

How To Use Loops

This document is short treatise on some uses of Loops in TDI AssemblyLines.

Introduction

Loops can be used to solve a number of scenarios that otherwise would require a good deal of scripting. Typical examples are:

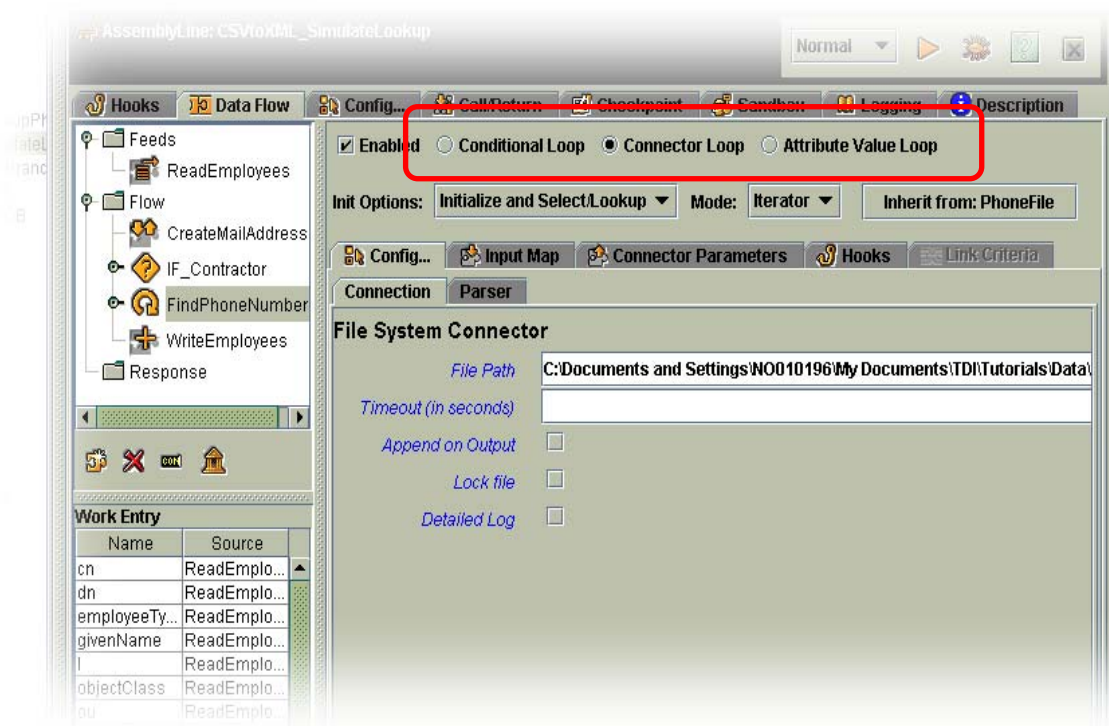
- Adding a “nested Iterator” – e.g. an Iterator that appears in the Flow section of the AL.
- Handling multiple found situations when doing a Lookup
- Cycling through the values of a multi-valued Attribute
- Branching based on the success of a Lookup
- Creating unique data values, like uids in a directory

Before exploring these scenarios, let’s take moment to look at the Loop Component and its settings.

The Loop Component

The Loop can be thought of as an AL *Branch* that repeats, or *cycles*. These two constructs are so closely related that you use the same method to exit Branches and Loops: the `system.exitBranch()` call.

When you set up a Loop, the first setting to decide on is which *type* of Loop you want.



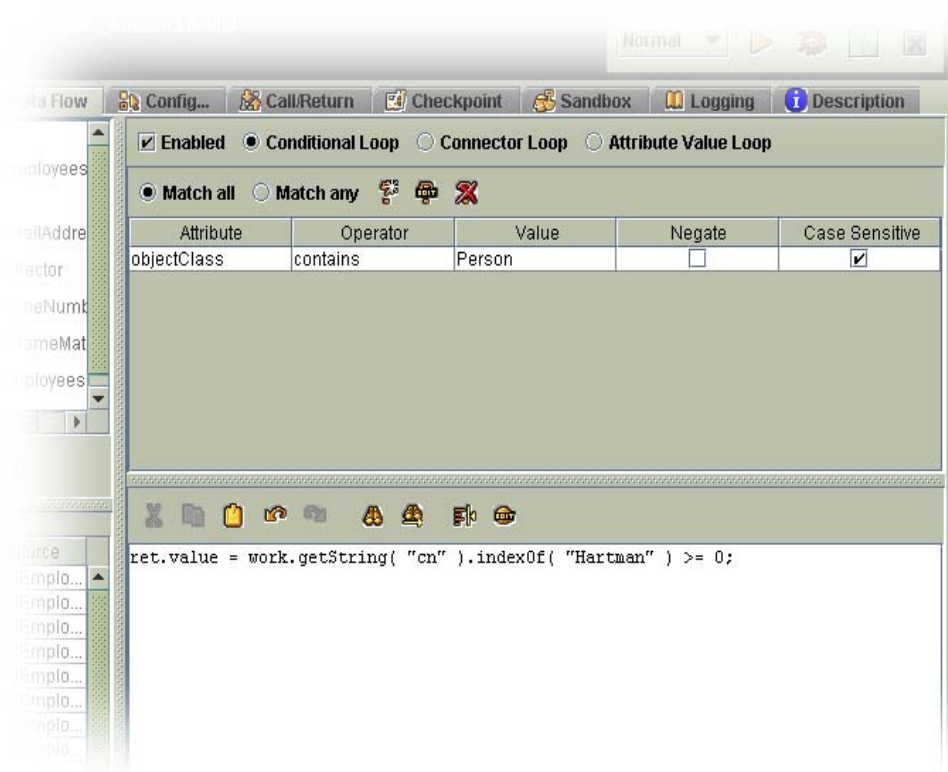
There are three types of Loop available:

Conditional	This Loop will cycle until the Condition(s) evaluate to <i>false</i> .
Connector	This Loop drives a Connector in either Iterator mode or Lookup, cycling for each Entry found.
Attribute Value	This Loop type cycles once for each value found in the specified Work Entry Attribute.

Conditional Loop

The Conditional Loop provides an interface similar to Link Criteria. You have the choice of setting up Simple Conditions; scripting an expression that returns either *true* or *false*; or you can combine Simple and Scripted Conditions..

It is important here to note that unlike Link Criteria, Conditions do not allow you to use the dollar (\$) or at (@) symbols in the Value field in order to access Work Entry Attributes. To do this you need to script the Condition (as shown below).

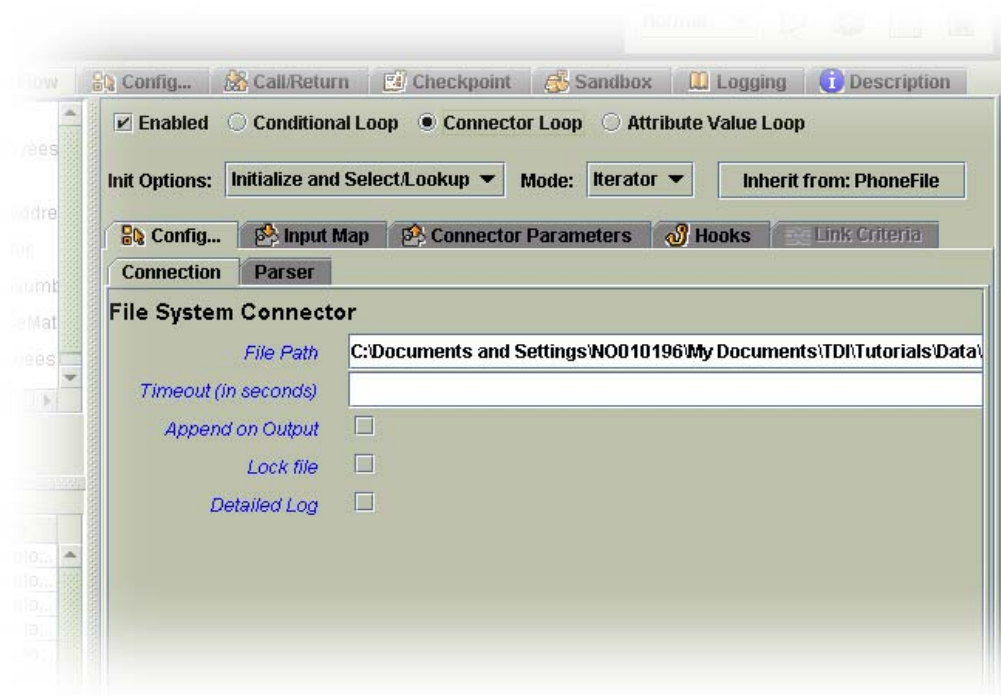


Connector Loop

This type of Loop has an **Inherit From** button for tying a Connector to the Loop. There is also a Mode setting so that you can configure the Loop to drive the associated Connector in either Lookup or Iterator mode.

In Iterator mode, the Loop behaves like a *sub-AssemblyLine* – e.g. driving components under this Loop once for each Entry returned, just like Iterators in the Feeds section drive components in the Flow section. Note that there is a **Connector Parameters** tab for mapping the various Connector parameter settings (such as the SQL SELECT or LDAP Searchfilter) in order to control the result set of the Iterator.

In Lookup mode, the Loop will cycle for each Entry found that matches your Link Criteria. This allows you to deal with Multiple Found situations without resorting to Hook scripting.

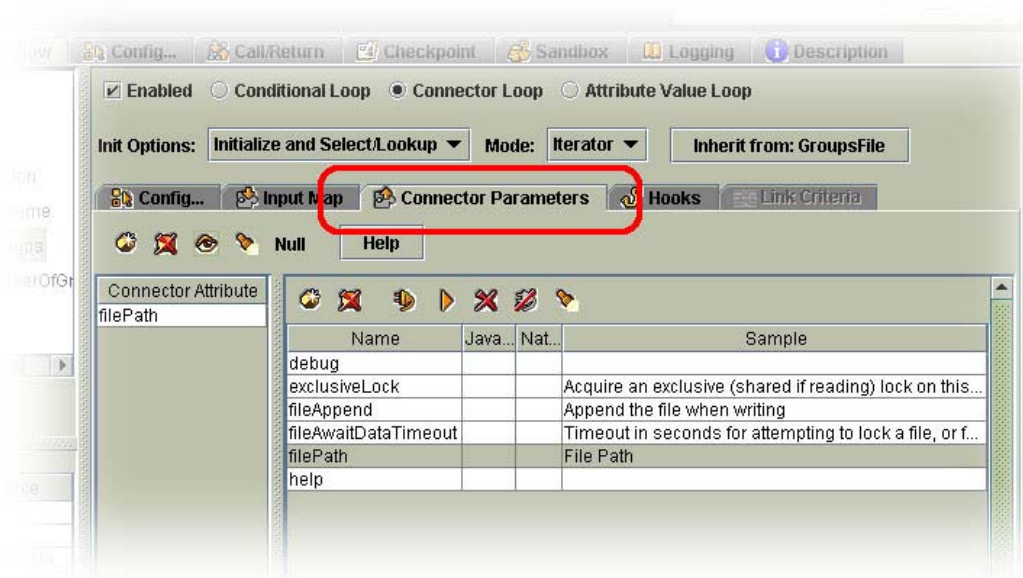


With both Iterator and Lookup modes you have an Init. Options setting to determine if the associated Connector should be (re)initialized each time the Loop is encountered during AL operation; or if only Lookup or Select (depending on the Mode setting) needs to be re-issued; or finally, if nothing should be done with the Connector.

Normal Hook flow occurs during Connector Loop operation, except for the **On Multiple Entries** of Lookup mode, as this is handled automatically by the Loop.

Connector Parameters tab

One tab of particular interest here is Connector Parameters. This panel allows you to *map* values to the parameters of the Connector associated with this Loop. These settings will be used for Connector re-initialization and selecting entries for iteration (depending on the Init Option settings).

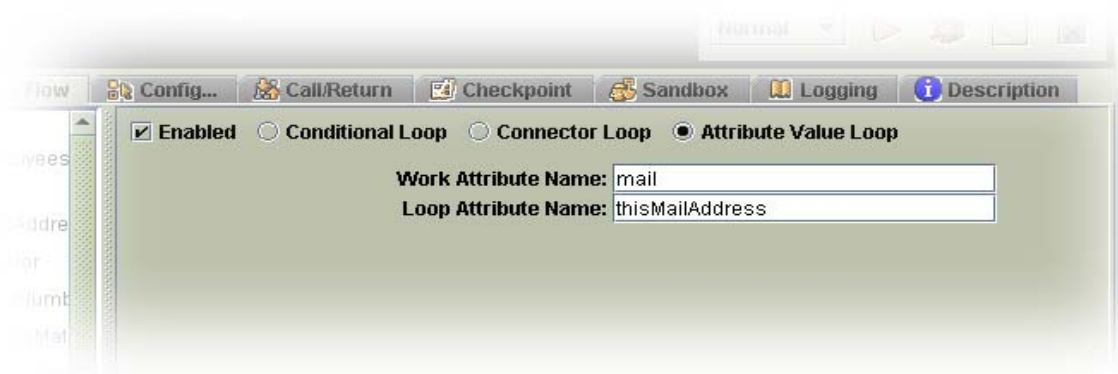


The **Connector Parameters** map works like an Attribute Map, allowing you to use Work Entry Attribute values, as well as Advanced Mapping scripts to set parameters. Note also that the **Sample** column, which is used in Attribute Maps to display data from the connected system, shows the tool tips for the parameters instead.

For example, you could have an AssemblyLine with a FileSystem Iterator that is reading a file with information on a number LDAP Servers (LDAP URL, id/password, search base and filter, etc.). The Flows section contains a Loop tied to an LDAP Connector with Init Options set to “Initialize and Select”. The details of each LDAP server read in by the FileSystem Iterator are mapped to this Loop Connector, so that each time the Loop engages, it will be connecting to (and iterating from) a different server.

Attribute Value Loop

This third Loop type allows you to specify the name of the Work Entry Attribute to cycle on,



as well as a **Loop Attribute Name**. The Loop will cycle for each value found in the specified Work Entry Attribute, placing this value in a new Attribute named as specified in this second parameter.

Scenarios

The zip-file associated with this document contains the following files:

UsingLoops.xml	This is the example Config.
People.xml	A data file with People information, including a unique id (uid).
Groups.xml	Contains data on groups and their members. This file contains a GroupName fields, as well as one called Members that contains zero or more member uids.

The UsingLoops Config has three AssemblyLines, each addressing a different scenarios using the techniques listed at the start of this document.

01_SimulateLookup

Displays (logs) each person in the People file, along with group memberships.

This AL Iterates through the People file. For each person Entry read, it then searches the Groups file to find all memberships. It uses a Connector-based Loop to Iterate through the Groups file and a Branch to find matches.

02_MultiValuedAttribute

Creates the GroupsDB table in Cloudscape and populates it from the Groups file.

In order to do the third AL, I needed to move my Groups info to a data source that supported lookup. So the second example here creates a GroupDB table (using direct SQL calls to the bundled Cloudscape database), and the populates it from the Groups file.

In order to make the AL easier to read, I put these scripted SQL calls in a Script Component. This needs to run first so that there is a table to write to. Since SCs can only be added to the Flow section of the AL, a Loop must be used to Iterate over the Groups file.

As this Loop works its way through the Groups, another subordinate Loop is used to add a new group-member record for each value of the Members Attribute.

03_HandleMultipleFound

Displays (logs) each person and the groups they are member of.

Iterates through the People file, doing a Lookup in the new GroupsDB. Since some people are a member of multiple groups, this can result in multiple Entries found. A Loop is used to handle this, as well as single matches. Note that you still need to script the On No Match Hook if some searches can fail.

04_BranchOnLookupSuccessful

Only writes to Notes if Lookup in DB2 successful.

This AL does nothing, apart from showing off the Loop as an If_LookupSuccessful tool.

05_CreateUniqueld

This exercise is left to the reader.

Imagine the second example above with another Loop around the AddGroupMember Connector. Just before the Loop you have a Script Connector where you create a counter variable and initialize it (e.g. `var addCount = 0;`)

The Loop around the AddGroupMember Connector (let's call it LoopUntilAdded) is Conditional, with a single Scripted Condition:

```
ret.value = addCount < 10;
```

As long as the addCount is less than 10, the Loop will continue to drive the AddGroupMember Connector to add a new Entry. One of the Attributes in the Output Map could be scripted to use the addCount as needed to create a unique value (called "uid" in the example below):

```
if (addCount == 0)
    ret.value = work.getString( "uid" )
else
    ret.value = work.getString( "uid" ) + addCount;
```

This way, addCount will only be used after the first attempt.

The following Hooks would need scripting as well:

Add Successful	// Set the addCount to 99, to end the Loop addCount = 99;
AddOnly Error	// Increment the addCount addCount++;

Please send corrections/comments/suggestions to eddie.hartman@no.ibm.com.