Job Control Story Report: JC7 – Define Log Tagging

Revision	Reason	Author	Date
0.01	Initial document	Peter Villiers	13Dec2010
0.02	Corrected spelling errors	Ann Bagley	15Nov2012

Story Summary:

As a Job Control Manager, I want to be able to tag in-process jobs with information from the job log, so that I can better trace them.

Preconditions:

- The Installer has installed the Job Controller client application.
- The server side of the application has been installed.
- The client side application has been tested to confirm that it works.
- The Job Control Manager has the ability to copy & edit the configuration files associated with the client application.

Story Path (Main):

The client side application has been installed and is known to work. The job control manager wants to define what should be searched for in the running SAS logs. This will allow the client side application to get more information about what is actually running on the server. In turn, this will allow for core complete historical data.

In the following text, JC_HOME is meant to be the location where the Job Controller client is installed.

There are 2 options for how to maintain the tagging rules:

- The JC_HOME/conf/sddRules.xml file can be edited directly. This is the simplest in terms of initial setup. HOWEVER, if a new version of the client is installed, the file should be backed up prior to the installation. The tagging definitions will then need to be transferred to the newly installed file.
- The JC_HOME/conf/sddRules.xml file can be copied to another location and edited there. In this case, the JVM option "-Djc_tag_defs=*the-path-to-the-custom-sddRules.xml*" can be added to the startup command to point to the custom file. This is the preferred option as it separates the installed application from the custom configuration.

The second option will be illustrated in the steps below. An example configuration directory will be shown however this could be anywhere:

1. Navigate to the parent directory of JC_HOME.
2. Create a directory called "JobController-customConfig".
3. Copy the sddRules.xml file from "JC_HOME/conf" to the "JobController-customConfig".
4. Edit the "JobController-config/sddRules.xml".
5. Save the file.
6. Remember to add the startup option to any scheduled jobs that should use the new tagging definitions.

Setting the number of lines to search in each log:

1. Set the value of the SddJobController/Tagging/TaggerGlobalProperties/LinesToSearch element to be the number of lines that will be searched.

Adding a new Tagger Definition

The element SddJobController/Tagging/Taggers contains a sub-element for each tagger.

1. Either:
    a. Copy/paste an existing element that is similar to what is needed, or
    b. Add a new one from scratch.

Adding a BasicTokenTagger

BasicTokenTagger takes the individual lines from a log file and splits them up based on a delimiter. It them compares one of these parts to a value specified. If the condition is met (the value is found) then the same line is split up using a potentially different delimiter and one of those values is returned.

In the following "@xxxxx" is meant to denote an XML attribute.

The main element <BasicTokenTagger> can have the following sub-elements:

- <SelectTokens> - identifies the sub-elements are used to "search" the input line. If ANY of the sub-conditions evaluate to true, then the line will be tagged.
    - <SelectToken> - is a unique condition that will be tested on the line. The value of the element is value which will be tested.
        - @delimeter – the set of characters that are to be used as delimiters in splitting up the string. The default is ":".
        - @tokenNumber – when the line is split using the delimiter, this is the number of the "part" to be returned. Must be greater than or equal to 1.
        - @tokenComparator – used to define the condition to be used when comparing the token and the provided value. Valid values are "eq", "contains" and "startswith".
        - @stripBlanks – if "true", will remove leading and trailing blanks from the token prior to comparing it with the value.
- <TagTokens> - a list of elements that define how tags will be pulled from the input line.
    - <TagToken> - identifies how to pull a tag out of the input line.
        - @delimeter – the set of characters that are to be used as delimiters in splitting up the string. The default is ":".
        - @tokenNumber – when the line is split using the delimiter, this is the number of the "part" to be returned. Must be greater than or equal to 1.
        - @stripBlanks – if "true", will remove leading and trailing blanks from the token prior to comparing it with the value.

As an example consider the following:

```
<BasicTokenTagger>
        <SelectTokens>
                <SelectToken delimeter=":" tokenNumber="1"
                        tokenComparator="contains" stripBlanks="true">TEST FIELD</SelectToken>
        </SelectTokens>
        <TagTokens>
                <TagToken delimeter="/" tokenNumber="3" stripBlanks="true" />
                <TagToken delimeter="/" tokenNumber="4" stripBlanks="true" />
        </TagTokens>
</BasicTokenTagger>
```

When it operates over a line from a SAS log:

```
96          TEST FIELD               : /level-A/Level-B/level-C/level-D
```

It will do the following:

- Search for the select token:
    - Split the line based on the delimiter ":".
    - Get the first part "`96          TEST FIELD          `".
    - Remove leading and trailing blanks.

    - Check if it contains “TEST FIELD”.
- In this case, the token is found so the definitions that pull the tags will be evaluated.
    - Evaluate the first TagToken:
        - Split the line based on the delimiter “/”.
        - Get the third part “`Level-B`”.
        - Strip the blanks.
        - Add it to the list of tags for the line.
    - Evaluate the second TagToken:
        - Split the line based on the delimiter “/”.
        - Get the third part “`Level-C`”.
        - Strip the blanks.
        - Add it to the list of tags for the line.

So the tags returned for this line will be “Level-B Level-C”.

Considerations / Assumptions:

- The more log lines that are searched, the more load that will be placed on the SDD system and the longer the data collection step will take.
- The more tagging definitions, the more load that will be placed on the SDD system and the longer the data collection step will take.

Test Automation:

- Some JUnit tests for testing select conditions and completeness of the select/tag tokens.

Known Limitations:

- None