Best Practice - Dividing Large PDI Repositories

This page intentionally left blank.

Contents

This page intentionally left blank.

Overview

Many customers initially create a single Pentaho Data Integration (PDI) repository to maintain multiple environments – e.g., development, quality assurance, stage, production – when first installing PDI. These customers then divide this single repository into different environments using nested folders.

Over time, this single repository may grow to a size that negatively impacts performance. Customers may also find that management of a single repository is cumbersome, even if all environments are non-production. This document addresses these problems by detailing an automated method to improve performance by segmenting a single repository into several smaller repositories using PDI.

Keep these Pentaho Architecture principles in mind while you are working through this document:

1. Architecture is important, above all else.
2. Platforms are always evolving: sometimes you will have to think creatively.

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

Software	Version
Pentaho	PDI 6.x, 7.x

Options to Divide the Repository

When a customer is faced with the need to separate a single, large repository into several repositories, they have some different options:

1. Export the entire repository to a file using Spoon or the CLI, and then re-import that repository into all of the new target environments using the same method.
   Once reimported into the new environments, repository folders that are not needed in a given environment have to be manually deleted. This may be a viable option when the number of repository folders is small; however, with very large repositories, this can be a very time-consuming, manually intensive, and tedious process.
2. Export repository folders individually using Spoon, and then import these folders individually into the appropriate new target repository.
   Similar to the previous option, this approach can be time-consuming with large repositories.
3. Use PDI objects to selectively export and import folders.
   This automated approach is very efficient, and repeatable should the large source repository change.

The following sections will detail the necessary steps required to implement the third option.

Use PDI to Divide the Repository

There are a few steps required in order to use PDI to divide your large repository. Before you begin, make sure to create a backup of your PDI repository.

This document uses examples based on sample files to help illustrate the processes. These samples, while not supported by Pentaho, can serve as a template for you to use in your environment. Such efforts should be validated in a test environment prior to advancing to a production environment.

First, you will need to segment the repository folders; then export those folders to files; and finally, import those files to a new repository.

Segmenting the Repository Folders

A process to match existing folders to new repositories must be completed, regardless of which option is chosen to divide the repository. Using an automated approach, this information is listed in a file.

For example, a file named `repository_folder_list.txt` can be used that includes a single field, `folder_name`. This field includes each repository folder that you want to move to a new repository. The example text below would move three dev folders from a single repository to a new development repository:

```
folder_name
/home/dev/application1/
/home/dev/application2/
/home/app3/
```

Exporting the Repository Folders to Files

Once the list of repository folders has been created, the repository folders can be automatically exported to XML files and folders in a file system. PDI includes an **Export Repository to XML File** job-entry specifically designed for this task.

Figure 1 - Export Repository to XML File Job-Entry shows a fully-parameterized **Export Repository to XML File** job-entry. As currently configured, it will create one set of export folder(s) and an XML file for every repository folder defined in the `repository_folder_list.txt` file.

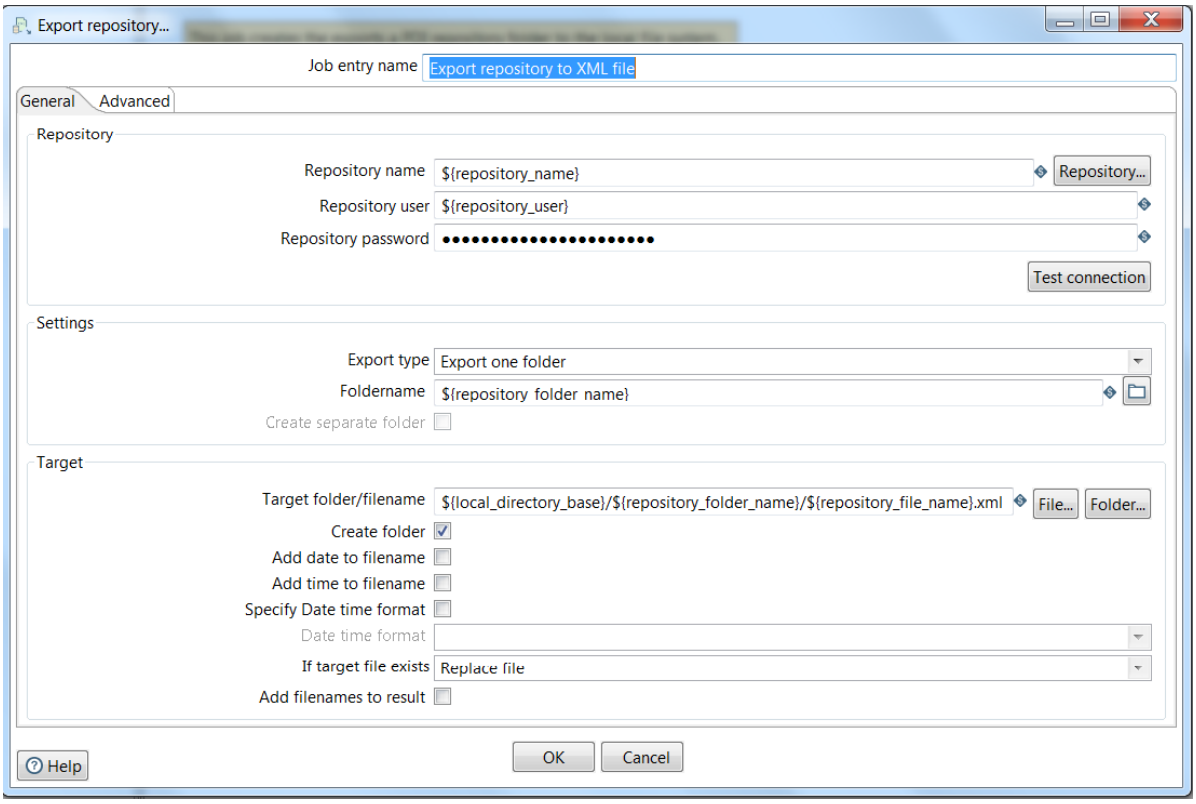

Figure 1 -Export Repository to XML File Job-Entry

In order to fully automate the process of creating export files for all of the folders listed in `repository_folder_list.txt` file, a few jobs and transformations must be built around the **Export Repository to XML File** job-entry to perform the following tasks:

1. Check for a `repository_folder_list.txt` file. If it exists, then read the list of PDI repository folders to be exported.
2. For each folder listed in the file, export the repository folder to the local file system using the **Export Repository to XML File** job-entry.

In order by make the export process flexible, the following five parameters must be provided:

Parameter	Definition
`file_name`	The file name for the file that lists the repository folders to be exported.
`local_directory_base`	The local directory where the repository folder files will be written.
`repository_name`	The name of the source repository.
`repository_user`	The name of the source repository user.
`repository_password`	The password for the source repository user.

Import the Repository Folders to the New Repository

Once the repository folders have been exported from the source repository to the XML files on the file system, they can be automatically imported into the new repository. Note: this section assumes that new repositories have already been created in your new PDI environments.

PDI includes script files - `import.sh` for Linux, `import.bat` for Windows - specifically designed for this task.

Figure 2 -Execute Import.sh Using the Shell Job-Entry shows a fully-parameterized command that will import one repository folder for each entry in the `repository_folder_list.txt` file by calling `import.sh`:

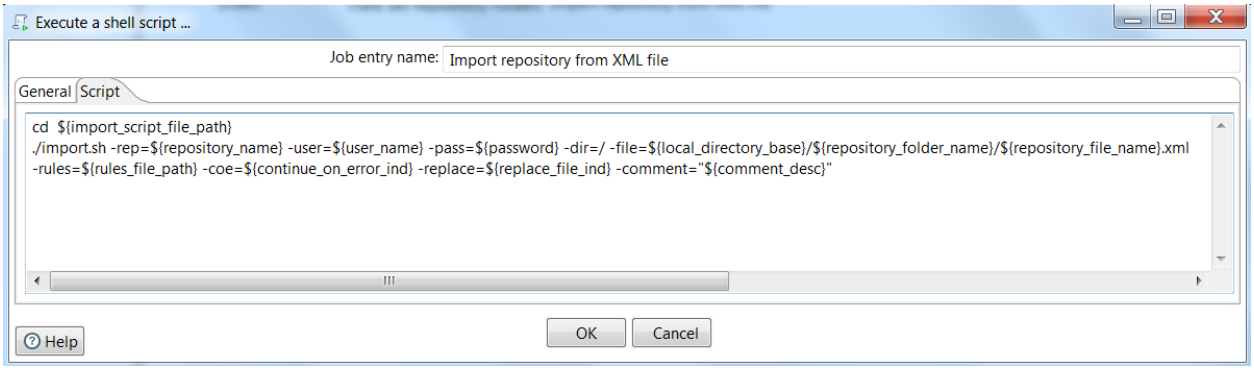

Figure 2 -Execute Import.sh Using the Shell Job-Entry

Here is the command used in the Shell job entry:

```
cd  ${import_script_file_path}

./import.sh -rep=${repository_name} -user=${user_name} -
pass=${password} -dir=/ -
file=${local_directory_base}/${repository_folder_name}/${reposit
ory_file_name}.xml -rules=${rules_file_path} -
coe=${continue_on_error_ind} -replace=${replace_file_ind} -
comment="${comment_desc}"
```

A few jobs and transformations must be built around the import script to perform the following tasks, in order to fully automate the process of importing all of the folders listed in `repository_folder_list.txt` file:

1. Check for a `repository_folder_list.txt` file.
    a. If it exists, then read the list of PDI repository folders to be imported.
2. For each folder listed in the file, import the repository folder from the local file into the target repository.

In order by make the import process flexible across environments, the following seven job parameters must be provided:

Parameter	Definition
file_name	The file name for the file that lists the repository folders to be imported. Use the same file used to create export list.
import_script_file_path	The path to the `import.sh` or `import.bat` file - does not include file name.
local_directory_base	The local directory where the repository folder files will be written.
password	The password for the `username` you specified with user.
repository_name	The name of the enterprise or database repository to import into.
rules_files_path	The path to the rules file, including full directory and file name.
user_name	The repository `username` you will use for authentication.

In addition, the following three optional job parameters may be helpful:

Parameter	Definition
comment_desc	The comment that will be set for the new revisions of the imported transformations and jobs.
continue_on_error_ind	Continue on error, ignoring all validation errors. Defaults to `false`.
replace_file_ind	Set to Y to replace existing transformations and jobs in the repository (creates a new version if versioning is turned on). Default value is N.

Finally, a `import-rules.xml` file must be created and placed in the path specified in the rules_file_path parameter.

Related Information

The following links provide additional information about the PDI job-entries mentioned in this document.

- Export Repository to XML job-entry
- More information about the Shell job-entry

Best Practice Check List

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 segment the PDI repository folders?	YES ___ NO ___	
Export the repository folders to files?	YES ___ NO ___	
Have you imported the repository folders to their new repositories?	YES ___ NO ___	