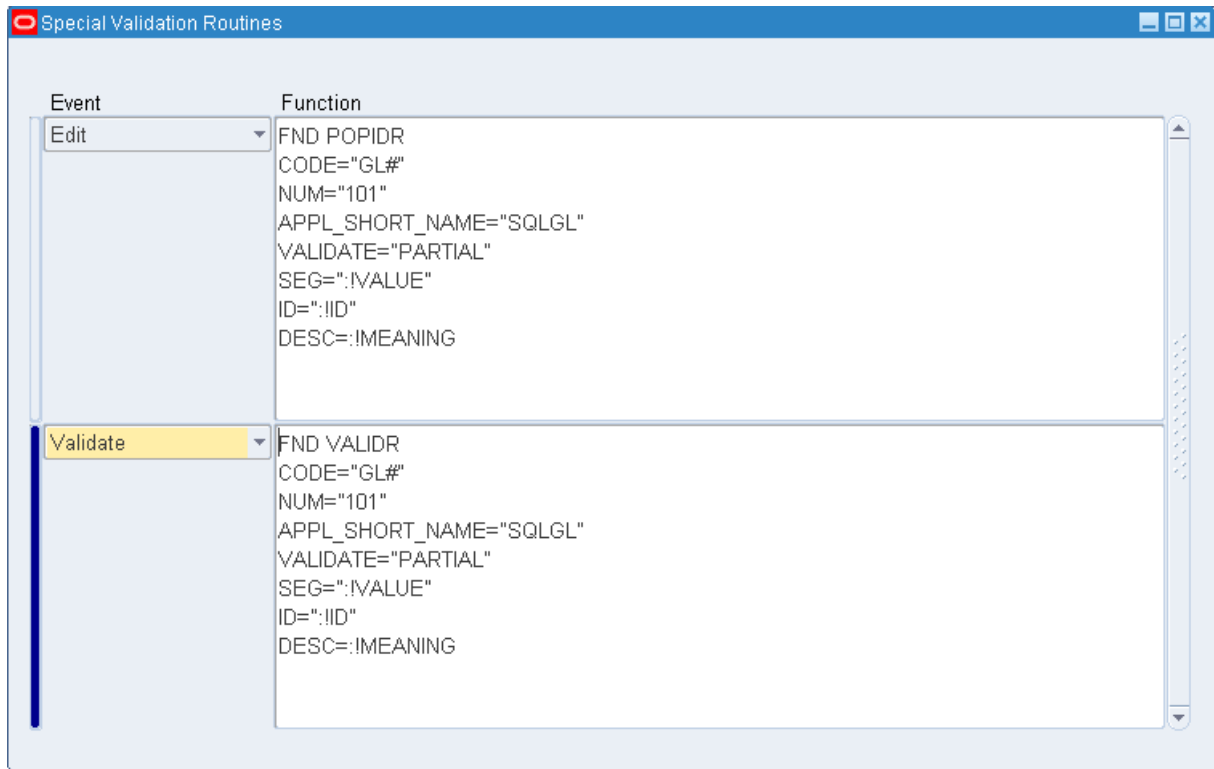
Now when we run the program, we can enter a range. This includes validation that the high value is indeed higher or equal to the low value.

Of course the concurrent program will receive the values for 2 parameters.

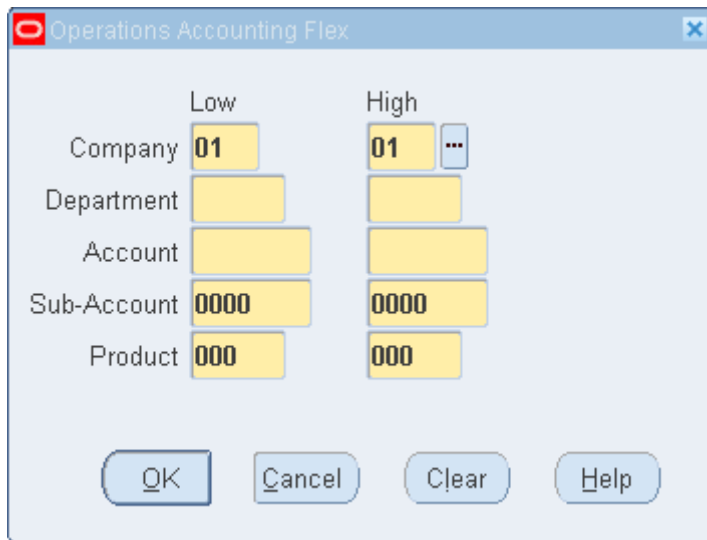
When we use the full validation we can enter a range of the whole account combination. Note that we cannot use the FULL validation for pair-validation. Because that would mean the use of the combination-id from the flexfield table and based on the combination-id's you cannot build a range.

So we can only use PARTIAL and NONE for the validation. For that same reason, I have not yet had a reason to use a LOAD event for PAIR validation. It is however allowed to use one.

I created a PAIR validation for the whole accounting range as follows:



When used in the concurrent program, it will indeed allow us to enter a range of all segments:



That completes the chapter on PAIR validation too.