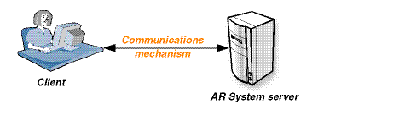
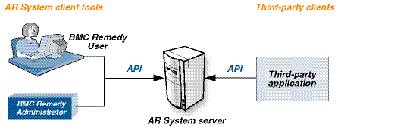
\*\* A client-server application model is a combination of the following items:

* A software component with a user interface on a client computer
* A software component on a server that processes and stores information
* A communications mechanism between the client and the server.

The user works at the client computer, which is "serviced" by the server.   
  
**Simple client-server model**   
  
  
Typically, the client and server components are on different computer systems, and the communications mechanism is based on some form of networking technology, but it is possible to have both the client and server on the same host simply communicating between processes.   
  
To make the development of client-server applications easier, an application programming interface (API) is used. Typically, the server is designed first, and its functionality is structured into a set of "commands." These commands are programmed into an API, which becomes the "language" for interacting with the server. When a client wants the server to do something, it makes a request through the API. The API handles all of the communications and operating system functions, gets the server to perform the task, and returns the appropriate information to the client. When the client software is developed, the programmer does not need to know any of the details of the server environment or communications mechanisms.   
  
The BMC Remedy AR System server has a fully defined API that is common to all server platforms, both Windows and the UNIX platforms. The BMC Remedy AR System client tools all use this API for interaction with the servers. They all speak the same language and are completely interchangeable. A client on any platform can work with a server on any platform. As long as a client can connect to a BMC Remedy AR System server, it can communicate all its requests and receive the replies using this common language. The client does not need to know on which platform the server is running. It also does not need to know about other clients that are using the same server.   
  
The BMC Remedy AR System API is defined by a strict set of programming functions and data structures, which are documented in [BMC Remedy AR System C API functions](https://docs.bmc.com/docs/display/ars81/BMC+Remedy+AR+System+C+and+Java+API+architecture) and [Data structures](https://docs.bmc.com/docs/display/ars81/Data+structures). The API is implemented as a C library and associated files that can be linked into your programs. Third-party programs linked to the BMC Remedy AR System API library become clients to a BMC Remedy AR System server.   
  
**BMC Remedy AR System API**   


->You can use active link guides and filter guides to control the order of workflow actions and organize a related set of workflow objects.

->Active links allow you to create workflow designed for user interaction. You can use buttons and field menus with active links to assist the user

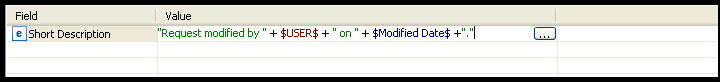
->Transaction: While changing in the DB

->To set a static value in the field, enclose the value in quotes in the expression.

\*\*For example, to set a field value that tracks what user modified the form and when, creates an expression like the following:

"Request modified by " + $USER$ + " on " + $Modified Date$ +"."

In this example expression, “Request modified by “ and “ on ” are static values, $USER$ is an AR System keyword that contains the current user’s login ID, and $Modified Date$ is a field in the current request. The “+” signs are operators that concatenate the parts of the expression.





In the Expression Editor Dialog box, build an expression to define the value:

􀂄 To use a value from another field in the current request, select the field from the Available Fields list.

􀂄 To use the current value of a keyword, select it from the Keywords list.

􀂄 To set a static value in the field, enclose the value in quotes in the expression.

For example, to set a field value that tracks what user modified the form and when, creates an expression like the following:

"Request modified by " + $USER$ + " on " + $Modified Date$ +"."

In this example expression, “Request modified by “ and “ on ” are static values, and $USER$ is an AR System keyword that contains the current user’s login ID, and $Modified Date$ is a field in the current request. The “+” signs are operators that concatenate the parts of the expression.

􀂄 In Set Fields, Service, and Push Fields actions, you can also use the results of data operations in the expression. For more information, see the descriptions of those workflow types in this document.

4 When the expression is complete, click OK to close the Expression Editor Dialog box and enter the expression in the field mapping table.

\*\*Commit Changes action

Use the Commit Changes action in: **Active links**

**The Commit Changes action two has functions:**

\*When used with a dialog box, Commit Changes works with the Open Window and Close Window actions to capture the data entered in the dialog box. In this case, the Commit Changes action pushes predetermined values from the dialog box to fields on the parent form, but does not initiate a save to the database.

\*When used with a regular form, join form, view form, or vendor form, the Commit Changes action applies the changes in the form and performs the major form action (for example, Submit, Search, or modify). In this case, the Commit Changes action does submit data to the database.

\*\*Push Fields action

Use the Push Fields action in:

--> Active links

-- > Filters

-- > Escalations

The Push Fields action enables you to automate updates to the database. You can transfer values from selected fields in the current request to another request in a different form or in the same form. The Push Fields action can update or create a request. You can push values from all field types, including tables, hidden fields, fields not in the active view, or fields in no views at all. However, you cannot push values to non-data fields, such as table fields, column fields, and panel holders. Each Push Fields action can push data from the current form to one other form,

although you can update multiple requests in that form. To push data from the current form to more than one form, you must create a separate Push Fields action for each destination form.

->**To define a Push Fields active link, filter, or escalation action**

s1 Right-click the If Action or the Else Action panel header.

2 Choose Add Action > Push Fields.

3 From the Data Destination list, select SERVER or SAMPLE DATA:

􀂄 To specify a specific server and form, select SERVER, and go to step a.

􀂄 To determine the server and form at runtime, select SAMPLE DATA, and go to step b.

a (Data Destination is SERVER) Select the Server Name (active links only) and Form Name to push data to.

Servers that appear in the Server Name list are those to which you are currently logged in. In the Form Name field, the forms that appear in the Form Selector dialog box are from the server specified in the Server Name field. Filters and escalations can push data only to forms on the server where the workflow is running.

b (Data Destination is SAMPLE DATA) Define the sample and runtime server and form names or values:

􀂄 In the Sample Server Name (active links only) and Sample Form Name fields, select the sample server and destination form name to use to design the Open Window action.

􀂄 In the Runtime Server Value and Runtime Form Value fields, identify the fields on the current form or the AR System keywords that will identify the destination server and form at runtime.

For more information, see “Using a dynamic data source or destination in workflow” on page 71.

4 In the Qualification field, enter a qualification, if any, to specify the request to which the Push Fields action will push the data.

Click the ellipsis button to use the Expression Editor Dialog box, or type the expression directly in the Qualification field. For more information about creating qualifications, see Chapter 3, “Building qualifications and expressions.”

􀂄 To select fields in the Expression Editor dialog box, first select either the current form or the destination form in the Form list. The list of Available Fields changes to display fields from the selected form.

􀂄 As you build the expression, fields from the current form are delimited with dollar signs ($). Fields from the destination form are delimited with single quotation marks ( ' ).

􀂄 To use content assist to build the expression, press Ctrl+Space to get a list of all possible choices. Alternatively, type a dollar sign ($) to get a list of fields from the current form and keywords. Type a single quotation mark ( ' ) to get a list of fields from the destination form.

If you associated additional forms and want to use a field from a form other than the primary form, enter the appropriate field ID in the appropriate format, for example, ‘$fieldID$' or ‘fieldID’. The field must exist on the form.

**NOTE**

Using a qualification causes the Push Fields action to run a query to obtain the list of matching requests, so make sure to optimize the Push Fields qualification for best system performance. See “Creating efficient qualifications” on page 56.

􀂄 To create a Push Fields action that does not search for existing records but instead always creates a request, do not enter a qualification. Instead, select the following values:

􀂄 In the If No Requests Match field, select Create a New Request.

􀂄 In the If Any Requests Match field, select Take No Action.

5 From the If No Requests Match list, select an option to control how the system responds when no matches are found in the destination form:

􀂄 **Display ‘No Match’ Error**—Returns an error message and stops processing.

􀂄 **Take No Action**—Skips this action and proceeds to the next action.

􀂄 **Create a New Request**—Uses the data specified in the Push Fields action to create a request.

6 From the If Any Requests Match list, select an option to control how the system responds when multiple matches are found in the selected form:

􀂄 **Display ‘Any Match’ Error**—Returns an error message and stops processing.

􀂄 **Modify First Matching Request**—Pushes data to the first request that meets the qualification.

􀂄 **Modify All Matching Requests**—Pushes data to every request that meets the qualification.

􀂄 **Take No Action**—Skips this action and proceeds to the next action.

7 Use the field mapping table to map the fields and values to push from the current form to the destination form as follows:

􀂄 To use matching field IDs to map the values of all matching fields from the current form to the destination form automatically, select the Matching IDs option. The field mapping table is disabled, and you cannot select specific field mappings.

In this case, all matching field IDs (except for table columns, panel holders, and core fields such as Modified Date and Request ID) are automatically set in the destination form from the values in the current form.

**TIP**

-When you use Matching IDs, the values are dynamic. This means that only the setting is stored as part of the Push Fields action, and not the -actual values. Therefore, you can add data fields to these forms at a later stage and the action uses them when executed.

-To use the Auto Map dialog box, click Auto Map.

-The Auto Map dialog box opens with all matching fields entered in the list. To delete a field you do not want to update, select the field and click Remove. After making selections in the Auto Map dialog box, click OK.

**TIP**

-You can use Auto Map to map fields according to the field name or field ID. When you use Auto Map with field names, the mappings are dynamic. However, when you use Auto Map with field IDs, the mappings are stored with the form. In this case, if you later add fields to the forms, they will not be part of the action and must be added manually.

􀂄 To map specific fields and to use an expression to define the value, follow the steps described in “Mapping fields to values” on page 68.

-In the Field column, select the fields in the destination form to which the Push Fields action will push data. In the Value column, use an expression to define the values that the Push Fields action will push to the fields in the destination

Form.

-To define the value, you can use field values from the current form, keywords, static values, or the result of a function. In active links, you can also use the result of a DDE operation. For information about using the result of a function, see “Assigning values are using function results” on page 251. For more information, see Integration Guide, “Using active links with DDE,” page 290.

8 Save the active link, filter, or escalation.

**Using a dynamic data destination**

To create a Push Fields action that uses runtime information to identify the destination server and form, specify SAMPLE DATA instead of SERVER for the data destination. At runtime, the field values are pushed to a server and form defined by workflow or selected by the user. See “Using a dynamic data source or destination in workflow” on page 71.

**NOTE**

Dynamic Push Fields actions cause browsers to perform extra HTTP fetches and cause the AR System server to perform extra queries to determine the data types for remote fields. To enable the mid-tier to fetch these data types ahead of time and avoid performance degradation, avoid using field references to store the server or form names

\*\*From the If No Requests Match list, select a handling option to control how the system responds when no matches are found in the selected form.

􀂄 **Display ‘No Match’ Error**—Returns an error message and stops processing.

􀂄 **Set Fields to $NULL$**—Sets field values obtained from a qualification expression to NULL unless the values are static or are based on keywords.

For example, suppose a field value is based on this expression:

$fieldName$ + " on " + $SERVER$

If no matches are found, $fieldName$ is set to NULL, and only the static value and the value based on the keyword appear in the field:

on serverName

\*\*To specify one or more recipients, enter the recipient information in any of the following formats:

􀂄 **AR System user names**

􀂄 Alert notifications are entered in the Alert Events form, with the user name appearing in the User field.

􀂄 Email notifications are sent to the email address specified in the User form entry for the user.

􀂄 **AR System group names**—AR System takes the group name from the entry in the Group form, searches the User form for all users belonging to this group, and deliver the notification for each member.

􀂄 For Alert notifications, an entry is made in the Alert Events form for each group member.

􀂄 Email notifications are sent to the email address specified in the User form entry for each user.

**\*\*Wait action**

Use the Wait action in: Active links

The Wait action suspends a guide in BMC Remedy User so that the user can interact with a field. After making a response, the user can continue the guide by pressing Tab or by clicking a button. The user can also terminate the guide during a Wait action.

For example, to create a training guide, you can use a series of workflow actions to walk a user through a form, and use the Wait action to pause the guide while the user completes each field. For a detailed example see “Creating interactive guides” on page 155.

**NOTE**

The Wait has no effect in the web client or outside of a guide. Also, the Wait action does not work for the Search execution condition.

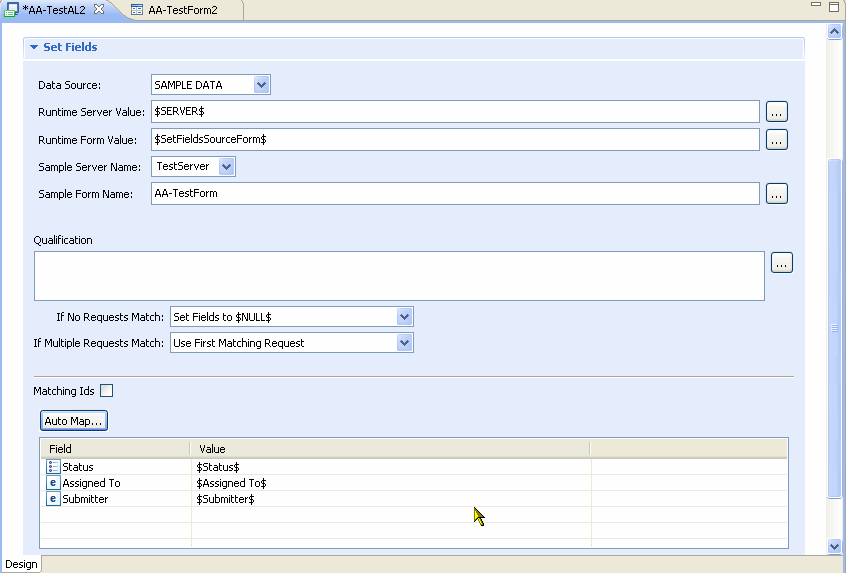
􀀀 **To define a Wait active link action**

1 Right-click the If Action or the Else Action panel header.

2 Choose Add Action > Wait.

3 In the Label for Continue Button field, enter the text you want to appear on the Continue button in the Prompt Bar in BMC Remedy User, for example, “Continue” (the default), “Next Step,” or “Finish.”

4 Save the active link or filter.



**\*\* Using active link guides in client-side table fields**

Active link guides are useful for creating workflow that walks you through the rows in a client-side table field. (For server-side table fields, see “Using a filter guide to loop through a table field” on page 160.) The workflow goes through every row in the table and then performs selected workflow actions on particular rows or sets of rows. For example, you can create workflow to find a specific row in a table field, then perform actions based on specific criteria. (For more information about optimization, see “Enhancing the performance of table loops” on page 154.)

You can also build workflow to:

􀂄 Skip rows, based on mathematical calculations on field values.

􀂄 Find a row that has changed values and perform an action on it, using the $ROWCHANGED$ keyword.

􀂄 Find a row that has been selected and perform an action on it, using the $ROWSELECTED$ keyword.

For more information about keywords, see Appendix A, “Operators, wildcards, keywords, and NULL values.”

The following example procedure creates simple loop that searches through all the rows of a table field until it finds a ticket with a specific user’s name on it. The workflow then sets a field with values from a column in the table field.

**NOTE**

This procedure assumes you already know how to create forms, fields, workflow, and guides.

􀀀 **To create an active link guide that loops through a table field**

1 Create a form (for example, **Loop Test**):

Add a button field (**Button**).

􀂄 Add a table field that includes at least the following fields as columns. Use the Tree/Table property in the Properties tab to add:

􀂄 Submitter (**Column**)

􀂄 Request ID (**Column2**)

􀂄 Save the form.

2 Create an active link (**Loop Test SF Active Link**):

􀂄 Associate it to the Loop Test form.

􀂄 Enter a Run If Qualification: **‘Column’ = $USER$**

This condition executes the active link only if the value of the Submitter field is set to the person who is logged in to BMC Remedy User.

􀂄 Add a Set Fields action and in the field mapping table, set:

􀂄 Name: **Short Description**

􀂄 Value: **$Column2$**

This action puts the Column2 value, which is the value of the Request ID field, into the Short Description field.

􀂄 Save the active link.

3 Create an active link guide (**Loop Test Guide**):

􀂄 Associate it to the form.

􀂄 Add the **Loop Test SF Active Link** to the guide.

􀂄 Save the guide.

4 Create an active link (**Loop Test Active Link Button**):

􀂄 Associate it to the form.

􀂄 Set the Execution Options to **Button/Menu Field** and select the **Button** field.

􀂄 Add a **Call Guide** action:

􀂄 In the Guide Name field, select **Loop Test Guide**

􀂄 In the Guide Field, select the table field added to the form in Step 1.

􀂄 Set Table Loop to “All rows.”

This action activates the guide and causes it to loop through the rows in the table field.

􀂄 Save the active link.

5 Log in to BMC Remedy User as user **Demo** and open the **Loop Test** form in New mode.

6 As a test, create several tickets, but only one with **Demo** as the value of the Submitter field.

7 Open the **Loop Test** form in Search mode.

8 Click the table field to refresh it. The tickets you just created appear as rows.

9 Click the button field on the form.

The active link guide is triggered and loops through all the rows in the table field until it finds a row with **Demo** as the value of the Submitter field. The workflow then fills in the Short Description field with the value of the Request ID field.

**Example from the AR System sample application**

In the AR System sample application, the button “Save Enrollee Edits” triggers the following actions:

􀂄 Clicking the button calls the **Sample:ReassignEditedRows active link.**

**􀂄 Sample:ReassignEditedRows contains** a Call Guide action, which calls the Sample:LoopReassignEnrollment active link guide.

􀂄 **Sample:LoopReassignEnrollment** is a table loop guide, which calls the active link Sample:LoopPushEnrollee.

􀂄 **Sample:LoopPushEnrollee** uses a qualification and a Push Fields action to locate the correct row or rows in the table and update them.

􀂄 When the guide **Sample:LoopReassignEnrollment** has completed looping through the table, it returns control to the **Sample:ReassignEditedRows active link** (the active link that called the guide).

**􀂄 Sample:ReassignEditedRows** executes its next action, which is a Change Field action that refreshes the table.

􀂄 **Sample:ReassignEditedRows** exits, and the form is now updated.

**Enhancing the performance of table loops**

With a large number of records, you might see slow performance in the table loop, especially as more records are selected. The following list suggests some tips for improving performance:

􀂄 Use data chunking in the table field. For example, in the Advanced Display tab of the Field Properties window, set the size of the chunk to 50. This means that the AR System processes only 50 records at a time.

􀂄 If you are not using data chunking, you can specify a maximum number of rows to be displayed in the table field, for example, 50. To set this property, enter a number of maximum rows in the Table Property tab of the Field Properties window.

For more information, see the Form and Application Objects Guide, “Working with tables,” page 233.

􀂄 In BMC Remedy User, set the Limit Number of Items Returned to some specified number, for example, 500 (by choosing Tools > Options > Behaviors).

􀂄 Make sure you define a Run If condition in your active link. Otherwise, the design of the table loop actually goes through every single record in the table and checks against the active links that are associated with that table.

􀂄 Limit the active links in the table loop to run only against rows that are selected. To do so, append `$ (backquote character followed by a dollar sign) to the end of the names of the active links contained within the table loop guide. (You do not need to change the name of the active link that calls the guide, the names of the table loop actions, or the name of the active link guide referenced by the call.

**\*\* Using interactive active link guides in BMC Remedy User**

One use of a guide is to help a user navigate through a series of interactive steps. To the user, a guide looks like any other application that you open in BMC Remedy User. When using BMC Remedy User with the Prompt Bar, a guide includes the Stop Guide and Continue buttons.

When the guide prompts a user for information, the user enters the information, and then clicks Continue (or presses the Tab key). Users can quit the guide at any time by clicking Stop Guide.

A guide that walks users through a form depends especially on the Wait active link action and the prompt bar in BMC Remedy User. To create the typical steps in a navigation-style guide, use the following active links in this sequence:

􀂄 **Change Field (Set Focus)**—Sets focus to a field and highlights the field.

􀂄 **Message (Prompt)**—Provides the user with instructions and information in the prompt bar.

􀂄 **Wait**—Temporarily suspends the guide while waiting for user input. These first three active links should be set for every field in the form.

􀂄 **Commit Changes**—After the user enters all the information on the form, saves the information and then creates a request.

Guides can also be used to set up an application environment. For example, you can create a guide that presets a form with tabs, specific colors, default field values, and so on.

Finally, you can create a guide that uses dialog boxes, much like a wizard. This is especially important in a web environment, because the Wait active link action does not work in the web client. Instead, you must create dialog boxes for user interaction within the guide. For more information, see the Form and Application Objects Guide, “Using a display-only form as a dialog box,” page 158.

**Creating interactive guides**

The following section describes how you can create an interactive (or navigational) guide. This example walks you through filling out three fields in a form. When you click the Help button, the interactive guide is activated. A halo appears around the first field you must enter information into, and instructions appear in the prompt bar. After following the instructions, you click the Continue button for more instructions.

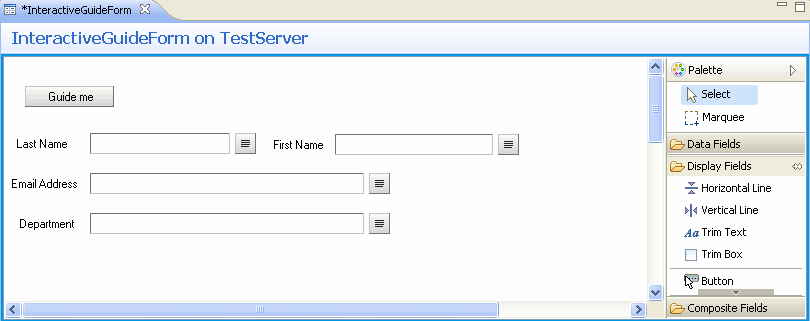
This example uses the Wait and Call Guide actions, but you can easily include more advanced actions (for example, the Go to Guide Label action). This procedure assumes you already know how to create forms, fields, and active link workflow.

**NOTE** Interactive guides are not supported in the web client.

**To create an interactive guide**

1 Create a sample form (for example, the **Interactive Guide Form**).

**Figure 5-4: Example form for interactive active link guide**



Users can complete the fields on this form by clicking Guide Me, which activates the interactive guide.

2 Create and save four active links, one for each of the fields on the form, for example

**ALG: Last Name**, **ALG: First Name**, and so on.

For each active link:

􀂄 Associate the active link to the **Interactive Guide Form**.

􀂄 Do not select any Execution Options and leave the Run If Qualification blank.

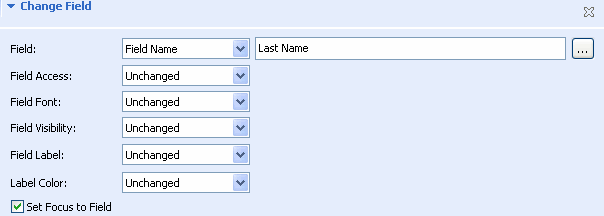
􀂄 Add a Change Field action:

􀂄 Select the appropriate field for the active link. For example, in

**ALG: LastName**, select the **Last Name** field.

􀂄 Select Set Focus to Field.

**Figure 5-5: Example Change Field action**



->Add a Message action:

􀂄 Enter an appropriate instruction for the field. For example, “Type the requester’s last name in the Last Name field, and then click Continue.”

􀂄 Select Show Message in Prompt Bar.

􀂄 Add a Wait action:

􀂄 For all active links except the one that will be last in the guide, leave “Continue” as the label for the Continue button.

􀂄 For the last active link, enter “Finish” as the label for the Continue button.

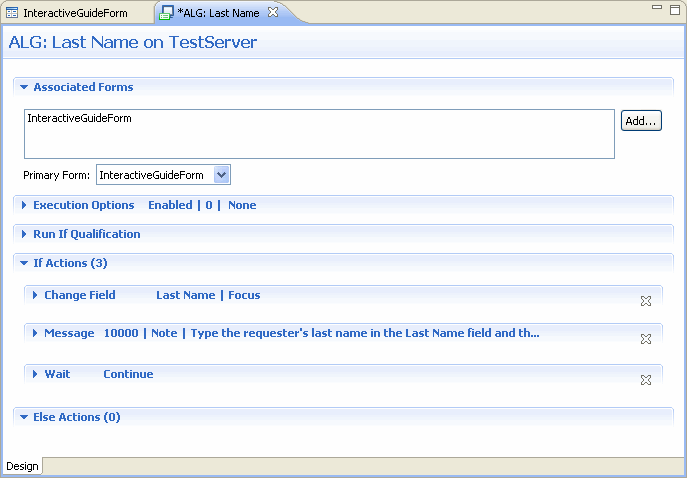
􀂄 Assign the Public group permissions for the active link.

􀂄 Save the active link.

**TIP**

After creating the first active link, use Save As to create the others, then edit the actions and active link name appropriately.

**Figure 5-6: Example active link for the interactive guide**



3 Create an active link guide (for example, **Incident Guide**):

􀂄 Associate it to the **Interactive Guide Form**.

**TIP**

You can create an entry point guide by clicking the Entry Point check and choosing an active link that opens a window. For more information, see the Form and Application Objects Guide, “Creating form entry points,” page 439.

􀂄 In the Active Links tab of the active link guide, add the four active links you just created, in the following order:

􀂄 **ALG:Last Name**

􀂄 **ALG: First Name**

􀂄 **ALG:Email Address**

􀂄 **ALG:Department**

􀂄 Grant the Public group Hidden access permissions to the active link guide.

􀂄 Save the active link guide.

4 Create another active link that calls the active link guide (for example, **Call Incidents Guide**):

􀂄 Associate it to the **Interactive Guide Form**.

􀂄 Select the Button/Menu Field Execution Condition, and specify the **Guide Me** button.

􀂄 Add a Call Guide action, and specify the **Incident Guide**.

􀂄 Grant public permissions to the active link.

􀂄 Save the active link.

5 Test the active link guide:

a In BMC Remedy User, open the **Interactive Guide Form** in New mode.

b Click the **Guide Me** button.

The focus should move to the first field and the message should appear prompting you to complete the field.

**\*\* Filter guides**

Filter guides are used to create reusable components of filter workflow by adding computational subroutines within filter processing. This provides for more sophisticated workflow solutions and allows easier reuse of functionality between

forms.

A filter guide is a list of filters that perform a task on a particular form. You create the filters before you insert them into a guide. Filters that execute in the context of guides are not triggered by the same Execute On conditions that trigger a filter during normal workflow. Filters are evaluated in their order within the guide, subject to redirection by the way of the Go to Guide Label action. If a filter used in a guide does have an execution condition, its execution condition is ignored.

You use three filter actions to implement filter guides:

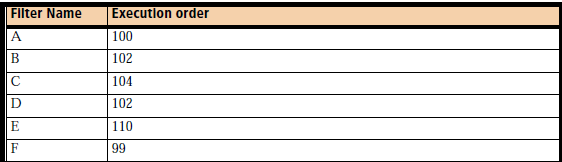
􀂄 **Call Guide**—Starts the filter guide. Like active link guides, the Call Guide action can be nested, calling yet another filter guide. The number of nested Call Guides cannot be deeper than the maximum depth of the filter action stack. For more information about the Call Guide action, see “Call Guide action” on page 76.

􀂄 **Exit Guide**—Terminates the filter guide. This action is ignored if it is not running inside a filter guide. For more information about the Exit Guide, see “Exit Guide action” on page 86.

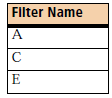
􀂄 **Go to Guide Label**—Redirects the flow of execution within a guide to a specific filter. This action is ignored if it is not running inside a guide. If the guide label is not found, this action is ignored as well. For more information about the Go to Guide Label action, see “Go to Guide Label action” on page 86.

Filter guides allow a developer to group a set of filters into a single unit of work (that is, a subroutine) and call only the filters referenced in sequence inside the guide without caring about the execution order of other filters. They also let developers call the filter guide under many different circumstances tied to multiple different forms. In essence, developers can create a piece of functionality that can be called as a unit of work, and not care about the execution order across all filters. In that way, developers can focus on what the filter guide as a unit of

work accomplishes. For example, take the following filters with their execution orders:



By design, each filter executes in its own execution order, based on which events trigger the filter action. By contrast, a filter guide could call these filters, in the following defined sequence:



This same filter guide can be triggered in different circumstances, at different times, from different forms.

Filter guides do not affect the “phases” of filter processing. A Set Fields action still executes in Phase 1, a Push Fields action in Phase 2, and so on. For more information about filter phases in AR System, see “Filter processing in the AR System server” on page 177.

You use essentially the same procedures to create filter guides as you do for active link guides. For more information, see “Creating guides” on page 140.

**Using a filter guide to loop through a table field**

Filter guides are useful for creating filter workflow that steps through the rows in a server-side table field. A server-side table field is a field in a supporting form that allows you to manipulate data on the server and then return the result to the client, rather working with the data directly in the client. This approach improves performance by reducing the amount of network traffic between the client and the

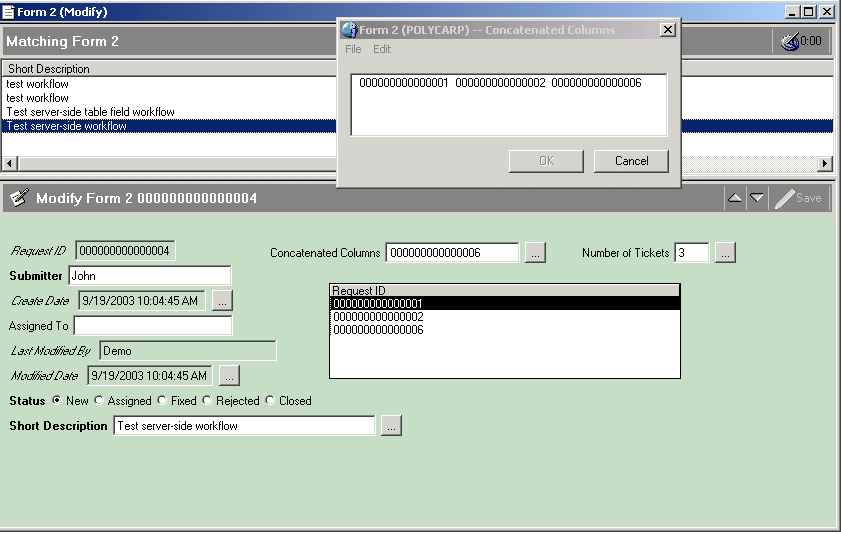
server. The filter guide workflow goes through every row in the table and then performs selected filter actions on particular rows or sets of qualifying rows. These qualifications can be built with reference to table column values. Because filters run on the server, you cannot use server-side table fields to highlight rows for display. You cannot select rows on server-side table fields, like you can on the client side. The user whose search initiates the server-side table loop must have at least hidden permissions to the table field.

**TIP**

For best performance, combine all operations on a server-side table field into a single filter guide. This avoids retrieving the data from the server more than once. This table looping functionality works only inside the filter guide. Filters function on only one AR System server. If a server-side table field points to a form on a remote server, the filter guide does not work. You can also perform calculations on the columns of the server-side table field, using the COLAVG, COLCOUNT, COLMAX, COLMIN, and COLSUM functions inside a filter guide. Use these functions just as you do with active link guides. For more information about table fields and functions, see the Form and Application Objects Guide.

**Server-side table field example**

The following procedure constructs a simple loop that searches through all the rows of a server-side table field on a helper form (Form1) until it finds all the tickets with a specific user’s name on them. The workflow then uses values from a column in the table field to set fields on a second form (Form2). The first Set Fields action concatenates all the returned values of the table field column into a character field; the second action uses the COLCOUNT function to display the number of columns returned.



You can enhance this example by including additional in-line workflow, for example, notifying a user about the number of rows returned.

**NOTE**

This procedure assumes you already know how to create forms, fields, filter workflow, and filter guides.

􀀀 **To create a filter guide that loops through a server-side table field**

1 Create a form (for example, **Form1**).

2 Create another form (for example, **Form2**):

􀂄 Add an integer field to display the number of tickets (for example, **Tickets**).

􀂄 Add a character field to display all the entry IDs of the columns returned (for example, **Concatenated Columns**). Align these fields side-by-side so that the results show all the tickets for a specific user.

􀂄 Add a table field:

􀂄 Select Form1 as the source form.

􀂄 Enter the qualification **'Submitter' = $Submitter$**

Using this qualification in the table field allows the database to select the entries that match the $Submitter$ field. This can improve performance because neither the AR System server nor the client needs to process the other entries in the form.

􀂄 Add the Request ID field (becomes **Column**)

􀂄 Add the Submitter field (becomes **Column2**)

􀂄 (Optional) Hide the table field.

Because you use server-side table fields exclusively for computation, you should hide them on forms that users interact with.

3 Create a filter (**Form2 Set Field**):

􀂄 Associate the filter with **Form2.**

􀂄 Add a Set Fields action:

􀂄 In the mapping table, map the field **Concatenated Columns** to the value

**($Concatenated Columns$ + " ") + $Column$**

The workflow concatenates all the returned columns into the Concatenated Columns field and separates them with a space. $Column$ references the table column that contains the request IDs.

􀂄 Add another Set Fields action, and map **Tickets** to **COLCOUNT($Column$)**

The action uses the COLCOUNT function to fill the **Tickets** field with the value of how many tickets this user has created.

􀂄 Save the filter

4 Create a filter guide (**Form2 Filter Guide**):

􀂄 Associate the filter guide with **Form2.**

􀂄 In the Filters and Labels panel, add **Form2 Set Field** to the guide.

􀂄 Save the filter guide.

5 Create a filter (**Form2 Call Guide**):

􀂄 Associate the filter with **Form2.**

􀂄 Select the Execution Option **Submit**.

This filter executes when you create tickets in Form2 and then display them.

􀂄 Add a **Call Guide** action and enter **Form2 Filter Guide** as the guide name.

􀂄 Set Table Loop to All Rows, and then select the table field on **Form2**.

This action activates the guide and causes it to loop through the rows in the table field.

6 To test the guide:

􀂄 Log in to BMC Remedy User and open **Form1** in New mode.

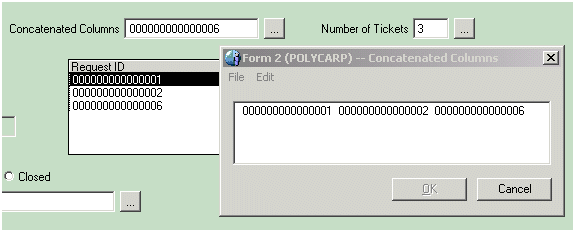
􀂄 Create several tickets, including two or three with **John** as the value of the Submitter field.

􀂄 Open **Form2** in New mode and create a ticket for John.

􀂄 Perform a search for John’s tickets, then click the request in the Results List.

The filter guide is triggered and loops through all the rows in the table field until it finds the rows with **John** as the value of the Submitter field. The workflow then completes the other fields in the form with the number of John’s tickets created in Form1. In addition, the concatenated entry IDs from the returned entries appear in the Concatenated Columns field.

**Figure 5-8: Tickets returned and columns displayed**



You can use similar functionality to loop through hundreds or even thousands of requests in the database, bypassing both the server and the client. By contrast, you could use client-side processing to calculate this information, but doing so increases network traffic between the client and the server and can impact performance.

'Name' = $First Name$ + $Middle Initial$ + $Last Name$

$TIMESTAMP$ - $CREATE-DATE$

$FIRST NAME$ + " " + $LAST NAME$

"hostname = " + $PROCESS$ hostname

