Pentaho Data Integration (PDI) Techniques - Restartability

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes
9/28/2017	1.0	Christopher Morehouse	
2/14/2019	1.1		Update to 8.2

Contents

This page intentionally left blank.

Overview

This document covers some best practices on building restartability architecture into Pentaho Data Integration (PDI) jobs and transformations. In the event of a failure, it is important to be able to restart an Extract/Transform/Load (ETL) process from where it left off. This is true whether you need to avoid duplicate entries in the target database, or you are simply seeking overall ETL efficiency and don't want to rerun processes that completed successfully in the previous run.

Our intended audience is Pentaho ETL developers and architects.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	6.x, 7.x, 8.x

The [Components Reference](#) in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

Prerequisites

This document assumes that you have general knowledge of the PDI user interface and have already installed it.

Use Case: Unreliable Network and Database Connections

Janice is a database administrator who has been tasked with running jobs on data stored in a location with an unreliable network. The CIO has decided not to upgrade the network at this time, but Janice must still deal with jobs that fail due to the network connections malfunctioning. Right now, she must restart the entire job when the network goes down, but she is looking for a solution that will allow her to restart a job from the last known good operation. Janice plans to use checkpoints for running her jobs. She expects that this will save her time, and impress the bosses since she will be able to get them their results with less trouble than in the past.

Restartability

Different methods for restarting PDI have their own advantages and disadvantages. Your situation will affect which path is the right one for you.

You can find details on these topics in the following sections:

- [Restartability Using Checkpoints](#)
- [Restartability Using Control Table](#)
- [Making the Job/Transformation Transactional](#)

Restartability Using Checkpoints

At the job level, PDI has the built-in capability to restart at certain points called checkpoints. Adding checkpoints to the hops connecting one of your job entries to another creates the option for you to restart a failed job from the checkpoint, rather than restarting the entire job from its beginning. This useful feature and its configuration options are documented in [Use Checkpoints to Restart Jobs](#).

The simplicity of checkpoints leads to certain limitations for jobs that deal with batch processing. For example, a single subjob or transformation may handle processing many files. If the parent job fails, the subjob or transformation responsible for processing all those files would have to read all of the files again upon restart, even if some of the files were already processed during the previous run.

A [control table](#) may help in this circumstance.

Restartability Using Control Table

Create a control table in a database to act as a queuing mechanism to keep track of which files have already been processed. That way, if the job fails and you restart it, the job will be able to pick up from where it left off and continue processing unprocessed files, instead of having to start over as it would if you used a [checkpoint](#).

To illustrate how useful a control table can be, this figure represents a job using this architecture:

Figure 1: Job Using a Control Table

1. In the job, the **Get Files** transformation will read the files from the filesystem, and insert the filename into the control table:
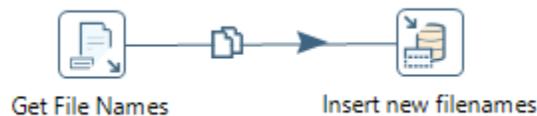

Figure 2: Get File Names and Insert New Filenames Steps

2. Each row in the table will have a **flag** column that marks whether that file has been processed or not. When a filename is inserted into the table, the flag will have a default value of `null`.

3. When a file is processed, the flag value will change to `1`.

Figure 3: Data Output from Steps

4. The **Read Files From DB Table** transformation will only read the files that have not been processed already. For this, use an SQL query that pulls only filenames where the flag value is `null`:

```
SELECT filename FROM ctl_table WHERE successful is null;
```

5. The resulting dataset will then be stored to result, using the **Copy rows to result** step, for further processing in the next transformation:

Figure 4: Get Filenames and Copy Rows to Result Steps

6. Check the box on the **Process Files** job entry to **Execute for every input row**, so that the **Process Files** transformation will execute once for every filename to be processed.

   At the end of the transformation, the file's flag will be marked with a `1`. If the job fails, any files flagged with a `1` will not be reprocessed during the next run.

Figure 5: Execute for Every Input Row

7. Create the **Process Files** transformation with a **Get rows from result** step and a **Text file input** step, so it can read the filenames stored to result by the previous transformation
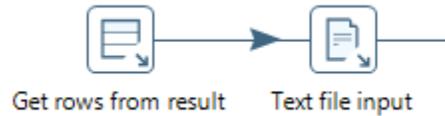

Figure 6: Get Rows from Result and Text File Input Steps

Making the Job/Transformation Transactional

Setting the **Make the transformation database transactional** option on a transformation or job, as in the following figure, will roll back any data in the event of a failure.

This option is a way to regulate duplicates during the restart of a failed job run. More information on the topic is available at Database Transactions in Jobs and Transformations.
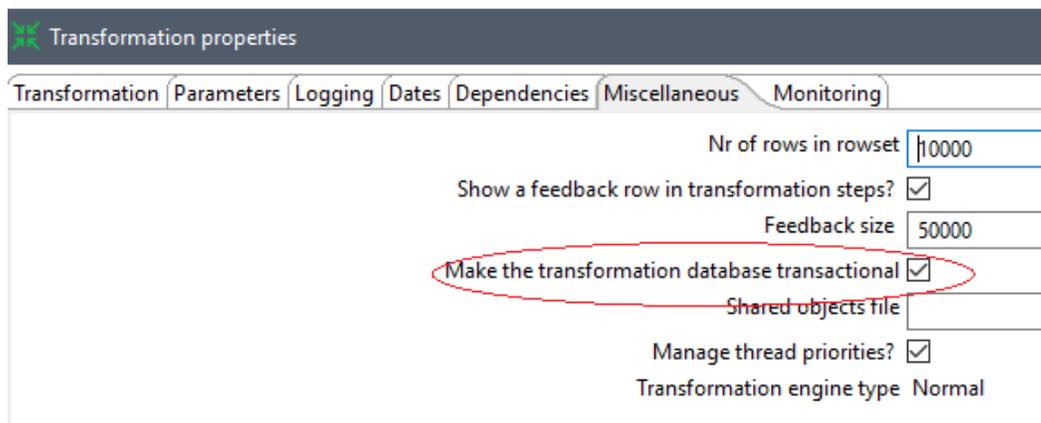

Figure 7: Make the Transformation Database Transactional

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- [Database Transactions in Jobs and Transformations](#)
- [Pentaho Components Reference](#)
- [Use Checkpoints to Restart Jobs](#)

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Item	Response	Comments
Did you consider which restartability option was best for your situation?	YES_____ NO_____	
If you chose checkpoints, did you configure them according to [Use Checkpoints to Restart Jobs](#)?	YES_____ NO_____	
If you chose a control table, did you configure your job according to the directions in the [Control Table](#) section?	YES_____ NO_____	
If you chose to make your job/transformation transactional, did you check the box on the job/transformation properties?	YES_____ NO_____	