Steve Rubenstein's Recurring Billing

List Targets

Version 1.1
Last Updated October 15, 2013

This document describes the method used for listing targets based on various search/filtering criteria. It describes how the list is generated and why various decisions were made.

The targets where this method applies are:

- companies
- users
- invoices
- subscribers
- products
- payments
- refunds
- payment credits
- cobrands
- vendors
- affiliates
- commission plans
- pricing plans
- contact management emails
- newsletters
- newsletter subscribers
- sales commissions
- tasks

For the purposes of this document, the code for generating the list of products will be used as an example. The primary code file that controls this process for products is:

control/c_product/control_listProducts.cfm

1. The initial `CFINVOKE` tags simply select the various filtering options for products. They are not relevant to the general method – they apply only to the list of products.

2. On (or around) line 19, we generate the default filtering parameters just like for all form processing:

```
<CFINCLUDE Template="formParam_listProducts.cfm">
```

All formParam files for listing products have the following code in common:

```
<CFINCLUDE Template="../../include/function/act_urlToForm.cfm">
<CFPARAM Name="Form.queryOrderBy" Default="">
<CFPARAM Name="Form.queryPage" Default="1">
<CFPARAM Name="Form.queryDisplayPerPage" Default="20">
```

The `act_urlToForm.cfm` file does exactly what its name suggests – it takes all URL variables and creates default form variables. This is necessary since the list of targets can be generated via a form submission or a URL. This generally occurs in one of three instances:

- Re-ordering the list of targets by a designated column at the top of each column.
- Clicking to continue to the next page via the next/previous page options at the bottom.
- If selecting the option to jump directly to targets beginning with a particular letter. (This option only exists if ordered by a column for which this has been enabled.)

The three form variables are used in all target lists:

- `queryOrderBy` – The field by which the targets are ordered. This is generally the actual database field name with an appended "`_d`" for descending order. To order by a user's name, `lastName` is used but it also automatically orders by last name and then first name. If changing the order in which the targets are listed, the list is re-displayed beginning on page 1 of the new order.
- `queryPage` – The current page being displayed. If jumping to a particular letter, the page number is approximately, so the previous/next page may contain records on that page.
- `queryDisplayPerPage` – The number of targets to display per page. If 0, all targets are displayed on a single page.

3. On or around line 31 is the following code:

```
<CFSET Variables.formName = "productList">
<CFINCLUDE Template="../../include/function/fn_listObjects.cfm">
<CFINCLUDE Template="../../view/v_product/form_listProducts.cfm">
```

The form name is necessary for JavaScript functions.

The `fn_listObjects.cfm` file contains two ColdFusion functions used to toggle the basic and advanced search forms and to ensure that only one Yes/No checkbox for each option is checked at a time, i.e., that clicking the "Yes" checkbox automatically unchecks the "No" checkbox if checked.

The `form_listProducts.cfm` file is the actual search filter form for the target list. It contains both a basic and advanced search form. They are separate forms with their own form tags, and the user can toggle between them.

If a user submits a search form, the system automatically tracks whether it was a basic or advanced form. Thereafter during the session, the system will continue to display either the

basic or advanced search form until the user submits the other search form type. All sessions default to the basic search form to start.

4. On or around line 35 is the following code:

```
<CFINCLUDE Template="../../view/v_product/lang_listProducts.cfm">
<CFINCLUDE Template="formValidate_listProducts.cfm">
```

The `lang_listProducts.cfm` stores the error messages that may be displayed if any errors are encountered while validating the search/filtering options that the user selected. The validate file performs the actual validation.

Unlike the normal form processing for creating or updating a target, the form validation occurs before the targets are displayed without having to first submit the form. This is for three reasons:

- The search/filtering criteria may have been sent via URL instead of a form.
- Even for the default list of targets before having selected any filtering criteria, the criteria must still be validated to be safe.
- If there are any errors in the filtering criteria, the errors will be listed below the search form and the list of targets will not be displayed.

5. On or around line 38 is the following code:

```
<CFIF Variables.isAllFormFieldsOk is False>
      <CFINCLUDE Template="../../view/error_formValidation.cfm">
<CFELSE>
```

This follows the same principle as the general form processing. If there are validation errors, we display the errors. Otherwise, we continue. (Note: There is an ending `</CFIF>` tag at the end of the file.)

6. The code until line 66 is specific to listing products. On line 66 is:

```
<CFSET Variables.queryViewAction =
"index.cfm?method=#URL.control#.#URL.action#&queryDisplayPerPage=#Form.queryD
isplayPerPage#">
```

This set the default action URL that can be used to re-display this list of targets with the specified search/filtering options. The actual filtering options will be appended to this variable as they are processed.

7. On line 71 begins the code that will both generate the URL mentioned above and the query parameters sent to the query which will actually select the targets. The query parameters are stored in a ColdFusion structure: qryParamStruct. The query parameters are submitted for queries via the `CFINVOKE ArgumentCollection` method. This enables re-using the query parameters for the 2-4 queries which use these parameters without re-processing them.

The companyID argument is the client (or primary) company that the user is logged in as. This restricts the list of targets returned to those owned by that company. In many instances, companyID_author is used instead of companyID. For products, since there is no need to reference a customer companyID, the product table uses the companyID. But for invoices, both the companyID_author of the client and the companyID of the client's customer company are stored, so companyID_author is used instead. (Yes, this is not the most consistent but the idea was that if we are only storing the companyID of the client *or* customer in a table, there is no reason to label the client companyID as companyID_author.

After the companyID or companyID_author fields, the remaining fields are processed to determine whether they exist and have a valid value for searching. We are not re-validating the values except to ensure they are seemingly valid values before sending them to the component, which will return an error if a variable with the wrong datatype is sent in.

In the process of sending the filtering criteria as inputs, the criteria are also added to the queryViewAction variable from step 6. To reduce the amount of redundant code, the system loops thru similar variables types, e.g., all boolean values, to determine if the form field exists and if it is a valid value for that data type. If yes to both, the field is sent to the component for inclusion in the WHERE clause and is added to the queryViewAction URL field.

8. On line 137:

```
<CFSET Variables.exportResults = False>
<CFIF IsDefined("Form.submitExportResults")
      and Application.fn_IsUserAuthorized("exportProducts")
      and IsDefined("URL.queryDisplayResults")
      and URL.queryDisplayResults is False
      and IsDefined("Form.exportResultsMethod")
      and ListFind("excel,tab,xml", Form.exportResultsMethod)
      and IsDefined("Form.exportResultsFormat")
      and ListFind("data,display", Form.exportResultsFormat)>
  <CFSET Variables.exportResults = True>
  <CFSET Form.queryDisplayPerPage = 0>
</CFIF>
```

This code determines whether the user has selected to export the target list rather than display them in the browser. The export includes all targets which meet the criteria in the selected order, regardless of which page is currently being viewed. The above CFIF statement validates that the user has permission to export this target type, that they clicked the "Export" submit button, and that all export options selected are valid. The default value is False, but if all conditions are met, then export is set to True.

This code is separated because if the targets will be exported, queryDisplayPerPage must be set to zero.

9. On line 153:

```
<cfinvoke
      Component=" #Application.billingMapping#data.Product"
      Method="selectProductList"
      ReturnVariable="qry_selectProductList"
      argumentCollection="#qryParamStruct#">
  <cfinvokeargument Name="queryOrderBy" Value="#Form.queryOrderBy#">
  <cfinvokeargument Name="queryDisplayPerPage"
      Value="#Form.queryDisplayPerPage#">
  <cfinvokeargument Name="queryPage" Value="#Form.queryPage#">
  <cfif IsDefined("Form.queryFirstLetter")
        and Form.queryFirstLetter is not "">
      <cfinvokeargument Name="queryFirstLetter"
            Value="#Form.queryFirstLetter#">
  </cfif>
  <cfif Variables.displayProductCategoryOrder is True>
      <cfinvokeargument Name="displayProductCategoryOrder"
            Value="#Variables.displayProductCategoryOrder#">
      <cfinvokeargument Name="categoryID" Value="#URL.categoryID#">
  </cfif>
</cfinvoke>
```

This function is the actual query and returns a query of the targets that meet the criteria in the given order for this particular page. (Or if exporting, all records are selected.) For those targets which support the ability to jump to a particular letter, the requested letter can also be sent as an input. For some targets, additional fields may also be sent.

The `Form.queryOrderBy` value refers to the column by which to order the results. It is generally the column name for ascending order or with an appended "`_d`" for descending order. Within the query code, this value is validated both for security and to ensure the results are ordered by an appropriate column.

10. On line 175:

```
<CFIF Variables.exportResults is True>
  <CFINCLUDE Template="act_exportProductList.cfm">
<CFELSE>
```

Now that we have the list of targets, we determine whether to display the results in the browser or export the results. If exporting, the export file is called. The export process is beyond the scope of this document, but the export file contains target-specific export options and then calls the general export script.

Note: All functionality beyond this point is only encountered if displaying in the browser (i.e., not exporting).

11. On line 184:

```
<cfinvoke
      Component="#Application.billingMapping#data.Product"
      Method="selectProductCount"
      ReturnVariable="qryTotalRecords"
      argumentCollection="#qryParamStruct#" />
```

This code returns the total number of targets that meet the search/filtering criteria, regardless of the order-by field or the current page. This is the second instance where the `WHERE` query parameters from qryParamStruct are used. The total number of targets is necessary both because it is displayed at the top of the results and to determine the total number of pages based on the number of targets displayed per page.

12. Beginning on line 186 is the code for jumping to a particular record based on the first letter. This option is only available for those targets which have a name (or other field) where this makes sense. For instance, it is logical to jump to products where the product name begins with a particular letter, but it is not appropriate for tasks which do not have a formal name. Furthermore, when viewing the list of products, the alphabet jump-to options are only available for appropriate order-by fields. If the list of products is ordered by price, the jump-to options are not listed as an option.

```
<CFIF Not ListFind("productName,productName_d", Form.queryOrderBy)>
  <CFSET Variables.displayAlphabet = False>
  <CFSET Variables.alphabetList = "">
<CFELSE>
  <cfinvoke Component="#Application.billingMapping#data.Product"
      Method="selectProductList_alphabet"
      ReturnVariable="qry_selectProductList_alphabet"
      argumentCollection="#qryParamStruct#" />

  <CFSET Variables.displayAlphabet = True>
  <CFSET Variables.alphabetList =
      ValueList(qry_selectProductList_alphabet.firstLetter, "|")>

  <CFIF IsDefined("Form.queryFirstLetter")
      and Form.queryFirstLetter is not "">
    <cfinvoke
        Component="#Application.billingMapping#data.Product"
        Method="selectProductList_alphabetPage"
        ReturnVariable="recordCountBeforeAlphabet"
        argumentCollection="#qryParamStruct#">
      <cfinvokeargument Name="queryFirstLetter"
            Value="#Form.queryFirstLetter#">
      <cfinvokeargument Name="queryDisplayPerPage"
            Value="#Form.queryDisplayPerPage#">
    </cfinvoke>

    <CFSET Form.queryPage = 1 +
      (recordCountBeforeAlphabet \ Form.queryDisplayPerPage)>
  </CFIF>
</CFIF>
```

The main `CFIF` statement determines whether the current field by which the targets are ordered is a valid jump-to alphabet field. If not, the alphabet with jump-to links should not be displayed at the bottom of the list.

The `alphabetList` variable is the list of letters for which there is at least one target record meeting the search criteria where the order-by field name begins with that letter. In the product example, if ordered by product name, it is the list of letters for which there is at least one product meeting the search criteria where the product name begins with that letter.

If the fields are ordered by a valid jump-to field, then we select the `alphabetList`. For those targets which have more than one order-by field which can be jumped to via the alphabet, such as companies (via company name or last name of primary company contact) the order-by field must be determined. This is slightly redundant with the order-by clause string generated in step 8, but is done separately because the ascending/descending does not matter and the order-by clause string may contain more than one field. In the case of products though, there is only one valid jump-to field – productName.

Finally, we must determine whether the user actually clicked a jump-to letter so as to jump to targets beginning with that letter. If so, we must determine the approximate page on which the first jump-to record would be listed. This will not be exist since the page might typically start before the selected letter. The query results from step 10 already took the jump-to letter into account, so the query results already begin with the requested letter.

This is the third instance of using the `WHERE` query search parameter string and the second instance of using the `ORDER BY` variable string.

12. We are now just about read to display our target records. We now create a few simple variables that represent the full URL to this list including order, first and last target records displayed on this page, the total number of targets, and the total number of pages. Beginning on line 205:

```
<CFSET Variables.queryViewAction_orderBy = Variables.queryViewAction
      & "&queryOrderBy=#URLEncodedFormat(Form.queryOrderBy)#">
<CFSET Variables.firstRecord = (Form.queryDisplayPerPage *
      DecrementValue(Form.queryPage)) + 1>
<CFSET Variables.totalRecords = qryTotalRecords>
<CFSET Variables.lastRecord = Min(Form.queryDisplayPerPage * Form.queryPage,
      Variables.totalRecords)>
<CFIF (Variables.totalRecords mod Form.queryDisplayPerPage) is 0>
  <CFSET Variables.totalPages = Variables.totalRecords \
      Form.queryDisplayPerPage>
<CFELSE>
  <CFSET Variables.totalPages = (Variables.totalRecords \
      Form.queryDisplayPerPage) + 1>
</CFIF>
```

13. On line 215, we include the ColdFusion functions that are used to display the top and bottom rows of the results table that include the record counts and column headers at the top; and the

next/previous page navigation links, go-to page form, and jump-to alphabet list (if appropriate) at the bottom.

```
<CFINCLUDE Template="../../include/function/fn_DisplayOrderByNav.cfm">
```

14. We now determine which columns to display in the results based on permissions and why the list was generated, i.e., whether you are viewing those targets directly or via another target that has a different purpose, such as selecting a product to add to as a line item to an invoice. For thos targets where the list may have multiple purposes like this, there is generally a `CFSWITCH` based on the `ACTION` requested, i.e., the page we are currently viewing.

15. Within the display file, the functions to display the table, including top and bottom rows described in step 13, are called. The display file is beyond the scope of this document and is fairly self-explanatory. The only points worth mentioning are that the top and bottom rows are displayed by passing in the appropriate values to their respective functions in `fn_DisplayOrderByNav.cfm`. The last value passed for the top row is whether the results table should be hard-coded to be 800 pixels wide or whether it can stretch (or compact) based on the actual results.

Final Note:

The code to display the list of targets was formerly contained within the component itself. The function that generates the `WHERE` query parameters originally called the `selectTargetList` function which performed all above features and called the display file. This method was simpler because all functions could simply be included rather than called as functions. However, there were several down-sides to it, which may or may not be ColdFusion-specific:

- ColdFusion returns an error if you call the same `CFSCRIPT` code block twice within the same request. Because components are cached, after the component was cached the first time, it would be necessary to test everytime whether a function defined within the `CFSCRIPT` block had already been defined – which was everytime after the component was cached.
- No other display files are called via components, making it inconsistent with the rest of the code.
- It violates the concept of using components purely as functions that return a value based on inputs.
- The control files are designed to control the workflow. Moving that logic to the component was a bit silly.

Components are cached for scalability and for web services. For some strange reason, when a ColdFusion Component is accessed as a web service, it cannot initiate another component as an object (i.e., it cannot access othern ColdFusion Components) unless that component is stored in the Application scope (or other scope). This means that all components accessed by web services must be cached in the application scope.