KINETIC

REQUEST

How to Build Parent and Child Service Requests

December 2012

KINETIC DATA

bmcsoftware
TECHNOLOGY ALLIANCE

235 EAST SIXTH STREET, SUITE 400B
ST. PAUL, MN 55101
WWW.KINETICDATA.COM

Kinetic Data, Inc, a BMC Software® Solutions Partner.

Remedy, a BMC Software Company

Contents

Chapter 1 OVERVIEW

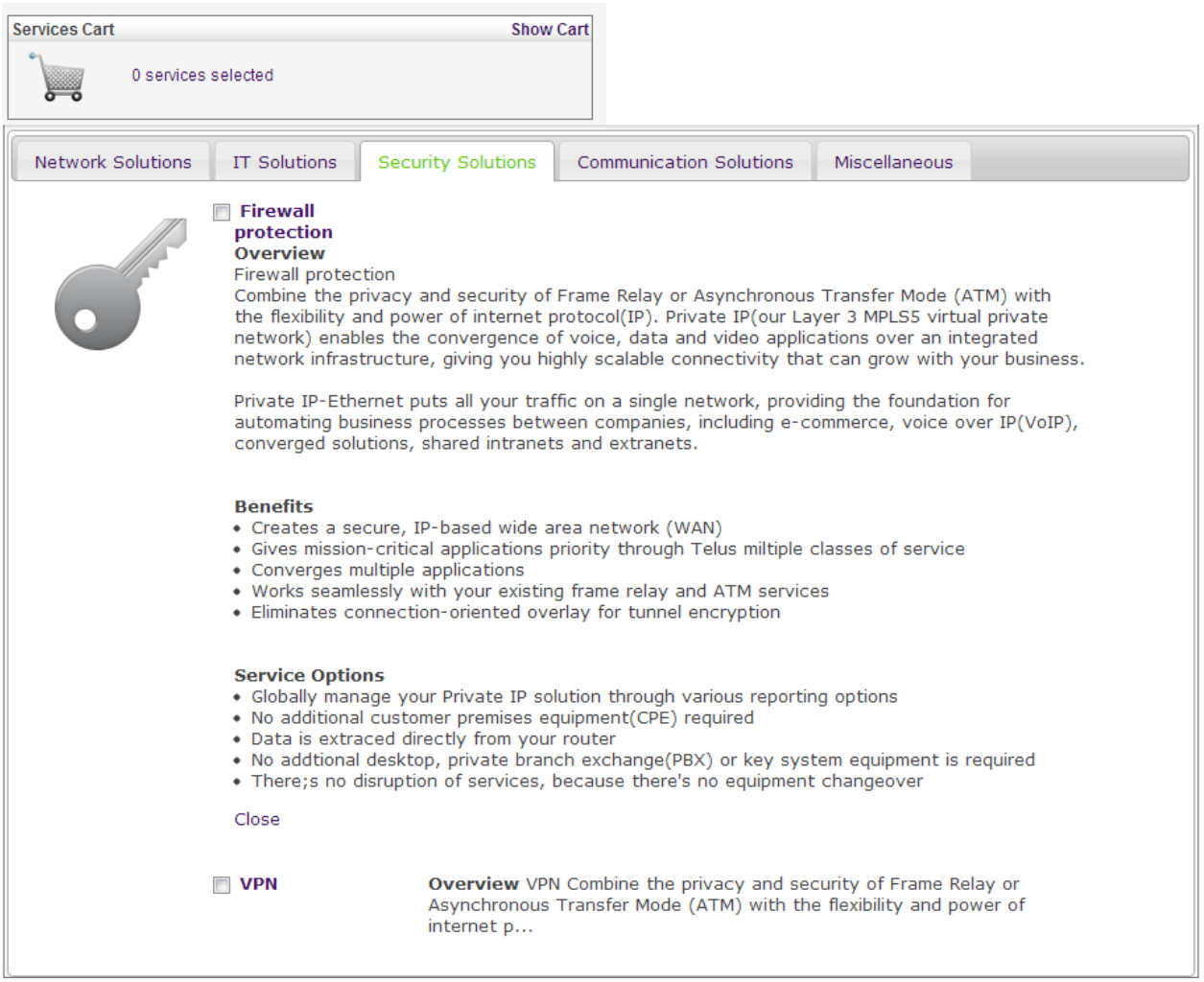
There are situations where you may want to fill out multiple requests at once, but same requests might not be chosen each time, ex. On-Boarding or Off-Boarding. This can be done by providing a list of the possible “child” requests and embedding the desired children in the “parent” request so that the user fills it all out together. This document covers the certain pages, elements, events, scripts and css necessary to accomplish this.

Chapter 2 PARENT SERVICE ITEM

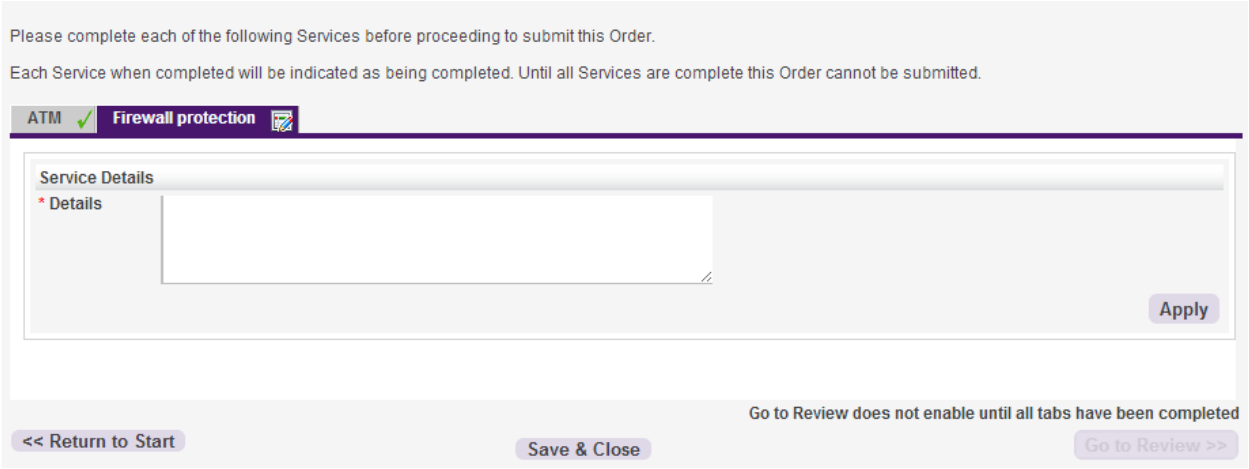
COMPONENTS

A Parent item is comprised of certain necessary components.

- Questions, generally minimally requested by and requested for
- A cart and tabs where the user selects which services they wish to order



- A Services Forms page that displays the children within iframes within the parent, so they can be filled out as part of the parent request.



Note that the parent questions can be on the same page as the cart, but the tabs containing the children (the System Forms page) should be on the next page.

- There must also be an attribute created for use by this parent record. It should be a Character Type and can be named (Attribute Type) whatever would be appropriate for this parent record. This is the attribute that will be added to the Child records. This will tell the parent record which items to load as possible children on the cart page.

CART PAGE DETAILS

Callbacks

- TabbedServices.jsp:: Creates the tabs that hold the service item list for items with the attribute specifically created for use with this parent with selection box and short and long description (long description from the text of the attribute), categorized (tabs named) based on "Type" of the service items. The description can be shortened into the short description by grabbing the first however many characters, or, as in this case, everything before the first
 in the description.
- TabbedServicesReview.jsp:: Creates the tabs that hold the service item list for items with the attribute specifically created for use with this parent with a disabled selection box and short and long description (long description from the text of the attribute), categorized (tabs named) based on "Type" of the service items. The description can be shortened into the short description by grabbing the first however many characters, or, as in this case, everything before the first
 in the description. If an item is in progress it is automatically selected and cannot be un-selected. Items that are in progress can also be cancelled unless they are required or already cancelled/cancelling, then they display the status of the item instead. Items are required if the "Service Item Description" field of the item contains the text "required".

The TabbedServices.jsp callback is used to construct the cart's tabbed pages if the item is NOT a review request. TabbedServicesReview.jsp should be used to construct the cart's tabbed pages if the items in in review request and you want the user to be able to see the status of and/or cancel individual portions of the request. If this ability to cancel children is not required, you may opt not to display the cart page at all on review of the item.

Resources

- tabbedservices.js:: Contains the functions that toggle the services into and out of the cart (both on click and on page load), the functions that toggle the long description on/off, and the function that saves the selected items into a question for future use.
- tabbedservices.css:: Contains the CSS to make the item descriptions (short and long) display. Note that to use the images, you much update the IDs to match the names of your categories (Types) with the spaces removed. Also note that you will need to updated the image paths of these files for your theme name.
- /webapps/kinetic/resources/jquery/jquery-ui-1.8.12.custom/css/thr/jquery-ui-1.8.12.custom.css:: Contains the CSS to make the tabbed pages display correctly. Note this file may be for a different version of jquery depending on what your installation is using. If your installation is not currently using anything, you may add these files into the specified directories. Or your theme might already have an equivalent file included for custom jquery css. That will likely work as well. Note that these are not within the theme directory, but within a core resources directory.
- primo_icons/*:: These images are available for use on your tabbed services pages and can be referenced in your tabbedservices.css file for use/display.

Events

Events are listed with the name of the event, a description, then an example of the event exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*. Note that only events used to display/process the cart and tabs are listed here. You may need other events for your service item, ex. events for "save & exit" function. Please note that the order of events is important.

- Display Service Items:: This on-load, custom action does a simple data request for the attribute type created for this parent (in this example On-boarding) and uses either one of the tabbed Services callbacks to display the available services.

```
event "Display Service Items",
      :custom_action,
      :load do
      custom_code "if (clientManager.submitType !='ReviewRequest') {\n  if
(nameKnown  == 'No')\n  {\n
tabServiceHelper.loadPreOrderTabbedServices(clientAction.actionId);\n  } else {\n
tabServiceHelper.loadTabbedServices(clientAction.actionId);\n  }\n} else {\n
tabServiceHelper.loadReviewTabbedServices(clientAction.actionId);\n}"
      data_request "KS_SRV_SurveyTemplateAttrInstance_join",
        :name => "Load New Services",
```

```
        :fields_available =>
"instanceId,Survey_Template_Name,Character_Value,Integer_Value,Attribute Type
Name,Priority,Type,Survey_Description",
        :sort_fields => "Priority,Survey_Template_Name",
        :max_entries => "500",
        :sort_order => "Ascending",
        :qualification => "'Attribute Type Name'=\"On-boarding\" AND  'Status'<=
\"Active\""
    end
```

- Remove for review request:: This event removes sections for the review request display.

```
event "Remove for review request",
      :insert_remove,
      :load,
      :fire_if => "clientManager.submitType ==\"ReviewRequest\" " do
      target "Bottom of Page",
        :section,
        :remove
      target "Cart Items",
        :section,
        :remove
      target "Service Cart",
        :section,
        :remove
      target "System Access",
        :section,
        :remove
    end
```

- set required items:: This event will have to be customized for whatever your required items are, it automatically checks the items that are required before submission if the user has not already selected them

```
event "set required items",
      :custom_action,
      :before_submit,
      :fire_if => "clientManager.submitType !=\"ReviewRequest\" " do
      custom_code " \nif (document.pageQuestionsForm.NetworkAccess.checked==false)
{\ndocument.pageQuestionsForm.NetworkAccess.checked=true;\ntabServiceHelper.toggleSe
lectedService(document.pageQuestionsForm.NetworkAccess,
'KRd8c9d5143ff39b73ee7c78916a68d424f');\n}"
    end
```

- valid Selection:: This before submit event checks that at least one selection has been done.

```
event "valid Selection",
      :custom_action,
      :before_submit,
      :fire_if => "!tabServiceHelper.validSelection()" do
      custom_code "alert('Please ensure that you have selected at least one System
option');\nfalse;"
    end
```

- save Selection:: This before submit event saves the selected items into an answer for future use.

```
event "save Selection",
      :custom_action,
      :before_submit,
      :fire_if => "tabServiceHelper.validSelection()" do
```

```
    custom_code "tabServiceHelper.saveSelection();"
end
```

Elements

Elements are listed with the name of the element, a description, then an example of the element exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*. Note that only elements used to display/process the cart and tabs are listed here. You may need other elements for your service item, ex. elements for "save & exit" function, or you may want additional elements for display purposes, ex. section headers.

- Section: Cart Items:: This section contains the three text elements for the cart
  - txt_CartItems:: this is a text element that displays the link to expand the cart details
  - cartSelectedItems:: this is a text element that displays the cart icon and the number of items selected
  - cart container:: this is a text element that contains, when toggled on, the full details of the cart.

```
section  "Cart Items"
    text "txt_CartItems", "Services Cart <span style=\"float:right;\"><a
class='tabbedLink' href=\"javascript:tabServiceHelper.toggleServicesCart();
void(0);\">Show Cart</a></span>"
    text "cartSelectedItems", "<div id='cartSelectedItems'>\n<div id='image-
cartItemList'></div>\n<div id='servicesSelectedLinkContainer'><a
id='servicesSelectedLink' href='javascript:tabServiceHelper.toggleServicesCart();
void(0);'>0 services selected</a></div>\n</div>"
    text "cart container", "<div id='serviceCart' class='nodisplay' style='border:
1px solid #7F7F7F;float: left;width: 99.5%;'>\n      <div class='dynamicText
headingLabel' style='margin-bottom:5px'>Full Services Cart\n      <span
style='float:right;'><a class='tabbedLink'
href=\"javascript:tabServiceHelper.toggleServicesCart(); void(0);\">Hide Full
Cart</a></span>\n      </div>\n            <!--<div id='image-selected'
class=\"imagesContainer\"></div>-->\n            <div id='serviceItems_selected'
class=\"servicesContainer\">\n           </div>\n</div>"
```

- Section: Item Options:: this section contains the tabs for the services.
  - Forms Table:: this text element contains a placeholder div
  - Tabbed Services:: this text element contains a placeholder div used by the callback to display the services available for selection

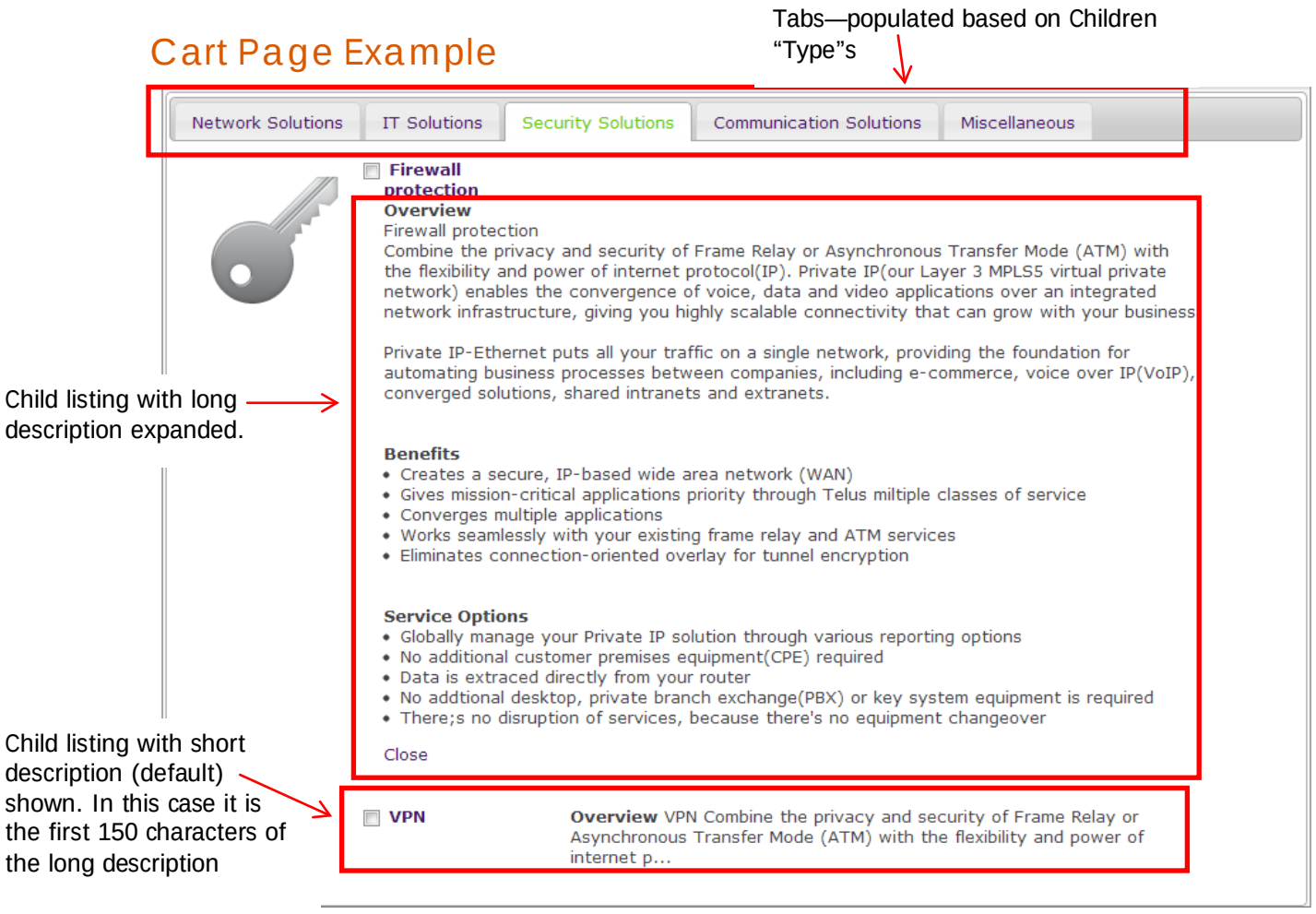
```
section  "Item Options"
    text "Forms Table", "<div id='formTable'> </div>"
    text "Tabbed Services", "<div id='tabbedServicesContainer' style=\"min
height:300px;\" > </div>"
```

- Section: System Access:: this is a removed (hidden) section that contains the questions necessary for processing the cart.
  - Systems:: this question is used for storing the selected services.
  - Multi Form Indicator:: this question is used to specify that this item uses Multi-Form

```
section  "System Access",
```

```
  :removed
question "Systems", "Systems", :free_text,
  :size => "20",
  :rows => "1",
  :field_map_number => "84"
question "Preorder", "Preorder", :free_text,
  :size => "20",
  :rows => "1",
  :field_map_number => "112"
text "On-Boarding Header Text", "Services Requested for On-Boarding",
  :style_class => " sectionHeader "
question "Multi Form Indicator", "Multi Form Indicator", :free_text,
  :default_answer => "MULTI-FORM",
  :size => "20",
  :rows => "1",
  :field_map_number => "85"
```

Cart Page Example



SYSTEM FORMS PAGE

Callbacks

- loadSystemsRequired.jsp:: This callback creates a table of rows of instance IDs and Template Names for the items chosen on the cart page.
- loadSystemsStatus.jsp:: This callback creates a table of rows of Name, Child Status, CSRV, Validation Status, Request Status for the children records created for the systems specified in the cart page. This allows us to reload the same children each time a user returns to this page.

Resources

- multiQuest.js:: This javascript contains the functions that build the tabs, fill in the iframes within the tabs, store/maintain the status of the tabs, and handle the availability of the submit button.
- multiQuest.css:: This CSS file contains styles for displaying the tabs of the service items, including the references for the complete/incomplete images.
- tick.gif:: The complete tab image in this example
- cross.gif:: The incomplete tab image in this example
- subform.css:: This CSS file contains the styles for the sub-form (the child displayed inside the parent). This is primarily controlling margin and width so items will display correctly.

Events

Events are listed with the name of the event, a description, then an example of the event exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*. Note that only events used to display/process the cart and tabs are listed here. You may need other events for your service item, ex. events for "save & exit" function. Please note that the order of events is important.

ON PAGE LOAD

- Clear selection:: This on load set fields internal event makes sure we are looking at a clean slate to start off with

```
event "Clear selection",
      :set_fields_internal,
      :load,
      :fire_if => "clientManager.submitType !=\"ReviewRequest\"" do
      field_map "Systems Selected", "$\\NULL$"
    end
```

- load Tabs In Review Mode:: This on load custom event builds the tabs for a review request.

```
event "load Tabs In Review Mode",
      :custom_action,
      :load,
      :fire_if => "clientManager.submitType ==\"ReviewRequest\" " do
      custom_code "\n    buildTabs='Yes';\nvar obj =
KD.utils.Util.getQuestionInput('Systems
Selected');\nKD.utils.Action._fireChange(obj);"
    end
```

- Remove for review request:: This on load insert-remove event hides items for a review request

```
event "Remove for review request",
      :insert_remove,
      :load,
      :fire_if => "clientManager.submitType ==\"ReviewRequest\" " do
      target "Bottom of Page",
        :section,
        :remove
      target "Go to Review Message",
        :text,
        :remove
      target "Info",
        :text,
        :remove
    end
```

- Trigger Load Systems Selected:: This on load custom event builds the tabs if this is not a review request. In this case you want to be sure you have the correct systems selected from the previous page.

```
event "Trigger Load Systems Selected",
      :custom_action,
      :load,
      :fire_if => "clientManager.submitType !=\"ReviewRequest\"" do
      custom_code "buildTabs='Yes';\nKD.utils.Action.setQuestionValue('Systems
Selected', 'LOAD');\n\nvar obj = KD.utils.Util.getQuestionInput('Systems
Selected');\nKD.utils.Action._fireChange(obj);\n\n"
    end
```

On Elements

A Number of the key events on this page on not on page load, but on an existing element. The following events are attached to the Question "Systems Selected". Please note that the order of events is important.

- Load Systems Selected:: This on change set fields external event, if the question value is LOAD, gets the answer from the previous page's "Systems Selected" question and places it into this question, "Systems Selected". If nothing is returned the question will get set to null.

```
event "Load Systems Selected",
        :set_fields_external,
        :change,
        :null_on_no_match,
        :fire_if => "obj.value == \"LOAD\"" do
        data_request "KS_SRV_SurveyAnswer",
```

```
          :sort_fields => "",
          :max_entries => "1",
          :sort_order => "Ascending",
          :qualification => "'CustomerSurveyInstanceID' = \"<FLD>Submission Instance
ID;clientManager.customerSurveyId;BASE</FLD>\"  AND  'QuestionInstanceID' =
\"KS0050569A648CD-tqUAmxR5KwNZUm\""
        field_map "Systems Selected", "<FLD>FullAnswer</FLD> ",
          :fire_change_event,
          :visible_in_table => "No"
      end
```

- Re-Run load Selected Systems:: This on change custom event triggers the LOAD to attempt again because the desired value was not found. These first two events basically loop until the answer is returned.

```
event "Re-Run load Selected Systems",
        :custom_action,
        :change,
        :fire_if => "obj.value== \"\"" do
        custom_code "\nbuildTabs='Yes';\nKD.utils.Action.setQuestionValue('Systems
Selected', 'LOAD');\n\nvar obj = KD.utils.Util.getQuestionInput('Systems
Selected');\nKD.utils.Action._fireChange(obj);"
      end
```

- Load Systems Required:: This on change custom action, now that a value has been set into this field that it can work with, performs a simple data request and uses the callback loadSystemsRequired.jsp to populate a table in a removed (hidden) text element. The values in this table are used by the pages JS (see resources section above) to create the working registry/hash and populate the children service item tabs.

```
event "Load Systems Required",
        :custom_action,
        :change,
        :fire_if => "(obj.value!= \"LOAD\" && buildTabs==\"Yes\") || (obj.value!=
\"LOAD\" && clientManager.submitType ==\"ReviewRequest\" )" do
        custom_code "param1 = '<FLD>Systems
Selected;KS0050569A648Cee1qUA1IOCKwN6om;ANSWER</FLD>';\nparam2 = '<FLD>Category
ID;clientManager.categoryId;BASE</FLD>';\nvar params = [param1, param2];\nvar
rqtParamString = KD.utils.Action._buildParamString(params);\n\nvar connection=new
KD.utils.Callback(processSystemsRequired,KD.utils.Action._addInnerHTML,['Systems_Req
uired'], true); \n\nvar runSynchronous=true;\nif
(clientManager.submitType=='ReviewRequest') {\nrunSynchronous =
true;\n}\nKD.utils.Action.makeAsyncRequest('loadSystemsRequired',
clientAction.actionId, connection, rqtParamString,
'../../themes/frb/packages/base/interface/callbacks/loadSystemsRequired.jsp', true,
runSynchronous);\n"
        data_request "KS_SRV_SurveyTemplate",
          :name => "loadSystemsRequired",
          :fields_available => "Survey_Template_Name,instanceId",
          :sort_fields => "Survey_Template_Name",
          :max_entries => "500",
          :sort_order => "Ascending",
          :qualification => "\"<FLD>Systems
Selected;KS0050569A648Cee1qUA1IOCKwN6om;ANSWER</FLD>\" LIKE ( \"%\" + 'instanceId' +
\"%\" )"
      end
```

- Load Systems Status:: This on change custom action, now that a value has been set into this field that it can work with, performs a simple data request and uses the callback loadSystemsStatus.jsp to populate a table in a removed (hidden) text element. The values in this table are used by the pages JS (see resources section above) to create the working registry/hash and populate the children service item tabs.

```
event "Load Systems Status",
        :custom_action,
        :change,
        :fire_if => "(obj.value!= \"LOAD\" && buildTabs==\"Yes\") || (obj.value!=
\"LOAD\" && clientManager.submitType ==\"ReviewRequest\" )" do
        custom_code "param1 = '<FLD>Submission Instance
ID;clientManager.customerSurveyId;BASE</FLD>';\nparam2 = '<FLD>Systems
Selected;KS0050569A648Cee1qUA1IOCKwN6om;ANSWER</FLD>';\nvar params =
[param1,param2];\nvar rqtParamString =
KD.utils.Action._buildParamString(params);\n\nvar connection=new
KD.utils.Callback(processSystemsStatus,KD.utils.Action._addInnerHTML,['Systems_Statu
s'], true); \nvar runSynchronous=true;\nif
(clientManager.submitType=='ReviewRequest') {\nrunSynchronous =
true;\n}\nKD.utils.Action.makeAsyncRequest('loadSystemsStatus',
clientAction.actionId, connection, rqtParamString,
'../../themes/frb/packages/base/interface/callbacks/loadSystemsStatus.jsp', true,
runSynchronous);"
        data_request "KS_SRV_CustomerSurveyResults_join",
          :name => "Load Systems Status",
          :fields_available =>
"CustomerSurveyInstanceId,AnswerViewer36,AnswerViewer37,AnswerViewer39,AnswerViewer4
1,CustomerSurveyStatus,ValidationStatus",
          :sort_fields => "",
          :max_entries => "500",
          :sort_order => "Ascending",
          :qualification => "'AnswerViewer37' =  \"<FLD>Submission Instance
ID;clientManager.customerSurveyId;BASE</FLD>\" AND  \"<FLD>Systems
Selected;KS0050569A648Cee1qUA1IOCKwN6om;ANSWER</FLD>\" LIKE ( \"%\" +
'surveyTemplateInstanceID' + \"%\" )"
      end
```

- reset frame height:: This custom action resizes the frame for a review request.

```
event "reset frame height",
        :custom_action,
        :change,
        :fire_if => "clientManager.submitType ==\"ReviewRequest\" " do
        custom_code "setReviewHeight();"
      end
```

Elements

Elements are listed with the name of the element, a description, then an example of the element exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*. Note that only elements used to display/process the cart and tabs are listed here. You may need other elements for your service item, ex. elements for "save & exit" function, or you may want additional elements for display purposes, ex. section headers.

Note that this page has the name of the submit button modified to: Go to Review >>

- Section: Hidden Details:: This (removed) section contains the hidden details that make this page function, both text elements and questions.
    - Systems Required:: This text element contains a div tag that is used to store the systems required table. Note that if this table is not being correctly built, the correct services/children won't be embedded into this page as tabs. If this is not functioning properly, look to the events and the simple data requests within them as your first line of troubleshooting.
    - Systems Status:: This text element contains a div tag that is used to store the systems status table. Note that if this table is not being built correctly the tabs won't contain the correct service item (that was previously begun, if there was one) or the status of the tab won't be correct. If this is not functioning properly, look to the events and the simple data requests within them as your first line of troubleshooting.
    - Systems Selected:: This question element contains the listing of systems that were selected on the previous page. It contains the events that create the tables in the above text elements.
    - Systems Selected Names:: This question contains the names of the services selected, for use in populating the tabs
    - Form IDs JSON:: This question is used to store the data from the working registry/hash so the status of each child is loaded correctly upon returning to this page.

```
section  "Hidden Details",
      :removed
    text "Systems Required", "<div id='Systems_Required'> </div>"
    text "Systems Status", "<div id='Systems_Status'> </div>"
    question "Systems Selected", "Systems Selected", :free_text,
      :size => "20",
      :rows => "1",
      :field_map_number => "90" do
```

Question events covered in previous section. Please see that section for those details. The events would be inserted into the KURL code here.

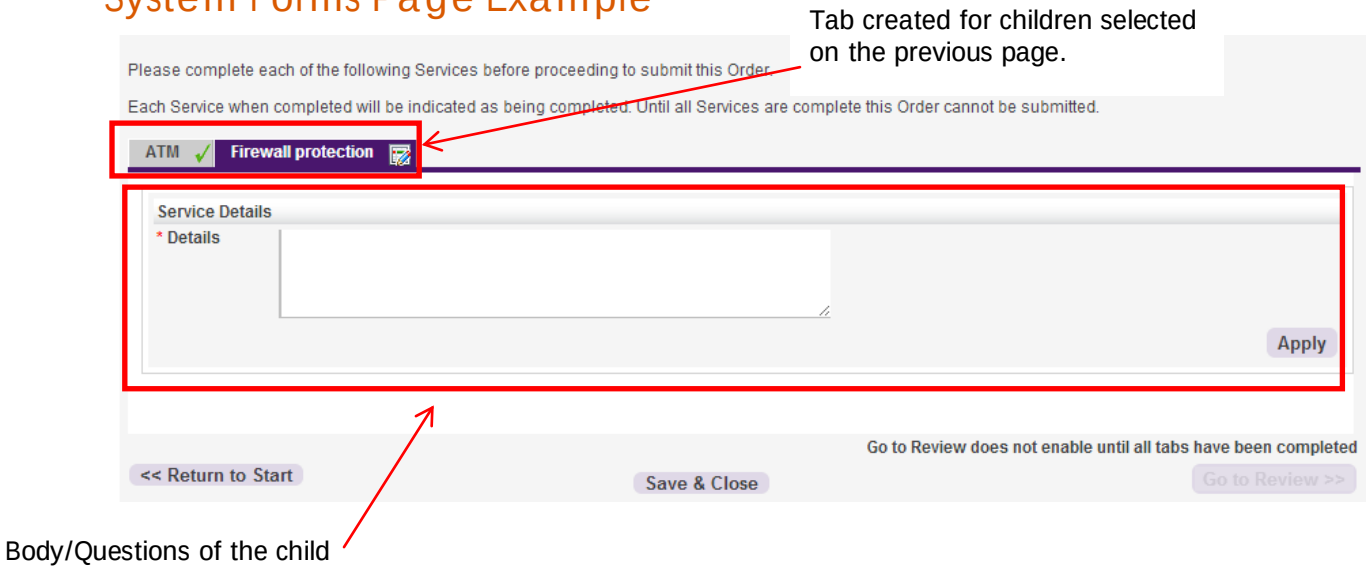
```
    end
    question "Systems Selected Names", "Systems Selected Names", :free_text,
      :size => "20",
      :rows => "1",
      :field_map_number => "91"
    question "Form IDs JSON", "Form IDs JSON", :free_text,
      :size => "20",
      :rows => "1",
      :field_map_number => "92"
```

- Section: Tab Section:: This section stores the tabs for the children items.
    - Info:: This text element contains some instruction as to how the tabs work
    - Tab Holder:: This text element contains the div tags for the tabs and the pages for those tabs where the children appear

- Go to Review Message:: This text element is a reminder about the submit button not functioning until the tabs have been filled out.

```
section  "Tab Section"
    text "Info", "<p>Please complete each of the following Services before
proceeding to submit this Order. </p>\n<p>\nEach Service when completed will be
indicated as being completed. Until all displayed Services are complete this Order
cannot be submitted.</p>"
    text "Tab Holder", "<ul id=\"systemTabs\">\n</ul>\n<div  id=\"systemTabPages\"
style=\"min-height:900px;\">\n</div>"
    text "Go to Review Message", "<b>Go to Review does not enable until all tabs
have been completed</b>"
```

System Forms Page Example



Chapter 3 CHILD SERVICE ITEM

Each child also has necessary elements and code. These elements vary depending on factors such as whether the child has multiple pages and whether the child also needs to be available as a stand alone. The case covered here is the multiple page stand alone, the most complex case. Where appropriate, it is indicated if the components are not necessary in the other cases.

COMPONENTS

- Each child must have an Attribute instance for the attribute created for this parent. The existance of this attribute in the child is what tells the parent that this record is a child. The text entered for the attribute is what determines the description used on the cart page in the parent record.
- The "Type" field of the child is what determines what tab the child appears on in the cart page of the parent.
- The "Service Item Description" field of the child determines whether the child is required (cannot be cancelled) or not. The item is required if this field contains the word "required" per the code provided in the parent section.
- A child must have a series of events on each page to function appropriately.
- A child must have certain questions on the first page to function appropriately, and if the child is multi-paged, there are possibly other necessary elements.
- A child must have a second "confirmation" page.
- The Submit Type field (field 700088475) will need to be added to your data set, if not already present.
- The child must use a different jsp for the Display Page (JSP) (from the advanced tab of the service item) that contains certain additional functions/features.

Note that the child depends on being called properly from the parent. The parent should be passing it's csrv and the system form name it's calling on the URL when it creates the child. If this data isn't present, the child will not process correctly. This is done inside the loadPage function in multiQuest.js if you are troubleshooting that item.

FORMS/RESOURCES

- subFormPage.jsp:: This jsp is used as the display page for a child item. This contains the necessary functions and features to process as a child. The attached version contains header and footer div tags. If your children requests will not stand alone (also be called individually, outside the parent), then you can remove these tags entirely and the events that correspond with them (see event section below).
- subFormPageReview.jsp:: This jsp is used as the review page for a child item. This contains the necessary functions and features to process as a child. The attached version contains

header and footer div tags. If your children requests will not stand alone (also be called individually, outside the parent), then you can remove these tags entirely and the events that correspond with them (see event section below).

- subform.css:: This CSS file contains the styles that ensure the child displays properly within the parent.

ELEMENTS

Each child must pull certain information from the calling parent. This information is put into questions on the first page of the child item. The information included here is for questions that are absolutely required for the child to function. You'll likely need to pull additional information from the parent, including requested by/requested for information.

Elements are listed with the name of the element, a description, then an example of the element exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*.

- Section: Tabbed Details:: this removed (hidden) section is used to store data pulled over from the parent that isn't necessary for the user to view. None of these fields contain values if this request is used outside of the parent (as a stand alone).
    - System Form:: This question contains the name/title of the child. This is used within events on the first page and passed on to other pages for use in events on those pages.
    - Primary Request ID:: This question is used to store the CSRV/Instance ID of the parent Service Item that generated this child. This question is mapped into the Originating ID field. Note: If you maintain a dataset other than the SYSTEM_DEFAULTS set the field you map to may have a different name. The Parent ID is also used by other pages to control which events fire.
    - ParentKSR:: This question is used to store the KSR number/request ID of the parent Service Item that generated this child. This question is mapped to the Originating ID-Display field. NOTE: If you maintain a dataset other than the SYSTEM_DEFAULTS set the field you map to may have a different name.
    - Submit Type:: This is set to Child if this is being called as a child and left blank if the child is stand alone. This can be defaulted to Child if the item is not available outside of the parent (as a stand alone). This question is mapped to the Submit Type field. **NOTE**: This is not part of the dataset. You will need to add this to your dataset.
    - subFormMarker:: This question contains SUBFORM if this is a child
    - _Tab_Mode_MarkAsComplete:: This question contains COMPLETED if the request is a child.

```
section  "Tabbed Details",
      :removed
    question "systemForm", "systemForm", :free_text,
      :size => "20",
      :rows => "1",
```

```
  :field_map_number => "36"
question "primaryRequestID", "primaryRequestID", :free_text,
  :answer_mapping => "Originating ID",
  :size => "20",
  :rows => "1",
  :field_map_number => "37"
question "ParentKSR", "ParentKSR", :free_text,
  :answer_mapping => "Original ID-Display",
  :size => "20",
  :rows => "1",
  :field_map_number => "43"
question "Submit Type", "Submit Type", :free_text,
  :answer_mapping => "Submit Type",
  :size => "20",
  :rows => "1",
  :field_map_number => "44"
question "subFormMarker", "subFormMarker", :free_text,
  :size => "20",
  :rows => "1",
  :field_map_number => "39"
question "_Tab_Mode_MarkAsComplete", "_Tab_Mode_MarkAsComplete", :free_text,
  :size => "20",
  :rows => "1",
  :field_map_number => "41"
```

You may be in a situation where all of the information on the first page of the child is information garnered from the parent, ex. Requested by and Requested for information. If that is the case, there will be additional events need to automatically submit the first page. This is not, however, instantaneous and you will likely want to hide the first page sections and replace them with a "page loading" type message so as not to confuse your users.

```
  section  "Child Loading Section",
      :removed
    text "Page Loading Message", "Page Loading <img id=\"ajax_searchReqFor\"
src=\"themes/frb/common/resources/images/ajax-loader.gif\" alt=\"searching...\" />"
```

There must be a question to store the parent csrv on each page—even the review page if it is used when the request is a child. In Kinetic Request 5.1, this can be defaulted to the value from the first page. In previous versions, this can be done with a set fields external event. A version of this event is included in the following events section.

There must be an additional page added to your child that is only used in the case where the request is a child.

- Page: System Form Confirmation:: This page comes right before the confirmation page and serves as the confirmation page of the child. The child never actually makes it to the submission page, since it is submitted by the submission of the parent (in the parent's Complete tree).
    - Section: Tabbed Details
        - Javascript:: this javascript loads certain values for use in questions on this page (note that this is only necessary in Kinetic Request 5.0.3 and will be replaced with defaults to previous page questions in Kinetic Request 5.1)
        - _Tab_Mode_Csrv:: This question stores the parent ID
        - _Tab_Mode_SystemForm_Conf:: This question stores the name of the child (used for registration in the parent)

- Info:: This is the “confirmation” text displayed to the user when they have “completed” the child within the parent.

```
section  "Tabbed Details"
    text "Javascript", "<script>\nvar tabModeCsrv =
\"<FLD>primaryRequestID;ANSWER</FLD>\";\nvar tabModSystemFormConf =
\"<FLD>systemForm;ANSWER</FLD>\";\n</script>"
    question "_Tab_Mode_Csrv", "_Tab_Mode_Csrv", :free_text,
      :removed,
      :size => "20",
      :rows => "1",
      :field_map_number => "52"
    question "_Tab_Mode_SystemForm_Conf", "_Tab_Mode_SystemForm_Conf", :free_text,
      :removed,
      :size => "20",
      :rows => "1",
      :field_map_number => "53"
    text "Info", "Your <FLD>Survey Template Name;700001000;BASE</FLD> Service
details are ready for submission.<br><br>The processing of this item will only begin
when your request is submitted.<br>"
  end
```

Note that if your child also functions as a stand alone item it will need to contain a Confirmation Page and branching logic on the appropriate previous page to go to the System Form Confirmation page if the request is a child and the Confirmation Page if the item is not a child. Included is an example of the branching rule for a page before the System Form Confirmation page that skips to the Confirmation Page if the item is a stand alone.

```
branching "Confirmation Page", "<FLD>wrk3 Parent ID;ANSWER</FLD> = \"\"  OR <FLD>wrk3
Parent ID;ANSWER</FLD> =  $\\NULL$",
      :description => "Go to confirmation for Stand Alone"
```

EVENTS

The number and detail of the events necessary for processing of a child depend on a number of factors, including whether the item also needs to be able to stand alone, how much data needs to be pulled from the parent, where that data is stored in the parent, etc. The events listed in this document are only those necessary for interaction between the child and the parent. Your child request may need additional events to process dates or hide/show fields, etc that are specific to your item or events to support such features as cancel or save & exit. Also, you may need to add qualifications to some events, possibly required fields or hide/show, so that they will only fire if the request is not a child.

Events are listed with the name of the event, a description, then an example of the event exported via KURL. For more details about KURL and it's formatting, see the *KURL User Guide* on the *Kinetic Community*.

PAGE 1

Your first page will likely contain the most events, because you will do the majority of pulling data from the parent into the child here.

- Set Primary ID:: This custom on load event, if there is a primaryCsrv in the URL, hides the header and footer sections and sets that value and the System Form name provided on the URL into questions.

```
event "Set Primary ID",
      :custom_action,
      :load,
      :fire_if => "primaryCsrv" do
      custom_code
"ReplaceContentInContainer('headertext',' ');\nReplaceContentInContainer('footertex
t',' ');\n\nKD.utils.Action.setQuestionValue('primaryRequestID',
primaryCsrv);\nKD.utils.Action.setQuestionValue('systemForm', systemForm);"
    end
```

- Hide/Remove for Child:: This event is used to hide all the existing elements and display a "Page Loading" message. This would only be necessary if all information on the first page of the request is provided via the parent. If there were some elements that were provided and some that weren't, you might want to hide those that were, but you would not display the "Page Loading" message.

```
event "Hide/Remove for Child",
      :insert_remove,
      :load,
      :fire_if => "(primaryCsrv || KD.utils.Action.getQuestionValue(\"Submit Type\") ==
\"Child\") && clientManager.submitType !=\"ReviewRequest\" " do
      target "Requested For Section",
        :section,
        :remove
      target "Manager Section",
        :section,
        :remove
      target "Manager Details Section",
        :section,
        :remove
```

```
      target "Child Loading Section",
        :section,
        :insert
      target "Bottom of Page",
        :section,
        :remove
      target :submit_button,
        :remove
    end
```

- Get info from parent:: This set fields external event sets a series of values from the parent. Which fields you need to write data into, which fields you need to fire change events on, and which fields they come from are all dependent on your set-up and attribute & answer mapping in your parent. You MUST set the ParentKSR to the Customer Survey ID. The rest of the field_maps in the example event are just an example.

```
event "Get all info from Parent",
      :set_fields_external,
      :load,
      :null_on_no_match,
      :fire_if => "KD.utils.Action.getQuestionValue(\"primaryRequestID\") != \"\" ||
primaryCsrv" do
      data_request "KS_SRV_CustomerSurveyResults_join",
        :sort_fields => "",
        :max_entries => "1",
        :sort_order => "Ascending",
        :qualification => "'CustomerSurveyInstanceId' =
\"<FLD>primaryRequestID;KS32e6dd259f907492accbd0dc0647g6412;ANSWER</FLD>\""
      field_map "ReqBy_Login ID", "<FLD>Attribute1</FLD> ",
        :visible_in_table => "Yes"
      field_map "ReqBy_First Name", "<FLD>Attribute2</FLD> ",
        :visible_in_table => "Yes"
      field_map "ReqBy_Last Name", "<FLD>Attribute3</FLD> ",
        :visible_in_table => "Yes"
      field_map "ReqFor_Login ID", "<FLD>Attribute4</FLD> ",
        :fire_change_event,
        :visible_in_table => "Yes"
      field_map "ReqFor_First Name", "<FLD>Attribute5</FLD> ",
        :visible_in_table => "Yes"
      field_map "ReqFor_Last Name", "<FLD>Attribute6</FLD> ",
        :visible_in_table => "Yes"
      field_map "ParentKSR", "<FLD>CustomerSurveyID</FLD> ",
        :visible_in_table => "Yes"
      field_map "Hire Login ID", "<FLD>AnswerViewer59</FLD> ",
        :visible_in_table => "Yes"
    end
```

- Tabbed Mode Deregister Form:: This on load custom action is just a clean-up. It ensures that there is no previous csrv for this request for this parent registered in the parent's registry/hash.

```
event "Tabbed Mode Deregister Form",
      :custom_action,
      :load,
      :fire_if => "KD.utils.Action.getQuestionValue(\"systemForm\") != \"\" &&
clientManager.submitType != \"ReviewRequest\"" do
      custom_code "var subFormMark =
KD.utils.Action.getQuestionValue('subFormMarker');\nif (subFormMark== '' || subFormMark
== null)
{\ncalldeRegisterFormSubmitted(KD.utils.Action.getQuestionValue('systemForm'));\n}\n\nRe
```

```
placeContentInContainer('headertext',' ');\nReplaceContentInContainer('footertext',
' ');"
    end
```

- Tabbed Mode set Defaults:: This on load set fields internal event sets some defaults used for processing of the child.

```
    event "Tabbed Mode set Defaults",
      :set_fields_internal,
      :load,
      :fire_if => "KD.utils.Action.getQuestionValue(\"primaryRequestID\") != \"\" ||
primaryCsrv" do
      field_map "_Tab_Mode_MarkAsComplete", "COMPLETED"
      field_map "subFormMarker", "SUBFORM"
      field_map "Submit Type", "Child"
    end
```

- set Form loading:: This custom on load event tells the parent that the tab for this child has loaded.

```
    event "set Form loading",
      :custom_action,
      :load,
      :fire_if => "primaryCsrv" do
      custom_code "setTimeout('setTabPageLoaded()', 200);"
    end
```

- If Child, go to 2nd page:: This custom on load event submits this page and moves automatically to page 2. This is only appropriate if you are acquiring all of the information on page 1 from the parent and the user doesn't need to interact with it at all.

```
    event "If Child, go to 2nd page",
      :custom_action,
      :load,
      :fire_if => "(KD.utils.Action.getQuestionValue(\"primaryRequestID\") != \"\" ||
primaryCsrv) && clientManager.submitType !=\"ReviewRequest\" " do
      custom_code "document.pageQuestionsForm.submit();"
    end
```

PAGE 2 AND ALL PAGES BEFORE THE SYSTEM FORMS CONFIRMATION PAGE

- Set Parent Id:: This on load custom action sets the parent ID from the answer on the previous page—in this case by pulling it from js on the page (see the `Javascript` text element from the System Forms Element discussion above for an example). This event could also be done Note that this event is not necessary in Kinetic 5.1. The parent ID field on this page can be set to default to the value from the page before.

```
event "Set Parent Id",
      :custom_action,
      :load,
      :fire_if => "clientManager.submitType !=\"ReviewRequest\" " do
      custom_code "KD.utils.Action.setQuestionValue('wrk2 Parent ID', tabModeCsrv);"
    end
```

- Remove buttons for child:: This on load insert-remove event hides custom buttons like “Cancel” and “Save & Exit” when the item is called as a Child. If this child is not also available as stand alone, it would not need to contain these buttons at all, so they would not be hidden.

```
event "Remove buttons for child",
      :insert_remove,
      :load,
      :fire_if => "KD.utils.Action.getQuestionValue(\"wrk2 Parent ID\") != \"\" ||
primaryCsrv" do
      target "Bottom of Page",
        :section,
        :remove
    end
```

- Tabbed Mode:: This on load custom action ensures, now that we are past page 1, that the parent knows what the csrv of this in-progress child is, so the parent can correctly pull it up if the user returns to that page of children in the parent. It also hides the header and footer for the child.

```
    event "Tabbed Mode",
      :custom_action,
      :load,
      :fire_if => "(KD.utils.Action.getQuestionValue(\"wrk2 Parent ID\") != \"\" ||
primaryCsrv) && clientManager.submitType !=\"ReviewRequest\" " do
      custom_code
"ReplaceContentInContainer('headertext',' ');\nReplaceContentInContainer('footertex
t',' ');\n\nKD.utils.Action.setQuestionValue('wrk2 Submit Form',
tabModSystemFormConf);\n\nregisterFormIncomplete(KD.utils.Action.getQuestionValue('wrk2
Submit Form'));\n\n\n"
    end
```

LAST PAGE BEFORE THE SYSTEM FORM CONFIRMATION PAGE

- Tabbed Mode:: This custom on load event will look different on the last page before the “Confirmation” page for the child. The “Submit” button on this page is probably called something like “Continue” or “Next” or “Submit” if the item is in stand alone mode. It should likely be named something more appropriate, ex. “Apply”, if the item is a child. Note that this additional code is only necessary if the item is available as a stand alone. You can simply rename the page’s submit button otherwise.

```
event "Tabbed Mode",
      :custom_action,
      :load,
      :fire_if => "KD.utils.Action.getQuestionValue(\"wrk3 Parent ID\") &&
clientManager.submitType !=\"ReviewRequest\" " do
      custom_code
"\nReplaceContentInContainer('headertext',' ');\nReplaceContentInContainer('footert
ext',' ');\n\nvar subObj=KD.utils.Util.getButtonObject('Submit');\nif (subObj &&
subObj.id) {\n    var but =
document.getElementById(subObj.id).getElementsByTagName('input')[0];\n    var origVal =
but.value;\n    but.value = 'Apply';\n}\n\n"
    end
```

SYSTEM FORM CONFIRMATION PAGE

This page is the last page seen by the customer if this request is called a child.

- ReviewMode:: This on load insert-remove event removes the only section on the page if the request is in review mode. This prevents this page from ever loading as part of a review.

```
event "ReviewMode",
  :insert_remove,
  :load,
  :fire_if => "clientManager.submitType ==\"ReviewRequest\" " do
  target "Tabbed Details",
    :section,
    :remove
end
```

- Remove Buttons:: This on load insert-remove event removes the submit button from view. You never want the user to move forward from this page.

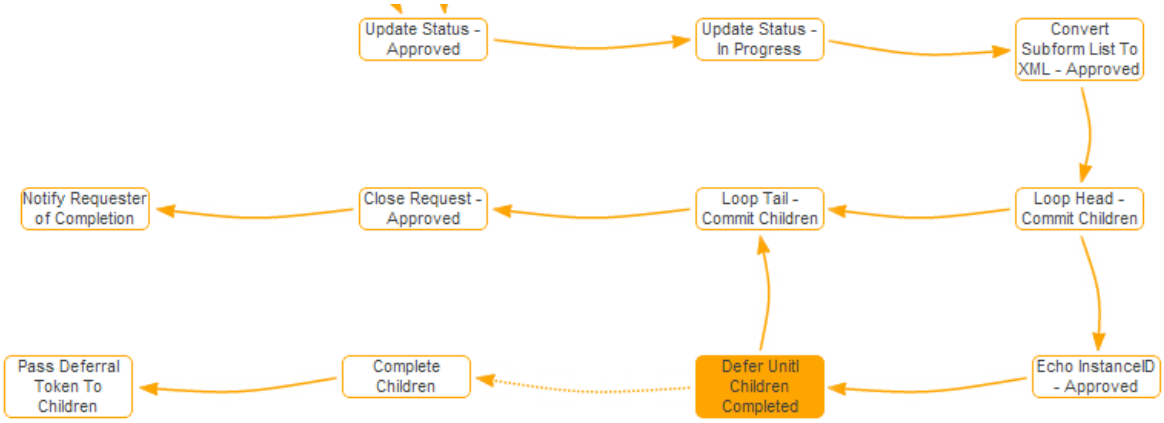
```
event "Remove Buttons",
      :insert_remove,
      :load do
      target :submit_button,
        :remove
    end
```

- Process Variables:: This on load custom event sets the two questions on the page from the javascript on the page and calls the functions to remove the request from the parent's registry as an Incomplete item and to tell the parent that this child csrv is completed for this request.

```
    event "Process Varaibles",
      :custom_action,
      :load,
      :fire_if => "clientManager.submitType != \"ReviewRequest\"" do
      custom_code "KD.utils.Action.setQuestionValue('_Tab_Mode_Csrv',
tabModeCsrv);\nKD.utils.Action.setQuestionValue('_Tab_Mode_SystemForm_Conf',
tabModSystemFormConf);\n\ncallde Register FormSubmitted(KD.utils.Action.getQuestionValue('
_Tab_Mode_SystemForm_Conf'));\nregisterFormCompleted(KD.utils.Action.getQuestionValue('_
Tab_Mode_SystemForm_Conf'));"
    end
```

Chapter 4 PARENT TASK TREE

The task tree of the parent may contain any workflow specific to your requirements, ex approvals, but there is one key component that must be present. The parent item task tree must submit the child records. In most cases, you will not want the parent item to complete until the children are complete. Included here is an example of this component of your parent item's task tree.



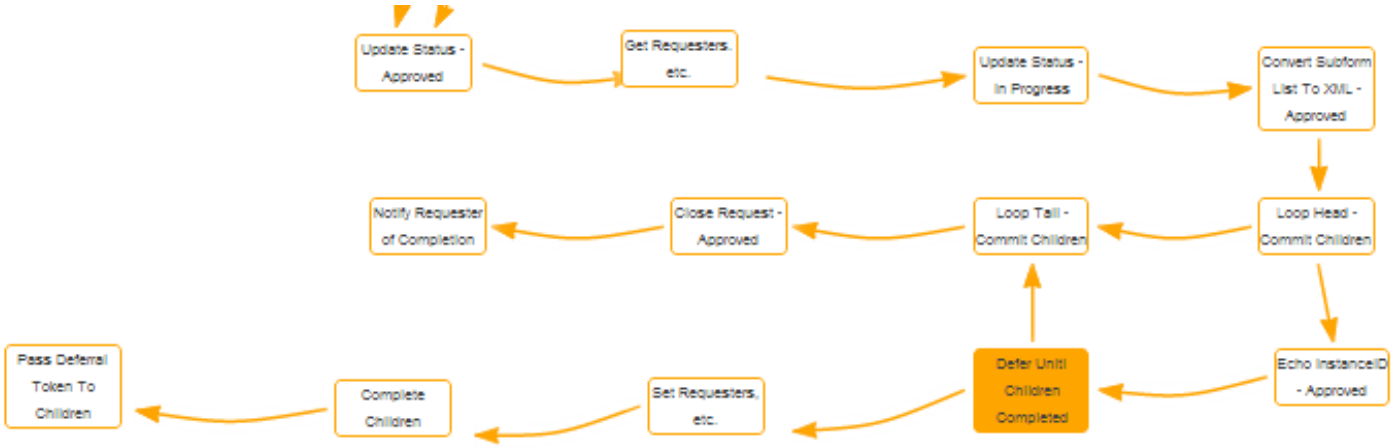
The "Convert Subform List to XML-Approved" node uses the *Utility JSON to XML* handler to convert the information stored in the Form IDs JSON field into an XML string so that the tree can loop through and process each child.

Then the loop processes the results of the previous node (Loop Path: //node/node[@type="Array"]/node/node). The loop first uses an Echo node to echo the instance Id for that loop instance. Then creates a Defer node with an Initial Deferral value of "In Process". The creation of this defer node triggers a *Kinetic Submission Complete* node that actually triggers the completion of the child for the loop instance, and then uses a *Kinetic Submission Update Attribute* node to pass the deferral node into the child request record. The loop tail is an "All", so that all of the children must complete before the parent is considered complete.

NOTE: An attribute should be set aside for use as the deferral token for child records so you are not accidentally overwriting this data with other values. It is recommended, if you have your own dataset, that you clearly name one of the attributes for this purpose.

Once the loop is complete, you can take whatever actions are appropriate in your workflow to close the submission process.

Now, it may be desirable to confirm that the requester and other relevant information hasn't been changed between the time when the user initiated the children records and they pulled that information down and when the record was submitted. This can be done by using the *Kinetic Request Submission Get Answer Set* hander before entering the loop to get the information you want to push to the children (see added "Get Requesters, etc" node in the tree below) and by setting these using the *Kinetic Request Submission Update Answers* handler before the child record is completed (see Set Requesters, etc. node in the tree below).



Another important factor to consider is that if you have an approval on the parent and the item is denied you must cancel or complete these children as denied. You would use a loop similar to the loop to complete the items, but instead of defering, you would cancel/complete as denied. The same holds true if your approval expires. You must loop through and cancel/expire the children.

Chapter 5 CHILD TASK TREE

Making your task tree work as a child might be as simple as adding one node to the completion flow. You **must** add a trigger node that has an action type of "Complete" and uses the deferral token you passed to the child in the parent tree. NOTE: If your child tree has more than one way to complete, ex. approval denied, you must add a trigger node to each branch of the tree that completes.

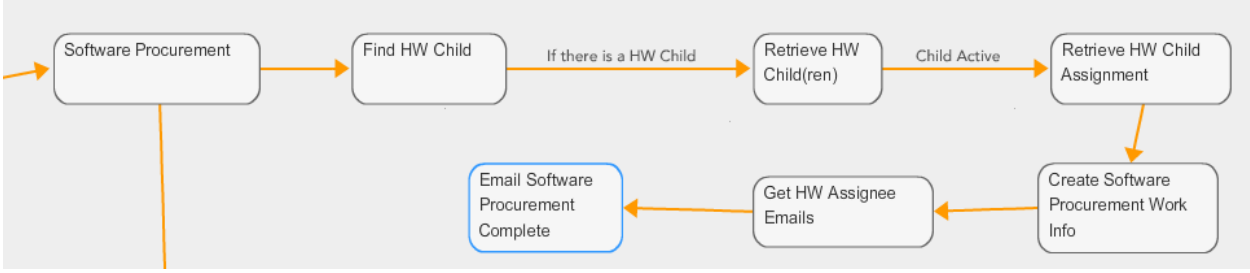
Realistically, there will be more involved to creating your child tree, particularly if that child is also available outside the parent/stand alone. You may have an approval or other item that are only supposed to occur if the request was made as a stand alone. Because of the way the Submit Type was set in your child records, you can use the following check on any connectors to see if an item is a child:
<%=@dataset['Submit Type'] == "Child"%>

You may also be in a situation where the children need to interact in some way (look up each other's status, send emails to the assignees of each other's work orders, set answers in each other, etc). It is convenient to use the KS_SRV_Helper form to store this data. A helper can be created within each child record using the *Kinetic Helper Record Create* handler. If using these records to manage the parent/child relationship, it should contain minimally 4 elements upon creation: An Active status, the instance ID of the parent (Originating ID of the Child), the instance ID of the Child, and the Name of the template of the Child record. Given the number of character fields, you can store these components in any of these fields, but the following are suggested: Status in the Status field, Originating/Parent ID in the SurveyTemplateID field, Child ID in the CustomerSurveyInstanceID field, and the Child Template Name in Character Field1. Where you decide to store these pieces of information is critical, because this is how you will be looking up the records from the parent or other child trees.

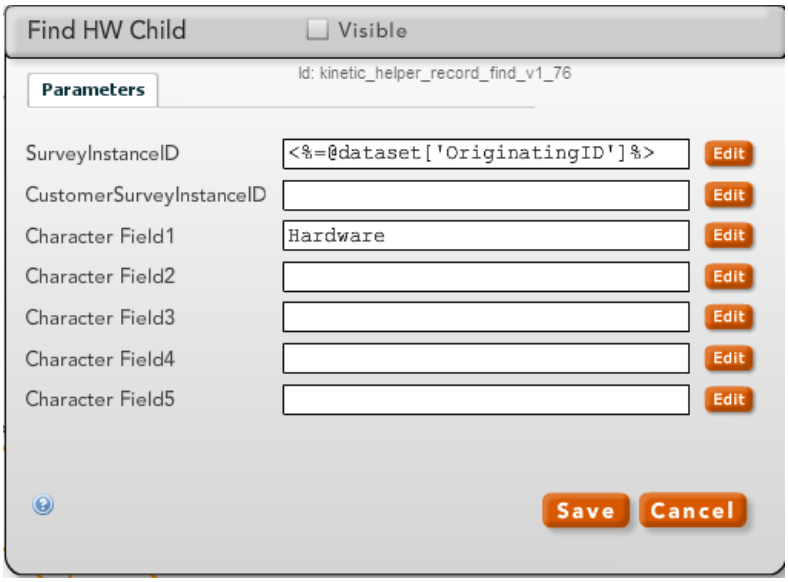
Throughout the workflow of the child, you will want to update this helper record with relevant information from the child. This information may consist of: work order names, work order instance IDs, work order answers, incident or change IDs, approvers, approval or completion dates, child validation status, etc. What you need to store will depend on what data you need to communicate between children. It will be important, though, that one particular update is made. When the child is closed (whether denied, expired, closed, or cancelled) that the child record is inactivated. This will be a way to tell what children are complete via the helper record and will be helpful when trying to manage the maintenance of record volume within the helper form. This is accomplished with the *Kinetic Helper Record Update* handler.

Now, you wouldn't be creating and maintaining these helper records within your child tree unless you needed to store or pass data. The previous paragraphs explain how to create and store this data, but you will also need to be able to find (using *Kinetic Helper Record Find* or *Kinetic Helper Record Find All*) and retrieve this data (using *Kinetic Helper Record Retrieve*), and then use it. There are a number of scenarios you could have where you would need these resources to pass/store data. Two examples will be covered here. First, entering a work information in another child's work order. Second, the case where you would want to pass data into all the other children for the parent.

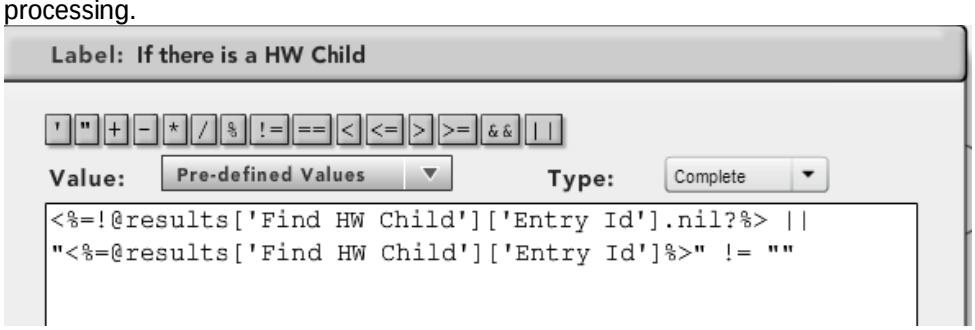
In this first example, one child (eg. Software) will create a work info in the active work order for another child (eg. Hardware) and notify the assignee(s) of the updated work order. In this example, the Software Child needs to notify the Hardware Child when the Software Procurements is complete. For this company, the hardware build cannot be completed until all of the software has been procured and is installed with the rest of the hardware build components. To do this the name and the Instance ID of the active work order for the Hardware child must be stored in known fields in the helper record. In this example, there is only one work order open for the hardware child at time and the active work order instance ID is stored in Character Field2 and the active work order name is stored in Character Field3.



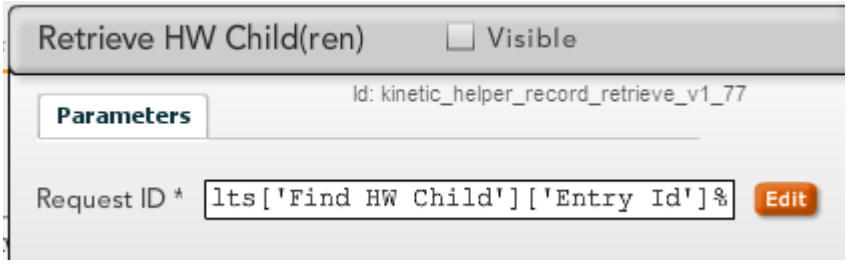
When the Software Procurement Work Order is complete, a look-up using the find for helper records:



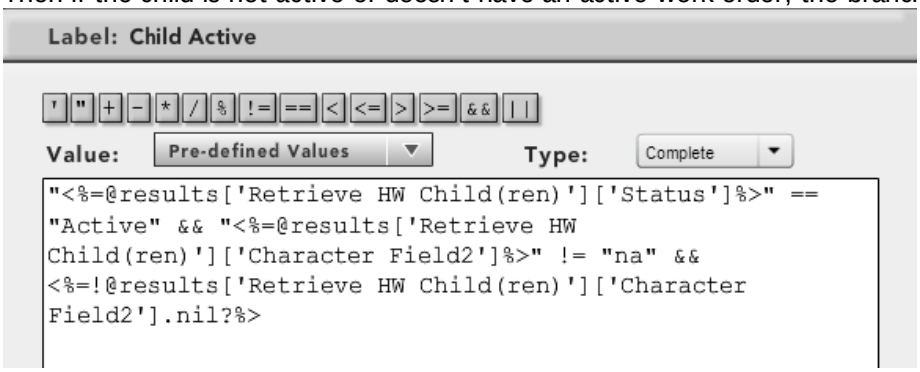
to see if there is a hardware child. If there is no related hardware child the branch stops processing.



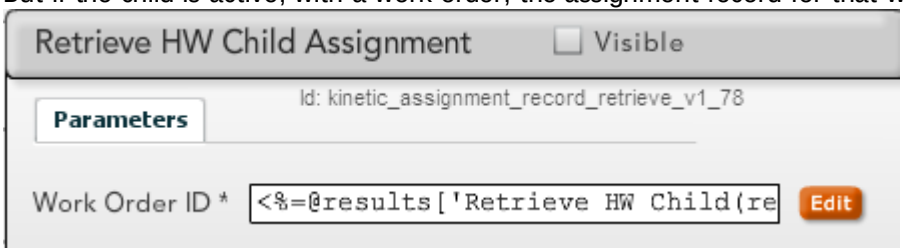
If there is a related hardware child, the helper record found is retrieved.



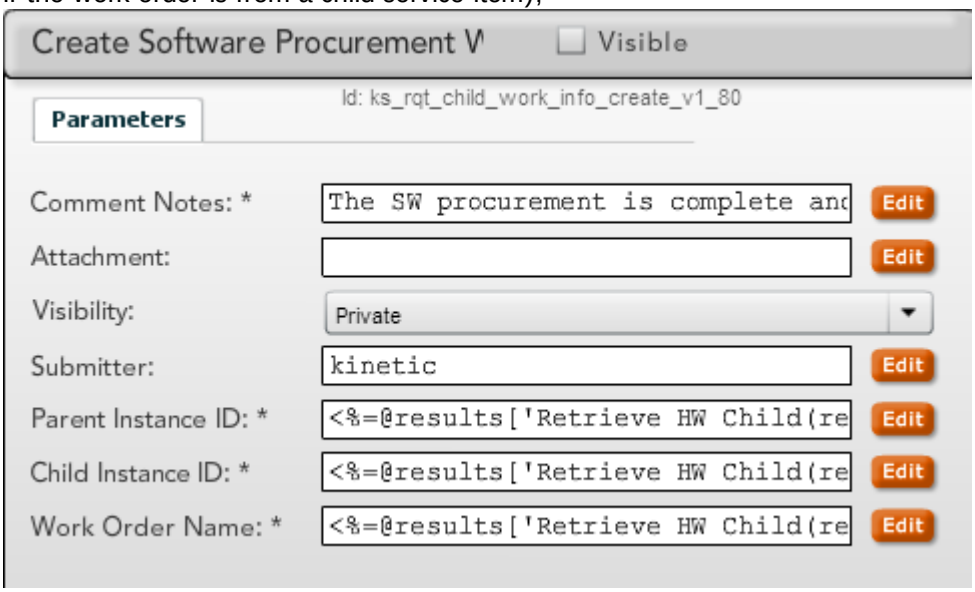
Then if the child is not active or doesn't have an active work order, the branch processing stops.



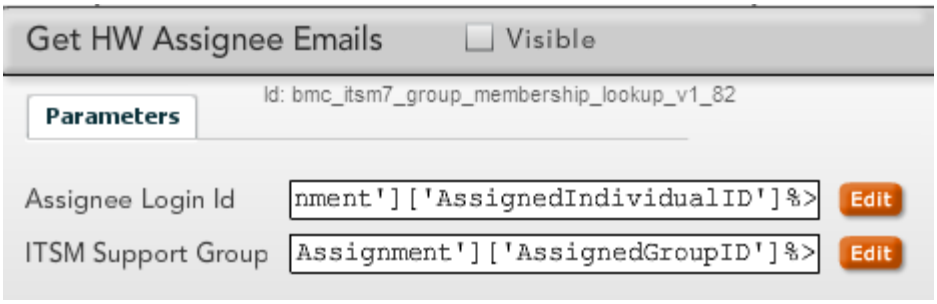
But if the child is active, with a work order, the assignment record for that work order is retrieved.



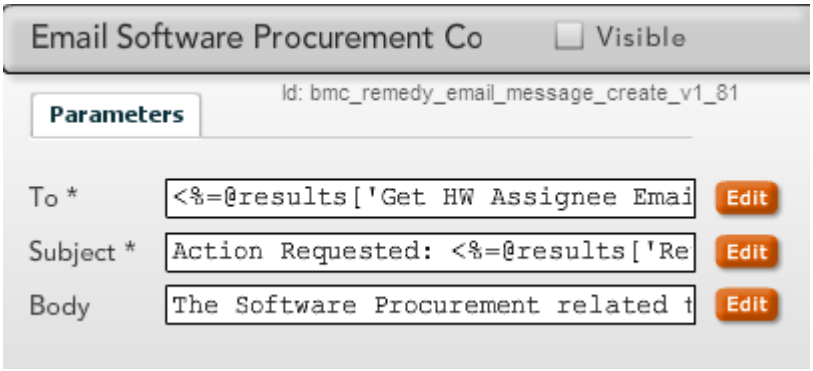
A work order is created (note that creatind a work info for a work order requires a different handler if the work order is from a child service item),



the emails of the assignees are retrieved,

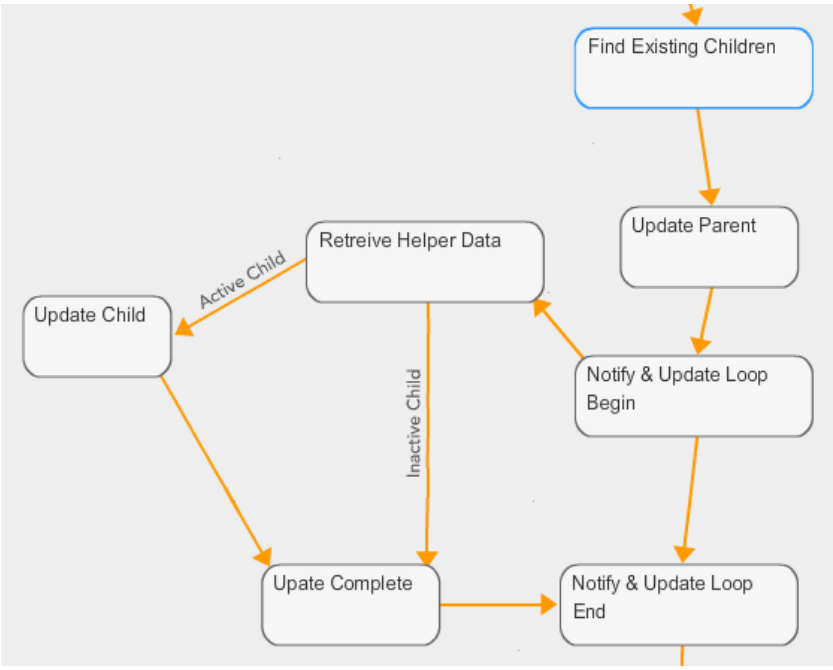


and the email is sent.

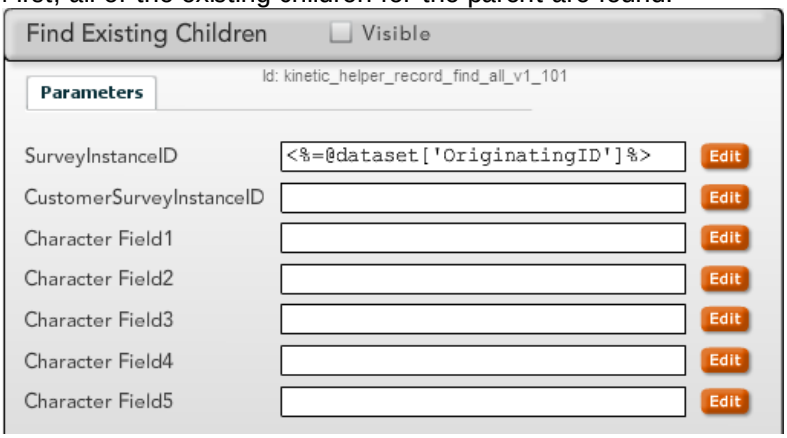


This process can still be followed with ITSM Incidents instead of work order. Different handlers would be needed for looking up assignment and creating work information, but the process would be the same.

The next example is of a network item updating all the parent and all existing children with a login ID and email address when they are entered into work order by the networking team.

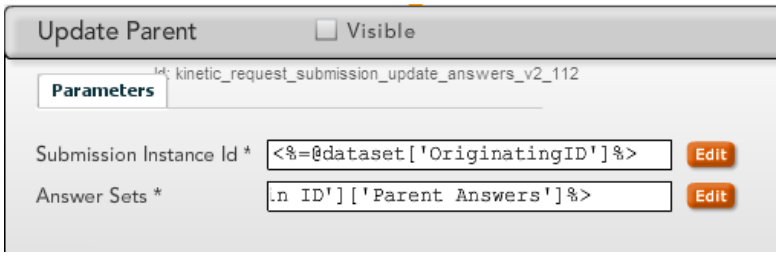


First, all of the existing children for the parent are found.

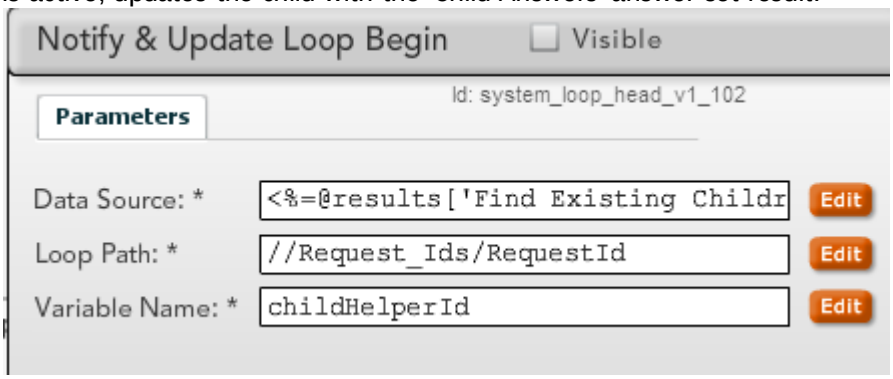


The parent is updated using an answer set in the results of the network work order. This answer set is created with a *Kinetic Request Submission Get Answer Set* handler and added to the results for that work order in its Complete tree (ex <result name='Parent Answers'><%=@results['Get Login ID and Email for Parent']['Answer Set']%></result>). In this case, there are separate answer sets for the parent and children. This is only necessary if the fields being set have different names in the parent and child records. This answer set is set using *Kinetic Request Submission Update Answers* to allow for the network team to override a requested login ID with the actual

one. If it is known that the fields being set are empty, the *Kinetic Request Submission Create Answers* handler will also work.



Then a loop processes each of the children. The loop retrieves each helper record and, if the child is active, updates the child with the 'Child Answers' answer set result.



The Update Complete is an "any" join node used to allow the loop to skip updating closed/cancelled children (whose helper records are inactive).