

Secrets of the Account Generator

Karen Brownfield
Venkat Kancherla
Solution Beacon

Introduction

Account Generators are workflows that provide the E-Business suite modules the ability to automatically construct Accounting Flexfield combinations using Oracle defined or your own business rules. Although automatic construction of account combinations improves the accuracy and ease of data entry, when new business situations arise that are not modeled by the Oracle seeded account generators, companies will need to modify these workflows.

This paper will explain the procedures that Oracle provides to build account generators, how Oracle initiates the seeded account generators, and the rules that companies must follow to customize the account generators. As part of this explanation, the paper will discuss the attributes that are present for all account generators, even though you will never see these attributes in the Workflow Builder, and how to use these attributes to send a notification when the account generator fails. The paper presupposes that the reader is familiar with workflow terminology and how to use the Builder tool.

Seeded Account Generators

11/10 provides the ability to register custom processes for the following account generators:

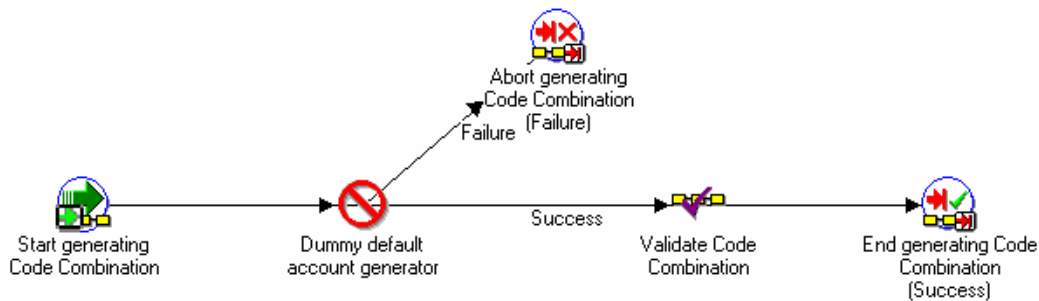
- * OM: Generate Cost of Goods Sold Account (OECOGS) – generates the cost of goods sold account when invoices imported into AR
- * PSB Account Generator for OLD Integration (PSBLDMAG) – Public Sector Budgeting – used to derive accounts for positions with POETA charging instructions that are then used to import salary distribution information from LD
- * ITR Account Generator (ITRWKFAG) – Account Generator for self service ITR. This workflow will build creation and receiving accounts for ITR service lines.
- * IAC account Generator (IGIACWF) – used in Public Sector Assets
- * MHCA Account Generator (IGIAMAWF) – used in Public Sector Assets
- * IGC Charge Account Generator (IGCACGNC) – Public Sector Contracts, this workflow is used to generate charge account for contract commitment.
- * IGC Budget Account Generator (IGCACGNB) – Public Sector Contracts, this workflow is used to generate charge account for contract commitment.

- * Project Budget Account Generation (PABDACWF) – used by Projects to generate accounting combinations for all budget items in an integrated project budget.
- * AR: Substitute Balancing Segment (ARSBALSG) – updates the balancing segment during various accounting activities against transactions and receipts
- * FA Account Generator (FAFLEXWF) – used by Assets to generate the accounting combinations for each asset transaction
- * Project Supplier Invoice Account Generation (PAAPINVW) – used by Payables to derive the invoice distribution accounting combination if the distribution is Project related – MUST be customized
- * Generate Cost of Goods Sold Account (SHPFLXWF) – Pre-11i Cost of Goods Sold Account Generator – see MetaLink note 260697.1
- * Project Web Employees Account Generator (PAAPWEBX) – used by iExpenses to derive accounting combinations for expense report lines that reference a project – MUST be customized
- * PO Account Generator (POWFPOAG) – used by Purchasing to derive the charge, budget, variance, and accrual accounting distributions for each PO line – MUST be customized if using Oracle Projects
- * PO Requisition Account Generator (POWFRQAG) – used by Purchasing to derive the charge, budget, variance, and accrual accounting distributions for each requisition line – MUST be customized if using Oracle Projects
- * Inventory Cost of Goods Sold Account (INVFLXWF) – called while processing Intercompany Transactions

Customization Rules

To customize an account generator, you must have the Workflow Builder. You may also need a PL/SQL developer tool. You will either need the apps user password, or your DBA can extract the .wft file that contains the item type(s) that you wish to customize. If someone is creating the .wft file for you, make sure it contains the account generator(s) requested, the Standard item type, and the Standard Flexfield Workflow item type. (Standard and Standard Flexfield Workflow should load automatically when loading any account generator into the Builder.)

The following is the seeded 'Generate Default Account' process for Project Supplier Invoice Account Generation. It contains the minimum required nodes for any account generator, plus the node 'Dummy default account generator' which must be replaced by your company's custom procedure(s).



Notice that instead of the standard Start or Receive Event activity, account generators begin with the node 'Start generating Code Combination'. This is required. Additionally, instead of End nodes with result types, account generators must contain at least one 'Abort generating Code Combination' (must be marked as an End node on the Node tab with a result of 'Failure') and at least one 'End generating Code Combination' (must be marked as an End node on the Node tab with a result of 'Success'). Processes that will be the top process must specify a result_type of 'Flexfield Result'.

Prior to navigating to 'End generating Code Combination' you must include the node 'Validate Code Combination'. Note that this node does not have a result_type associated with it. So you cannot branch from this node to different nodes based on whether or not the code combination is valid. What this node does and how to get around the inability to branch will be explained later.

Since account generators are designed to be called from a form or batch process (hereafter referred to as "the calling process") and return the combination to be used in the form or batch process, there are restrictions on the type of activities that can be included in the account generator.

- * No notifications
- * No deferring activities to the background engine (see comment on cost later)
- * No Parallel flows
- * No 'Any' transition
- * No Master/Detail
- * 'On Revisit' behaves as if set to Loop
- * The following standard activities are not allowed (because they defer processing):
 - > And
 - > Block
 - > Defer Thread
 - > Wait
 - > Continue Flow / Wait for Flow

- > Role Resolution
- > Voting
- > Compare Execution Time
- > Notify
- * The use of WF_ENGINE API's is restricted to:
 - > CreateProcess
 - > StartProcess
 - > SetItemAttribute
 - > GetItemAttribute
 - > GetActivityAttribute
 - > CompleteActivity
 - > AddItemAttribute
 - > LaunchProcess

How Account Generators Work

The calling process initiates the account generator by first calling the function FND_FLEX_WORKFLOW.INITIALIZE. The parameters passed to this routine are:

- * Application Short Name – for the accounting flexfield, this will be 'SQLGL'. This value is used to find application_id from FND_APPLICATION (= 101)
- * code – 'GL#' for the Accounting flexfield, 'GLAT' for the Reporting Attributes Accounting flexfield
- * Num – id_flex_num – along with the application_id and id_flex_code, these fields form the primary key to FND_ID_FLEX_STRUCTURES, which stores the definitions of your accounting flexfield
- * ItemType – the name of the workflow that is your account generator

The function returns a VARCHAR variable which is the itemkey. This function performs the following tasks:

- * Fetches the number of segments in your chart of accounts.
- * Determines, based on the profile option 'Account Generator:Run in Debug Mode', whether to set the item key to #SYNC or to FND_FLEX_WORKFLOW_ITEMKEY_S.nextval .
- * Calls the api 'Wf_Engine.CreateProcess'.
- * Creates the following attributes
 - > FND_FLEX_APPSNAME – Flexfield Application Short Name, i.e. 'SQLGL'
 - > FND_FLEX_CODE – 'GL#' or 'GLAT'
 - > FND_FLEX_NUM – id_flex_num
 - > FND_FLEX_APPLID – 101
 - > FND_FLEX_NSEGMENTS – the number of enabled segments in your flexfield
 - > FND_FLEX_CCID – will be the code combination id built by the account generator

- > FND_FLEX_SEGMENTS – will be the concatenated segments
 - > FND_FLEX_DATA – will be the concatenated ID's
 - > FND_FLEX_DESCRIPTIONS – will be the concatenated descriptions of each segment
 - > FND_FLEX_MESSAGE – if the generator fails, will be the error message
 - > FND_FLEX_STATUS – will be the validation status
 - > FND_FLEX_INSERT – whether new combinations can be inserted
 - > FND_FLEX_NEW – will be whether this is a new code combination
 - > FND_FLEX_SEGMENTn – one attribute for each segment of the accounting combination – will store the value of each segment
- * Sets the values for FND_FLEX_APPSNAME, FND_FLEX_CODE, FND_FLEX_NUM, FND_FLEX_APPLID, and FND_FLEX_NSEGMENTS.

The attributes created are only added to the table WF_ITEM_ATTRIBUTE_VALUES.

This is why even when Account Generator: Run in Debug is set to 'Yes', these attributes can't be seen in the workflow monitor.

The calling program then calls one of two versions of FND_FLEX_WORKFLOW.GENERATE. The first version is used when called from a form and passes back (via IN OUT variables) the code_combination_id, the concatenated segments, the concatenated descriptions, and concatenated ids, an error message and (via a RETURN BOOLEAN) True/False as to the success of the generation. This routine calls the second generate and passes the RETURN BOOLEAN from the second generate as its own RETURN BOOLEAN. Since this version is designed to be called from a form, the assumption is made that the form will check whether the generation was successful, and if so, places the combination in the form field and calls the appropriate routines to validate whether the combination exists or can be added and whether the combination is valid.

The second one can be called directly from batch programs. This one passes in whether insertion of new accounts is allowed, and returns (via IN OUT variables) the code_combination_id, the concatenated segments, the concatenated descriptions, and concatenated ids, an error message, whether the returned combination is a new combination (Boolean variable) and (via a RETURN BOOLEAN) True/False as to the success of the generation. The calling program controls whether new combinations are allowed. If called with new combination allowed=TRUE, and the function passes back new combination generated = TRUE, then the calling process must handle the insertion of the new combination and issue a commit. Failure to issue the commit places a lock on the code combination table.

The 2nd generate function sets the engine threshold to 999999 so that none of the functions will ever be run in the background, then issues a call to Wf_Engine.StartProcess. Due to the value of the threshold, the generate function will not continue until the account generator finishes. If the generation is successful, the generate function returns TRUE as the result, and, via IN OUT variables, the ccid, the concatenated segments, concatenated id's, concatenated descriptions, error message and whether or not the code combination is new.

If the generation is not successful, as much information as can be generated is returned along with the error message.

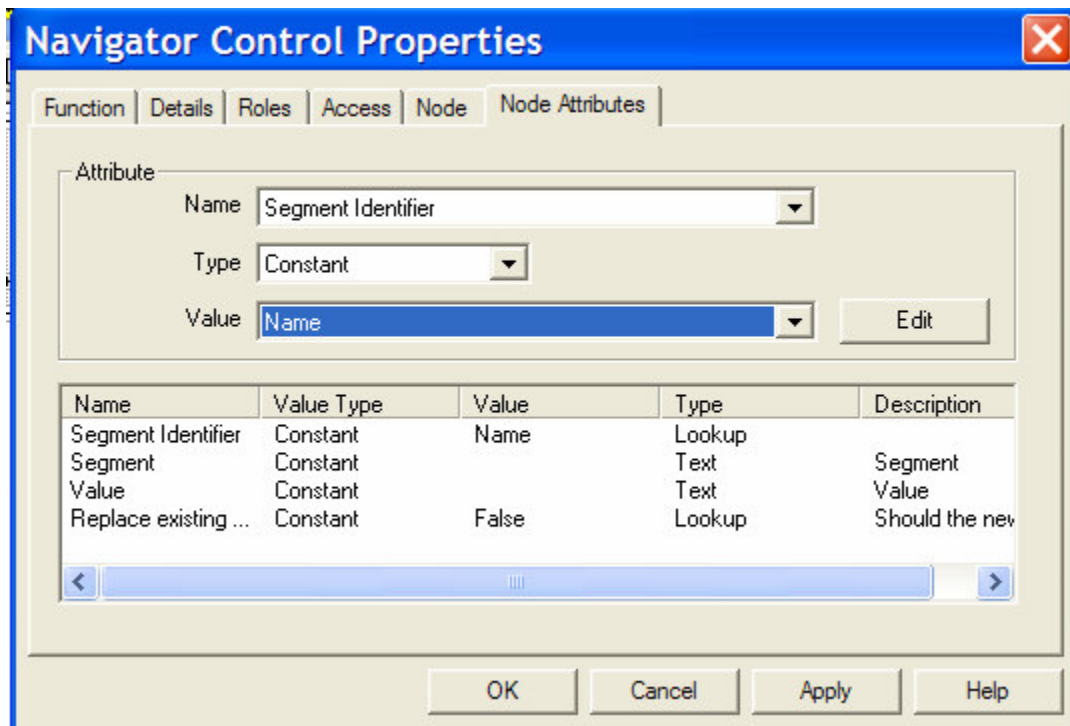
Standard Flexfield Activities

When customizing an account generator, the developer can use any of the functions contained in Standard Flexfield workflow, any of the functions contained in Standard workflow except those in the prohibited list, and/or develop custom functions (as long as the custom function obeys the rules described above). Any function in the Standard Flexfield workflow itemtype that is copied into the process diagram must have values set for its activity attributes. This is done by double-clicking the node in the diagrammer window (which opens the properties page) and then setting the values in the Node Attributes tab.

There are several activity attributes that are common to many of the functions. These attributes are:

- * Segment Identifier – specify 'Qualifier' if you can identify the segment by type (Balancing Segment, Cost Center, Natural Account, Intercompany, Secondary Tracking) or specify 'Name' if you can't
- * Segment – enter the qualifier or the name of the segment. If you use name, you must match exactly the value in the Name field of the Key Flexfield Segments form.
- * Value – the value to be assigned to the segment
- * Structure Number – the accounting flexfield structure number, if you are building a combination for a different set-of-books.
- * Replace existing value – use 'True' to override and replace any value that already is assigned to the segment, use 'False' to copy a value into a segment only if that segment is null.

The following screen shows the Node Attributes page for 'Assign Value to Segment'. It uses the four of the five attributes above. While for each attribute you can choose to assign a constant value or an item attribute, the use of item attribute is rarely used for any of the above attributes except Value (and maybe Structure Number).



'Assign Value to Segment' assigns a value to a specific segment. It uses the activity attributes 'Segment Identifier', 'Segment', 'Value', and 'Replace existing value'. Whatever value is contained in the attribute 'Value' is then stored in the item attribute FND_FLEX_SEGMENTn (if 'Replace existing value' is TRUE or 'Replace existing value' is FALSE and FND_FLEX_SEGMENTn is null).

'Copy Segment Value from Code Combination' copies a specified segment value from one combination into the combination being built. It uses the activity attributes 'Code Combination ID', 'Segment Qualifier', 'Segment', and 'Replace existing value'. 'Code Combination ID' is usually set to an item attribute which then must be populated prior to executing this function activity. The function finds the code combination id, extracts the segment specified and stores the result in the item attribute FND_FLEX_SEGMENTn (if 'Replace existing value' is TRUE or 'Replace existing value' is FALSE and FND_FLEX_SEGMENTn is null).

'Copy Segment Value from Other Structure Code Combination' copies a specified segment value from a combination in an accounting flexfield other than the set-of-books accounting flexfield into the combination being built. It uses the activity attributes 'Structure Number', 'Code Combination ID', 'Segment Identifier', 'Segment', and 'Replace existing value'. 'Code Combination ID' is usually set to an item attribute which then must be populated prior to executing this function activity. The function finds the code combination id, extracts the segment specified and stores the result in the item attribute FND_FLEX_SEGMENTn (if 'Replace existing value' is TRUE or 'Replace existing value' is FALSE and FND_FLEX_SEGMENTn is null).

Because the above routine is working with two different accounting flexfields, some additional rules must be followed. If Segment Identifier is set to Name, both flexfields must contain a segment with the name specified in Segment. If Segment Identifier is set to Qualifier, both flexfields must contain a segment assigned to that qualifier. The rules for the specified segment must be the same (length of field, numerics allowed, etc).

'Copy Values from Code Combination' copies all values from a specified code combination id into the combination being built. It uses the activity attributes 'Code Combination ID' and 'Replace existing value'. 'Code Combination ID' is usually set to an item attribute which then must be populated prior to executing this function activity. The function finds the code combination id, extracts all the segments and stores each segment in the item attributes FND_FLEX_SEGMENTn (if 'Replace existing value' is TRUE or 'Replace existing value' is FALSE and FND_FLEX_SEGMENTn is null). 'Replace existing value' is evaluated for each segment.

'Get Value from Code Combination' retrieves a value from a specific segment and stores the result in an item attribute. It uses the activity attributes 'Code Combination ID', 'Segment Identifier', 'Segment', and 'Attribute to assign value'. 'Code Combination ID' is usually set to an item attribute which then must be populated prior to executing this function activity. 'Attribute to assign value' must be set to an item attribute. The function finds the specified segment in the specified code combination and copies the value to the specified item attribute.

'Copy Value from Other Structure Code Combination' retrieves a value from a specific segment in a combination in another accounting flexfield and stores the result in an item attribute. It uses the activity attributes 'Structure Number', 'Code Combination ID', 'Segment Identifier', 'Segment', and 'Attribute' to assign value'. 'Code Combination ID' is usually set to an item attribute which then must be populated prior to executing this function activity. The function finds the code combination id, extracts the segment specified and stores the result in the item attribute specified in 'Attribute to assign value'.

Because the above routine stores the retrieved value in an item attribute, the additional rules detailed for 'Copy Segment Value from Other Structure Code Combination' do not apply.

'Is Code Combination Complete' checks to see if the combination being built has values in each segment. It uses the activity attribute 'Check only for required segments'. If this attribute is set to 'False' all segments are checked. If this attribute is set to 'True' only the required segments are checked (Note: The Reporting Attributes Flexfield does not have to have values in every segment).

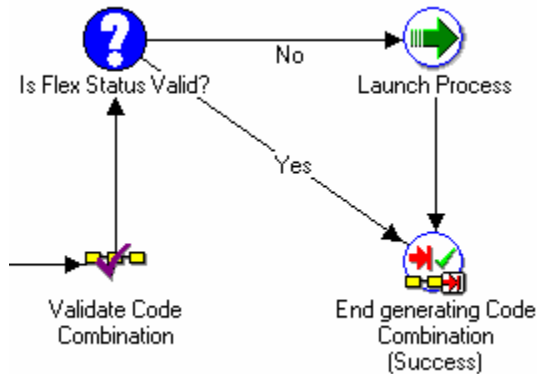
'Validate Code Combination' validates the generated code combination according to the values assigned to the activity attributes 'New code combinations are allowed' and 'Validation Type'. If 'New code combinations are allowed' is set to 'True' and the accounting flexfield has Dynamic Insertion turned on, then the fact that the generated combination does not already exist does not constitute an error. Conversely, if 'New code combinations are allowed' is set to 'False' and/or the accounting flexfield has Dynamic Insertion turned off, then the fact that the generated combination does not already exist is logged as an error.

If 'Validation Type' is set to 'Generate Code Combination ID', then all segments must have a valid value and the resulting combination must be valid (subject to the checking dictated by 'New code combinations are allowed'). If 'Validation Type' is set to 'Validate Segments with Values only', then the only check done is whether the value assigned to each segment is enabled for that segment.

Note that if the combination is new and allowed, this procedure does not do the insert nor does it generate a CCID. It merely sets the CCID to -1. It is up to the calling process to insert the combination and generate the CCID.

This routine becomes a pivot point in the account generator as it sets many of the added item attributes. Each of the FND_FLEX_SEGMENTn is retrieved and the values concatenated together and stored in FND_FLEX_SEGMENTS. If the validation fails, FND_FLEX_STATUS is set to 'INVALID', the reason why is stored in FND_FLEX_MESSAGE, FND_FLEX_CCID is set to 0, FND_FLEX_DATA and FND_FLEX_DESCRIPTIONS are set to NULL, FND_FLEX_NEW is set to 'N'. If the validation is successful, FND_FLEX_STATUS is set to 'VALID', FND_FLEX_CCID contains either -1 (new combination) or the found combination_id, FND_FLEX_DATA contains the concatenated id of each segment, and FND_FLEX_DESCRIPTIONS contains the concatenated description of each segment. If the combination is a new one, then FND_FLEX_NEW = 'Y', else 'N'.

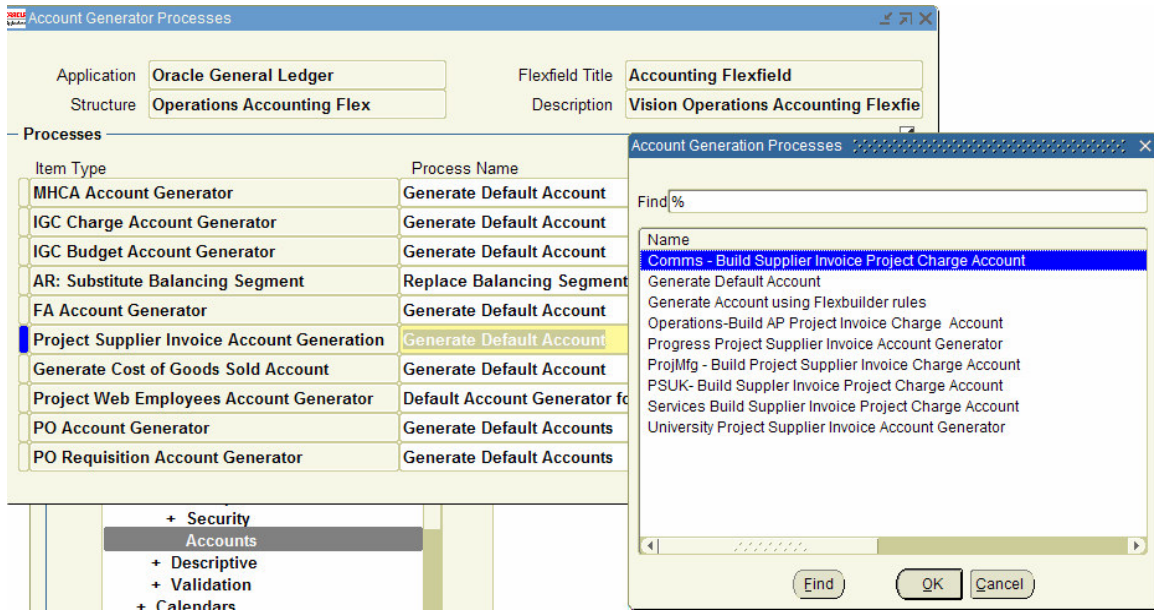
As stated earlier, Validate Code Combination does not return a result code, so you cannot branch based on whether FND_FLEX_STATUS is 'VALID' or 'INVALID'. But as the following shows, you can insert a custom node and do your own checking.



Most account generators display the error message when they fail and return enough information for users to fix any issues. However, the OECOCS generator, which is called during the AR invoice interface, does not. It merely records in the log file that it failed. It does not even identify the order or line that failed. This makes it very difficult to debug. (Note: the seeded generator simply moves the COGS account from the item referenced on the order line to the invoice line, and since the COGS account for the item is a required field, there is never a reason the seeded generator would fail.) Modifying the AR invoice import program has all the problems of modifying any Oracle program, patches, maintenance, etc. But the account generators were designed to be modified.

Register Your Customization

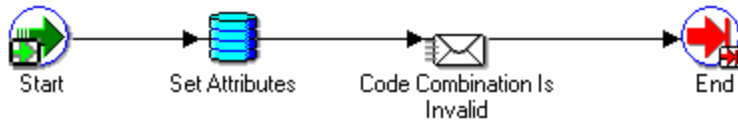
To customize this or any account generator, open the itemtype seeded by Oracle. Note that the Standard and Standard Flexfield Workflow itemtypes are copied in as well. Ensure your customization level is set to 100 or higher. Copy the default process and rename it to your custom name. This can be accomplished by right-clicking the default process, select copy. Then right-click 'Processes' and select paste. The Properties page will open. Type in the new Internal Name, Display Name and (optionally) Description. Now you can add any customization you choose, as long as you follow the rules described earlier. Once the customization is finished, save the workflow back to the database. Then sign into the applications, and select a responsibility that has the Setup | Flexfields | Key menu. Select Accounts. Query the records for the Accounting Flexfield and arrow down until the key flexfield for your set-of-books is displayed. Select the appropriate account generator and change the default process to your custom process.



So how to send a notification, when notifications are prohibited?

In the above customization, the PL/SQL routine 'Is Flex Status Valid', if FND_FLEX_STATUS is 'VALID', then the routine returns 'Y' and the account generator branches to the end. If FND_FLEX_STATUS is 'INVALID', then the routine retrieves the header_id, line_id, org_id, and inventory_item_id of the order (item attributes for the workflow), and FND_FLEX_SEGMENTS (the concatenated segments). Then an attribute is set to hold the role of the person who will be responsible for clearing any flexfield errors. Another attribute is set to the concatenated segments and a 3rd attribute is set to the concatenation of the line_id, header_id, inventory_item_id, operating_unit and SYSDATE. Each field is separated by a colon. SYSDATE is used to provide uniqueness. The result is set to 'N'. The next node launches a custom workflow. The itemkey is the concatenated field formed by the line_id, header_id, inventory_item_id, operating_unit and SYSDATE. The user_key is the concatenated segments. The owner is the role that will resolve the error.

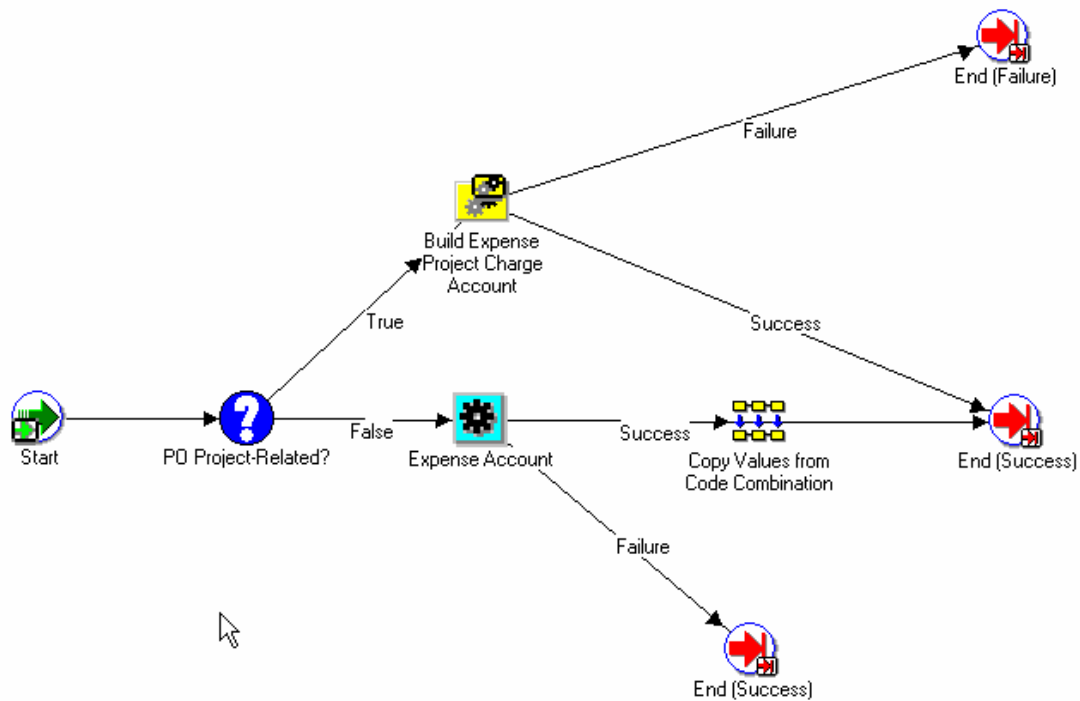
The custom workflow (shown below) calls a PL/SQL routine to break out the itemkey, userkey and owner to item attributes. The owner becomes the performer. Then a notification is sent that includes the information extracted from the account generator. The fields itemkey and userkey are limited to 240 characters, so you can even modify the 'Is Flex Status Valid' to pass part or all of FND_FLEX_MESSAGE.



The same logic could be used to launch the same custom workflow before transversing to the 'Abort Generating Code Combination' node. But note that many of the hidden attributes are not set at this point (except for FND_SEGMENTn), so the logic to build the itemkey and userkey may be a bit more complex.

Projects Account Generators

The picture shown at the beginning of the article shows the account generator used by AP invoices that reference projects. Below is the 'Build Expense Charge Account' from POWFPOAG, the PO Account Generator. Note that if the PO references a project, it branches to the 'Build Expense Project Charge Account' process, which is shown below the 'Build Expense Charge Account process'. And this process only has a Start and End node. Thus if you are using Projects, every account generator must be customized.





PA account generators do provide one more standard function that can be used, 'Segment Lookup Set value'. If this function is not already seeded in the account generator you start with, open PAAPIN VW, 'Project Supplier Invoice Account Generator', and copy is from there. This procedure has two activity attributes, 'Lookup Set Name', and 'Intermediate Value'. It also requires that you define an item attribute with the internal name 'LOOKUP_SET_VALUE'. You can pass this procedure the name of a Projects Lookup Set and a lookup code and the procedure will set the item attribute 'LOOKUP_SET_VALUE'. This value can then be passed to a procedure such as 'Assign Value to Segment' and used to build the account combination.

Debugging

Account Generators do not log records in the workflow runtime tables, so you cannot view the results through the monitor. However, if you set the value for the profile option 'Account Generator:Run in Debug Mode' to Yes, then history will be generated and you can trace what happens as you would for any workflow. Note: as mentioned earlier, even with the profile option set to Yes, you will not see the "hidden" attributes. This is the best way for the account generators that consist only of the standard flexfield activities.

For some workflows, Oracle provides scripts to help you with the debug process. For example, MetaLink note 159998.1 has a script to help with debugging OECOGS. And MetaLink note 113492.1 has an excellent white paper on this account generator. The best place to look for these scripts is Top Tech Docs. Find your product and then you can use the Search tab and search for "account generator".

Others have the ability to log "breadcrumbs" about every step that is taken. Generally you'll have to recompile the package used to set a global variable to True. And you'll have to have UNIX access to see the DBMS_OUTPUT that the package generates. To see if the account generator you are working on has these capabilities, look at the workflow and see if it uses packages other than the Standard Flexfield package. If so, open the package and scan the code.

Conclusion

For certain projects, Account Generators are required workflows. However, you are not required to accept Oracle's definition. Customizing these workflows allows you to fit these rules to any unique requirements that your company has. Additionally you now know a way to notify someone when errors occur in an account generator and the error message is not transmitted elsewhere. Remember Project-related generators MUST be customized. Start by making simple customizations or using just the standard activities. Once you see expected results, then you can add to your workflow until the accounting generated is exactly what is needed and required.

About the Authors

Karen Brownfield of Solution Beacon has 25+ years experience programming, installing, and managing applications used in various industries including Chemicals, Entertainment, Defense, Recruitment, and Hospitality. The past 15 years she has focused on Oracle Applications specializing in Financials and Workflow as functional specialist, programmer, team lead, and/or project manager. Karen is also a member of the OAUG board of directors, serving in multiple capacities in the last 10 years, including President, Past President, other Executive Committee roles, and has presented many papers and training sessions at the various OAUG and user group conferences locally, regionally, and internationally. Karen can be reached by email at kbrownfield@solutionbeacon.com.

Venkat Kancherla of Solution Beacon is a developer with many years experience customizing and adapting Oracle applications to meet the needs of customers in various industries. He has customized the account generators for Purchasing, Projects, Fixed Assets and Order Management. He has also installed workflow as a standalone product for clients who do not use the E-Business Suite. Venkat can be reached by email at ykancherla@solutionbeacon.com.