Create a Loop Using Workflow Activities

Scenario
You're developing in K2 Studio, and need the workflow to iterate over a list and take some sort of action for each list item.

Overview
To create our loop, we're going to use two activities. The first will increment an index variable that will start at 0. This is important, because lists in K2 are one-based. So the first time the incrementation activity fires, the value of the index will become 1. Once completed, the workflow will move to our second activity, which will take care of the heavy lifting for us. At this point, the worker activity will examine and operate on the first value in the list. A rule placed on a line will ensure that the workflow routes back up to the increment activity if the value of our counter not greater than the length of the list.

A rule placed on a second line out of the Incrementation Activity will ensure the workflow escapes the loop once that value has been exceeded.

Steps
Create an Index Variable
1. Click on Object Browser
2. Click the Process/Activity Data tab (the third one down that has an image of a database on it).
3. Expand the Data Fields tree to Data Fields → {Process Name}.
4. Right-click on {Process Name} and click Add... to bring up the Add Data Field dialog.
5. On the **Add Data Field** dialog, name your new variable (I called it “Index”) and change the **data type** to **Long**. (Be sure to do this, because it is set to **String** by default.) *An inline function used later requires use of the Long data type.*

<table>
<thead>
<tr>
<th>Name:</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type:</td>
<td>Long</td>
</tr>
<tr>
<td>Category:</td>
<td>Binary, Boolean, Date, Decimal, Double, Integer</td>
</tr>
<tr>
<td>Initial Value:</td>
<td>0</td>
</tr>
<tr>
<td>Metadata:</td>
<td>String</td>
</tr>
</tbody>
</table>

6. Click **OK** at the bottom of the dialog to close the dialog.

**Create the Incrementation Activity and its Data Event**

1. On a blank area of the design canvas of your process, right click your mouse pointer and draw a capital “D”.

This will automatically create a **Data Event** inside of a new **Default Activity** (Sweet, right?) and will launch a **Data Event** configuration wizard.
2. Click **Next >>**.
3. Change the event name to **Increment Index** and ensure the **Data Replication** type is set to **Transfer Data** (it is, by default). Click **Next >>** to move forward to the **Data Locations** panel.
4. Click the **Assign** button in the horizontal bar near the top of the panel.
5. On the **Select Source and Destination** panel, you should see two textboxes.

Perhaps the best analog I have for K2’s notion of **Source** and **Destination** hails from the cassette era. Suppose you were awesome enough to have a dual cassette deck component to your stereo. On that component, the deck on the left had your standard functions: play, rewind, fast-forward, stop, eject. But the deck on the right had the standard functions PLUS a record function. Say you want to make a mix tape of your favorite songs. You’d place the tape you’re copying from in the deck on the **left** (the one with just the play buttons – the source deck), and the tape you’re copying to in the deck on the **right** (with the record button – the destination deck). You can put whatever tapes you want into that source deck, but it’s all getting recorded onto that cassette in the destination deck.

Okay. Back to the future. I’m going to explain the **Destination** field first:

**Destination**

**Destination** refers to the variable containing the product of the data operation. For this wizard, we’re incrementing the value of our Index variable. So your Index variable is what goes in the Destination field. Here’s how.

a. On the right side of the wizard you have a compact copy of the solution explorer, called the **Context Browser**. It is docked to the wizard.

b. Find your variable by clicking on the **Process/Activity Data** tab (it’s the third tab down, with an image that resembles a database). Expand the **Data Fields** tree to **Data Fields → {Process Name}**. You should see your variable listed below the process name.

c. **Left-click and hold** the name of your variable until you see your mouse pointer change to display a plus symbol. Now drag it over to the **Destination** field and release the button. You should see your variable name in white on a green field inside the textbox.

**Source**

**Source** refers to whatever you’re going to place inside the **Destination** field. This is an important concept, because you can do whatever operations on the data you want via the **Source** field. In this case, we’re going to simply increment our index variable. Here’s how.

a. Go back to the **Context Browser** on the right side of the wizard.

b. We’re going to create an expression for the math we’re going to do. On the bottom tab (the **Function Browser**, marked with the function symbol), expand the **Expression** node and drag it into the **Source** field. Remember you have to left-click it and pause briefly
until the mouse pointer changes to display a small plus sign. The Build Expression panel will be displayed.

c. Back in the Context Browser, find your index variable as you did above and drag it into the large field on the Build Expression panel. If done successfully, you should see your variable name in white on a green field inside the textarea.

d. Click the “+” button and type “1”. Your expression should now look like this:

![Build Expression panel](image)

```
Expression Name: Index + 1
Index + 1
```

e. Click Finish.

6. Your completed values on the Select Source and Destination panel should look like this (the data in the Source field is also highlighted, so it looks greenish-blue):

![Select Data Source and Destination panel](image)

```
Source:
Index + 1

Destination:
Index
```

7. Click Finish.

8. Your completed values on the Data Locations panel should look like this:

![Data Locations panel](image)

```
Source | Destination
-------|-------------
Index + 1 | Index
```

9. Click Next>>, then Finish.
10. Double-click on the name of the activity and rename it to something that makes sense for your workflow.

Create the Worker Activity and its Data Event

1. Draw another “D” on the canvas just like you did before. This will automatically create a Data Event inside of a new Default Activity and will launch a Data Event configuration wizard.
2. This time, instead of doing operations to increment our index variable, we’re doing some actual work. The only part of this I want to explain is how the index variable is used to move through our list. I’m using two inline functions for this: Index Item() and Split(). Using these functions and my index variable, I’m able to pick out a specific value from the list for further operations:

   a. Go back to the Context Browser on the right side of the wizard.
   b. In the Function Browser, expand the Lists node and drag Index Item() onto the Source field to expose the Configure Function panel.
   c. The Configure Function panel features three textboxes: Values refers to the list. Index refers to the n\textsuperscript{th} value in that list of values. Expected When Empty refers to a value to be returned if the n\textsuperscript{th} value in that list is empty.
   d. Use the Context Browser to find the Empty String() function. It’s located under the Text node in the Function Browser. Drag it to the Expected When Empty field.
   e. Use the Context Browser to find your index variable. Remember it’s located on the Process/Activity Data tab, under the Data Fields tree. Drag it onto the Index field.
   f. This leaves us with the Values field. Go back to the Text node in Function Browser and drag the Split() function into the field. This will launch another Configure Function panel.
   g. This second Configure Function panel features two fields: Text, which refers to our list, and Separator, which is the character the function will use to split the list. Place your values in the fields and click Finish to return to the previous form.
   h. In my case, I’m splitting a comma-delimited string called UsersApproved. Here’s what my Configure Function panel looks like:
3. To recap, I’m using `Index Item()` and `Split()` to isolate a specific value in the list. I may be performing additional operations on that value as well. The product of all of these operations is the Source data to be poked into my Destination variable.

4. Double-click on the name of the activity and rename it to something that makes sense for your workflow.

5. Place the activity below the **Incrementation Activity** you created previously.

**Connect the Activities**

The key to the loop is the rules that are placed on the connections.

1. Right-click on the **Incrementation Activity** and draw a line down to the **Worker Activity**.
2. Right click on the line and select **Properties**.
3. The Line General Properties wizard has three panels: **General Properties**, **Line Rule**, and **Exception Property**. You may access each of these by the icons arranged vertically and to the left of the panel area.
By default, the **General Properties** panel is displayed (note the shading of the top icon).

4. It is good practice to label lines that have rules associated with them. We want to type in a description of the rule being applied. The label is displayed on the workflow canvas, so it becomes very easy for someone unfamiliar with your work to understand what’s happening in your workflow.

   In my case, I have a variable called UsersSelectedCount that contains a count of all of the items in the list I’m iterating over. So, in the **Label** field on the **General Properties** panel, I have typed in: “RULE: Index <= UsersSelectedCount”.

5. Click on the icon for the **Line Rule** panel. We’re going to add the rule we just described in the label:
   a. Click **Add** to bring up the **Add/Edit Rule** wizard.
   b. The **Add/Edit Rule** form has several textboxes and dropdowns on it, in various states of ability. We’ll focus on three: **First Variable**, **Comparison Operator**, and **Second Variable**.
   c. Click the black button (with the white ellipses) to bring up a **Context Browser** window. Unlike in the previous wizards, this will be undocked. Use it to find your index variable. Drag your index variable into the **First Variable** field.
   d. Use the **Comparison Operator** dropdown to select <=.
   e. Use the **Context Browser** to locate the variable or expression to which you’re comparing your index variable. In my case, it’s that UsersSelectedCount variable.
f. My completed rule looks like this:

![Add/Edit Rule Form](image)

- First Variable: Index
- Logical Function: 
- Logical Data: Type text and/or Drop a single field
- Comparison Operator: \( \leq \)
- Second Variable: UsersSelectedCount

![Ok and Cancel Buttons](image)

- Click **OK** to close the form.

h. My completed **Line Rule** form looks like this:

![Line Rule Form](image)

- Click **Finish** to close the wizard.
6. You now have a downward connection between the **Incrementation Activity** and your **Worker Activity**.

7. Now, right-click on the **Worker Activity** and draw a line up to the **Incrementation Activity**. There are no rules associated with this line.

**The Great Escape**

1. Finally, we need an escape route for the **Incrementation Activity** once your index variable has been incremented beyond the upper bound of what the existing rule will handle.
   a. From the **Activity Wizards** panel of the **K2 Studio Context Browser**, create a new **Default Activity** and place it beside the Worker Activity.
   b. Rename the new activity to something meaningful for your process. We’re not worried about fleshing everything out in it right now; we’re more interested in making sure the loop gets escaped properly.
   c. In the **Event Wizards** panel of the K2 Studio context browser, drag a **Placeholder Event** inside of the new activity.
   d. Right-click on the **Incrementation Activity** and draw a line down to that new activity.
   e. Follow steps 2 through 5 under “**Connect the Activities**” to create a rule that will allow the loop to be exited. In my case, that rule might resemble “Index > UsersSelectedCount”.

---

[Diagram showing connections between Incrementation Activity, Worker Activity, and Escape Activity]
Deploy and Test
You now have all the pieces in place. Save, compile, and deploy your process.

When done right, you should be able to monitor the activity and values from the workspace.

Once the process completes, you should see the incrementation and loop activities appear one after another for each item in your list, like the example at right. In that example, I had a list of five people. The incrementation activity (called “Denied Users – Count” here) fires a total of six times – the first time is the initial increment (from zero to one), and the rest are looped back from the worker activity (“Denied Users – List”).

Additionally, if you drill into the data recorded in the process, you should see your index value is one greater than the number of items in the list – just like the number of occurrences of the incrementation activity.

Summary
You’ve learned how it is possible to create a loop using workflow activities, and how to use inline functions to split and iterate over a list. This was accomplished through creating an index variable, creation of an activity and a data event for incrementing that index variable, creation of an activity and a data event for doing work on a specific list item, connecting those activities and using line rules, and creating an activity and line rule for escaping the loop.

Afterword: Real-World Applications
Skills such as those demonstrated here are particularly valuable in the context of the use of repeating controls in Microsoft InfoPath. For example, one can use a multiple-selection list box on an InfoPath form, and bind it to a variable. When the collected data is passed to a variable in the workflow, it may be converted into a comma-delimited list. Using the skills you learned here, you may easily iterate over the list and operate on the list items individually.

Conversely, you may use these skills to drive a cleaner and nicer UI experience for the user. For example, the user can now use checkboxes to approve requests for multiple authorizations all in a single form instead of having to approve separate requests for single authorizations. It saves the business time and creates enthusiasm for your product from the people who use it most.