Component Based Development in Perforce

A Reference Implementation Tool Set

# Introduction

Component based development (CBD) adopts production line techniques for software development. A complex product may be composed of tens or hundreds of component building blocks. Each component may be developed by a specialized team, working on an independent iteration cycle. A product architect will define how the complete product is assembled from specific versions of the necessary components. The CDB process provides some insulation from the vagaries of scheduling such a complex effort, just as a car may be shipped with the latest stable version of a GPS navigation system in order to maintain the overall delivery schedule.

However, just as changing manufacturing specifications require close coordination of the production line, the components of a complex software product must be compatible at some interface level. Additionally, in many industries a finished product can be configured in specific ways for particular customers. In the cellular communication industry, a chip set may be tailored for different networks and carriers, resulting in a complex feature matrix.

This document describes a reference implementation of a tool set for managing CBD in Perforce.

# Introduction to the Tools

All scripts and configuration data are kept in a depot called cbd. In the distribution you will find all the pieces of the tool kit as they should be checked into the cbd depot.

├───applets

├───broker

├───configurations

├───log

├───sample\_depot

├───scripts

└───secure

## Installation

* Install and configure the Perforce broker using //cbd/broker/broker.conf as an example. You will need to put the shell scripts in //cbd/scripts under the broker working directory.
* On the broker machine, set the environment variable CBD\_HOME to point to a local copy of the cbd depot.
* Enable the spec depot.
* On the broker machine, set up a workspace called cbd\_manager that maps in the entire cbd depot. The broker will use this for applet-driven configuration updates.
* As a best practice, use the +l file type modifier on the entire cbd depot to prevent concurrent updates of configuration files and important scripts.
* Create a file //cbd/secure/login.cfg that has authentication credentials for the broker scripts. It should list the user name on the first line and the password on the second line. You can add any appropriate access control to this file so it is not visible to ordinary users.

# The CBD Data Model

The tool set uses the following model. A configuration is defined as a set of one or more components. Each component is specified using four pieces of information:

| Field | Purpose | Example Value |
| --- | --- | --- |
| Depot path | Locates the component in the Perforce repository. | //depot/comp/abc/main/… |
| Access | Describes how the component may be used in this configuration.   * Active components may be used freely. * Write components may be edited but will not accept merge changes or refactoring. * Read components may not be modified. * Binary components are non-source elements imported for building and testing. They may not be modified. | active |
| Client location | Where this component should exist under the workspace directory. | abc/… |
| View (optional) | A label that restricts the view of the component in the workspace. A label may contain a view and, for read or binary components, a revision specifier. | Comp.abc.l |

## Data Model Representation

A configuration is specified in an XML file. The example below shows a simple configuration with four components.

<configuration>

<component>

<depotPath>//depot/comp/abc/...</depotPath>

<access>active</access>

<clientLocation>abc/...</clientLocation>

</component>

<component>

<depotPath>//depot/comp/def/...</depotPath>

<access>write</access>

<clientLocation>def/...</clientLocation>

</component>

<component>

<depotPath>//depot/comp/ghi/...</depotPath>

<view>comp.ghi.l</view>

<access>read</access>

<clientLocation>ghi/...</clientLocation>

</component>

<component>

<depotPath>//depot/lib/xyz/...</depotPath>

<access>binary</access>

<clientLocation>lib/xyz/...</clientLocation>

</component>

</configuration>

## Data Model Storage

A depot called cbd is used to store all configuration data and tools for this package. Configuration files are stored under //cbd/configurations.

## Creating, Updating, and Deleting Configurations

The configurations can be modified using normal Perforce client software to add, edit, and delete the files. Alternatively, a P4V applet can be used to view, edit, and delete configurations.

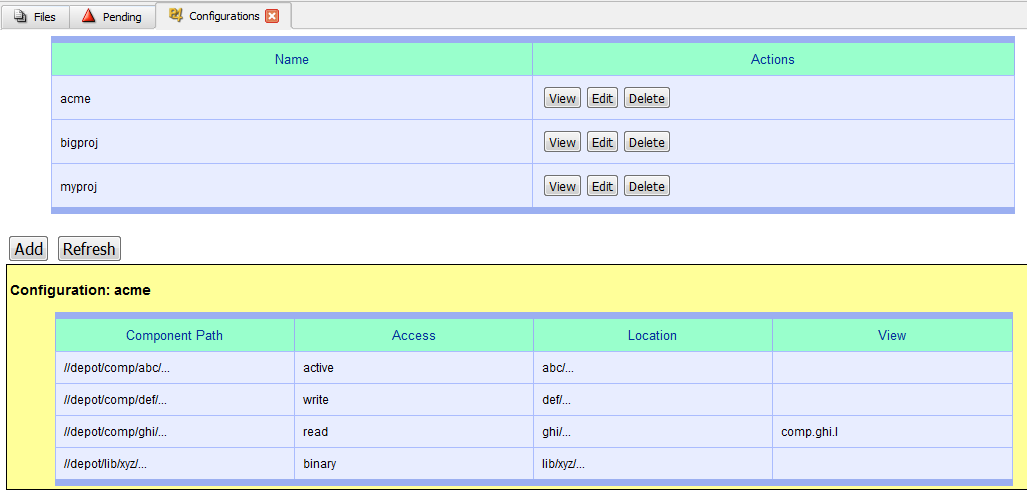


Figure 1: Configuration Applet

You can also view the producer-consumer relationships (i.e. which configurations use certain components).



Figure : Consumer Dependency List

To install this applet:

* Make a protections table entry (see //cbd/applets/protect.txt for an example).
* Enable applets in P4V for the broker you are using.

# Usage

This tool kit supports several operations.

## Creating a workspace tied to a configuration

Use the special broker command wscreate.

p4 wscreate my-config my-ws

This creates a workspace my-ws whose view is determined by the configuration my-config.

## Switching a workspace to a new configuration

Use the special broker command wsswitch.

p4 wsswitch other-config my-ws

This updates the workspace my-ws to the configuration other-config. The workspace view is adjusted accordingly.

## Populating a workspace

Use the normal p4 sync or p4 update commands with no arguments to populate the workspace correctly. If you run these commands with file arguments, the broker assumes that you are explicitly working outside the bounds of the configuration view. This choice can be changed with a minor adjustment to the scripts.

## Access control

The broker intercepts commands like p4 edit and p4 submit to make sure that you are not checking out or submitting files with read or binary access. Each argument to these commands is verified. You can adjust the broker scripts to suit your own workflow preferences and provide more or less guidance.

## Updating workspaces when a configuration changes

If you modify a configuration, each workspace tied to that configuration is updated on the next sync or update command. For example, if you use an updated release of a component (//depot/comp/abc/r2.0 instead of //depot/comp/abc/r1.5), a workspace using this configuration will have its view automatically updated the next time a sync or update command is run.

# The Nuts and Bolts

* The scripts use the spec depot to record the links between workspaces and configurations using attributes.
* The scripts use python logging. You can adjust the log configuration in the file //cbd/log/log.cfg.

# Known Limitations

* The p4 unshelve command is filtered, but optional file arguments are not considered.
* The reconcile and status command filtering may not correctly limit components that are scoped by a label.
* This tool set cannot be used with streams.
* Automatic labels cannot be used to specify a configuration.
* Static labels can only be usefully applied to components with read or binary access.
* Components only support a single depot path. Multiple depot paths would provide more flexibility.
* The configuration applet does only basic error checking and could provide additional assistance like label look-up.