HITACHI
Inspire the Next

Pentaho Data Integration (PDI) Development Techniques

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

This page intentionally left blank.

Overview

This document is designed to give developers and administrators best practices around the setup and configuration of directories to be used for Pentaho Data Integration (PDI) development and execution.

Some of the things discussed here include folder structures for workstations and servers, configuration, and migrating content.

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, 8.0

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Directory and Folder Structures

The development and deployment of PDI-based solutions follows a convention that places all project-related development items into one folder. That folder and its content can be migrated from environment to environment, as appropriate.

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

Client/Workstation Folder Structure

To organize your client/workstation folder structure, set up one main folder for each project using PDI.

Each project and variant should have its own unique `KETTLE_HOME` variable that is not shared and is set for each execution, as described in the Kettle and Project Configuration section.

Each project should have the following folders:

Folder		Folder Definition
Content		Transformations and jobs will be stored in this folder. Root folders of `home` and `public` should always be used.
Config		Project properties and other configuration files.
Input		Files that are input to the solution.
Output		Files that are produced as part of this solution.
Env		Environment folder where each unique environment variation is stored. The following variants will help you create different folders within one project:
-	**Pentaho Version**	Each version should have its own folder.
-	**Content Storage**	If you are connecting to files or a repository.
-	**Server Environment**	Development, testing, and production should all have their own env variant.
-	**Container**	Where the variant is intended to execute (workstation, Carte, or DI server).

Here is an example folder structure for a project called Project1:

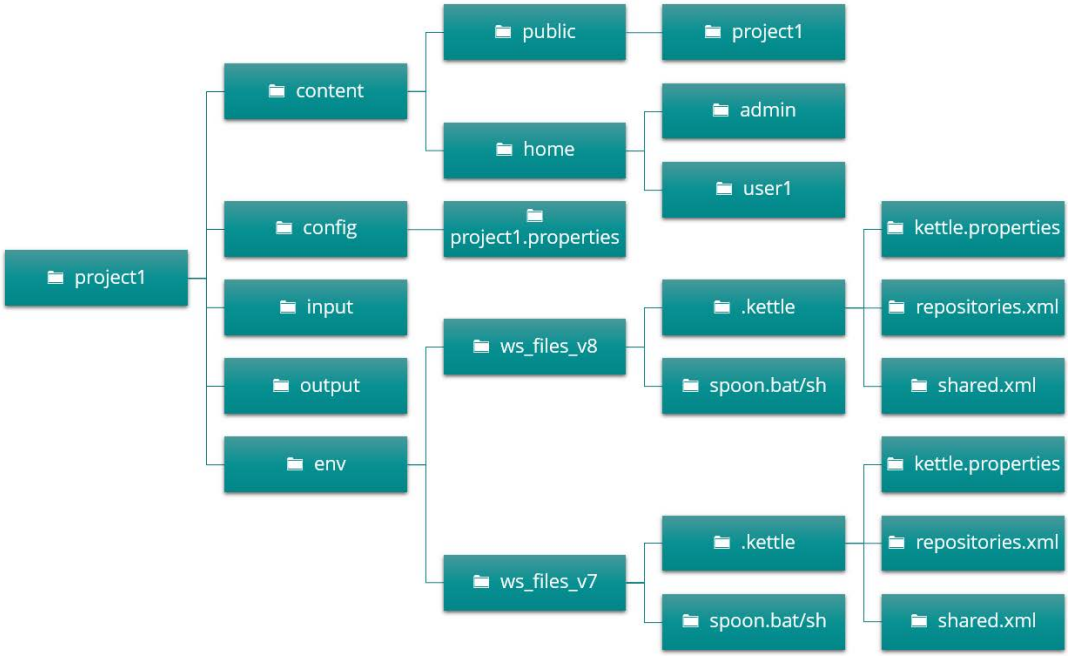

Figure 1: Project1 Folder Structure

Here is a text representation of the same Project1 folder structure:

```
project1
-content
--public
---project1
--home
---admin
---user1

-config
--project1.properties

-input
-output

-env
--ws_files_v7
---.kettle
----kettle.properties
----repositories.xml
----shared.xml
---spoon.bat/sh

--ws_files_v6
---.kettle
----kettle.properties
----repositories.xml
----shared.xml
---spoon.bat/sh
```

Server Folder Structure

The server running Pentaho and supporting multiple projects at once within one Java Virtual Machine (JVM) should be set up in an orderly fashion with several directories, including one for logs.

Note that logs are collected centrally for ease of monitoring, archive, and maintenance. Parameters, inputs, and outputs are all collected per project.

Here is a text representation of the structure for a server install (`/opt/pentaho`):

```
server
-data-integration-server or -pentaho-server (if using Pentaho 7.x +)
--pentaho-solutions
---system
----slave-config.xml
-data-integration

User Home Directory (use script in the env folder)
Nothing is actually here (bad practice)

Project Directory (/projects/project1)
-content
-config
-input
-output
-env

Central Log Directory (/projects/log)
-project1
--job1datetime.log
--job2datetime.log
-project2
--job3datetime.log
--job4datetime.log
```

Here are visual representations of those directories:

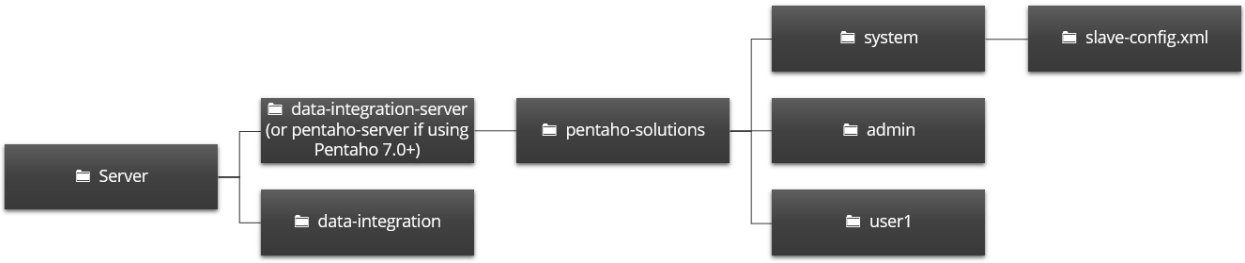

Figure 2: Serve

r

Figure 3: User Home Directory

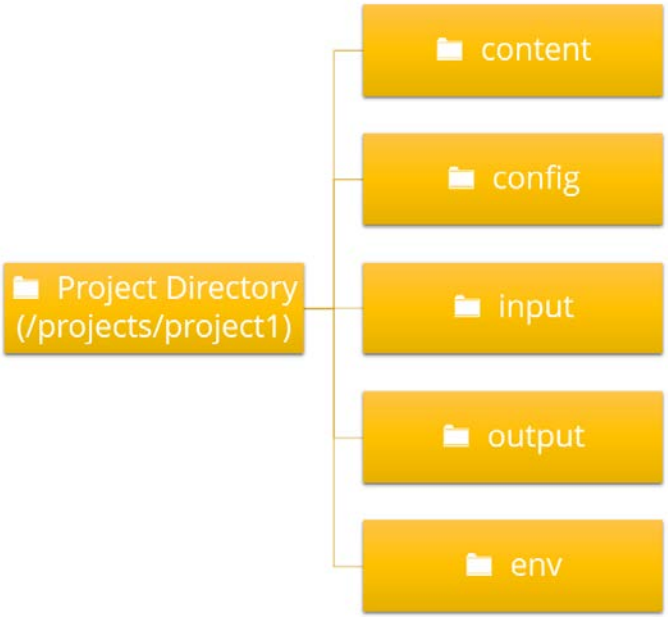

Figure 4: Project Directory

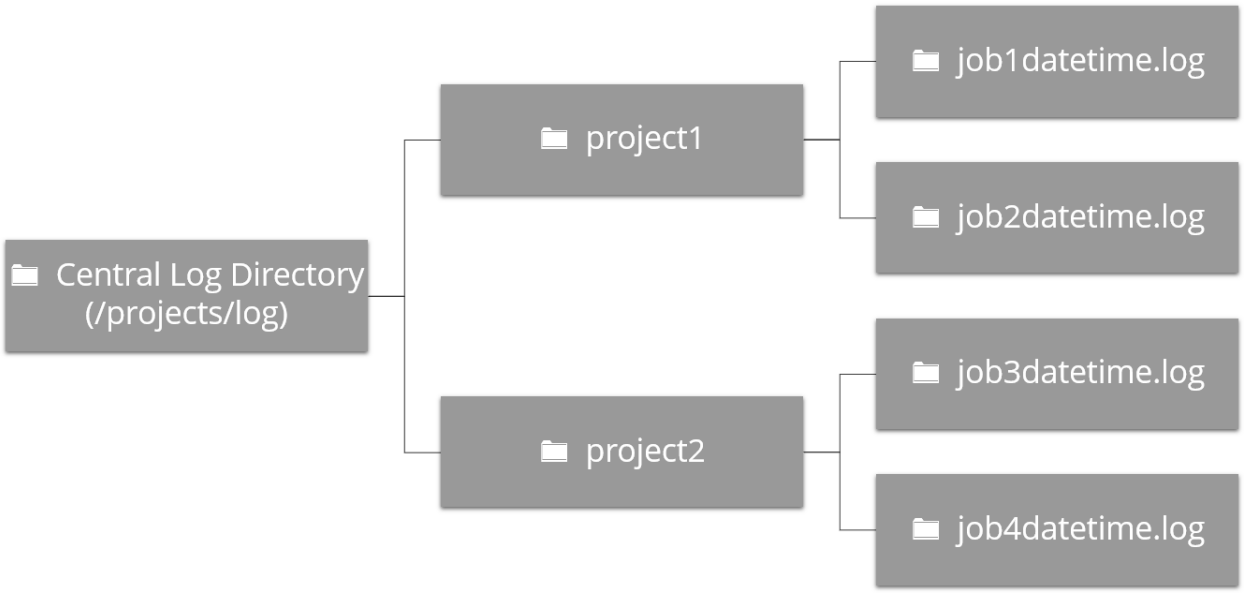

Figure 5: Central Log Directory

Development for Your Project

When developing transformations for `project1`, run the `bat/sh` script that is unique to the type of connection that you have (local, development, production, and so forth).

Each connection should launch Spoon in its own Kettle home. Setting `KETTLE_HOME` this way allows all connection-specific information to be contained in this `KETTLE_HOME` directory.

Use a separate `KETTLE_HOME` with different configurations when you switch to talking to each repository.

Here is a sample `spoon.bat` in `./projects/project1/env/ws_files_v51`:

```
set KETTLE_HOME=U:/projects/project1/env_ws_files_v51
call U:\myfiles\pdi-ee-client-5.1\data-integration\Spoon.bat
```

Kettle and Project Configuration

The configuration for the Kettle properties only needs to be migrated for the first time a server is stood up. Each time you add a new project, or make modifications, you must synchronize the `project.properties` file with the server.

When migrating a development properties file to the server, you must replace the development references with non-development references wherever necessary.

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

Kettle Properties

Each JVM that runs PDI should source the default Kettle properties for that environment. Each `kettle.properties` should define these minimum global settings.

Additional project-specific settings should be in the `project.properties` files, not the global `kettle.properties` file.

```
KETTLE_CHANNEL_LOG_SCHEMA=
KETTLE_CHANNEL_LOG_DB=
KETTLE_CHANNEL_LOG_TABLE=
KETTLE_JOB_LOG_DB=
KETTLE_JOB_LOG_SCHEMA=
KETTLE_JOB_LOG_TABLE=
KETTLE_JOBENTRY_LOG_SCHEMA=
KETTLE_JOBENTRY_LOG_DB=
KETTLE_JOBENTRY_LOG_TABLE=
KETTLE_TRANS_LOG_SCHEMA=
KETTLE_TRANS_LOG_DB=
KETTLE_TRANS_LOG_TABLE=
KETTLE_STEP_LOG_SCHEMA=
KETTLE_STEP_LOG_DB=
KETTLE_STEP_LOG_TABLE=
KETTLE_TRANS_PERFORMANCE_LOG_DB=
KETTLE_TRANS_PERFORMANCE_LOG_SCHEMA=
KETTLE_TRANS_PERFORMANCE_LOG_TABLE=

KETTLE_REDIRECT_STDERR=Y
KETTLE_REDIRECT_STDOUT=Y

PROJECT_DIR=\projects
```

Project Properties

Each project running under PDI should have project-specific settings in the `project.properties` files, not the global `kettle.properties` file.

Use a transformation to handle the `project.properties`.[1]

```
ActiveProject.Home=$PROJECT_DIR\$PROJECT_NAME
project1_target_hostname=192.168.1.1
project1_target_db=dbname
project1_target_user=admin
project1_target_pass=password
```

Content Migration Overview

There are generally two modes of operation to migrate content, depending on your environment and usage:

- **External repository** (SVN, Git, or others): With this method, you would use external tools and methods to migrate the content to the server. Typically, this involves a checkout of the project.
- **Pentaho repository** (enterprise, database, file): When you are migrating from a repository, you must export the content from the workstation and import it into the server repository.

Exporting Content

Three ways to export content are:

- **Graphical Export**: Using this, you can export the entire repository or individual folders.
- **Command Line**: Use a command line for the export step where many options are available. This is often used in script automation scenarios.
- **Job step**: There is a job step that can selectively export repository contents based on variables and parameters.

Import Content

Methods for importing content include:

- **Graphical Import**: Using this, you can import the contents of a prior repository export through the Spoon Repository menu. Always choose the `/` node for import to maintain correct directories.
- **Command Line**: Use a command line for the import step where many options are available. This is often used in script automation scenarios.

[1] Roland Bouman's blog post on Managing kettle job configurations has more details on working with `project.properties` files in Kettle.

Related Information

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

- Pentaho
  - Components Reference
  - Import and Export PDI Content
- Roland Bouman's Blog- Managing Kettle job configuration

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 set up the workstation folder structure?	YES_____ NO_____	
Did you set up the server folder structure?	YES_____ NO_____	
Did you configure the Kettle properties?	YES_____ NO_____	
Did you configure the project properties?	YES_____ NO_____	
Have you migrated your content?	YES_____ NO_____	