

Perforce + Java: The Best Of Both Worlds

Hamish Reid and Kevin Sawicki, Perforce Software Inc.

Abstract

Perforce's Java products bring a Java-natural approach to both traditional and agile development environments. In particular, Perforce's new re-architected version of the P4WSAD plug-in for Eclipse integrates Perforce's excellent software configuration management features into the Eclipse environment in a way that feels natural to both Eclipse and Perforce users, and that plays on the strengths of each. A crucial part of the new product line is a new Java native API – the P4Java API – that allows users and developers to access Perforce from within Java apps and plug-ins in Java-natural ways, and to smoothly integrate Perforce functionality into tools such as cruise control, maven, and ant, and environments such as the J2EE web applications and web services frameworks. This talk will present Perforce's new Java product line and the thinking behind it, and illustrate them with real-world demonstrations of the Perforce products at work. In particular, the talk will focus on examples of the Perforce + Java combination's powerful support for agile techniques such as continuous integration and refactoring, and applications such as Perforce-backed web services.

The Vision Thing

Here's a snapshot from the near future: you're a development manager in charge of several Java teams developing products ranging from standalone Java apps to J2EE web apps to design metric plug-ins for third-party Eclipse developers. How will Perforce fit into or help you with all this? Put aside for a moment the usual more abstract Perforce strengths – smart merges, atomic changelists, visual tools, etc. – and concentrate on some concrete here-and-nows: your workflow is generally Agile, involving a great deal of continuous integration, constant refactoring, a test-driven development methodology, and a lot of codeline branching and subsequent merging. You use tools like ant, cruise control, junit, maven, Bugzilla, wikimedia, etc., to implement this. How does Perforce fit into this? *Seamlessly* – by providing the essential underlying software configuration management (SCM) bedrock to these tools and your developers in a way that your developers don't even have to know they're using Perforce, just that they're using an Agile workflow or a set of standard tools, and that Perforce "just fits in" to this environment without them having to do anything special.

Similarly for your developers using the Eclipse IDE to develop Java apps and plug-ins: Perforce is there, supplying the essential services and features to the IDE, but your developers don't so much see "Perforce" as they see the usual familiar panoply of Eclipse team provider features and perspectives, crucially augmented by valuable tools like dynamic branch integration timeline graphs and visual merge tools, closely integrated into the Eclipse look and feel. Your developers access Perforce through Eclipse – and get the full benefits of Perforce's SCM features and safeguards – but to most of them they're just accessing an extended Eclipse team menu from a Java perspective, and Perforce is the invisible backbone allowing it all to happen.

As for your app and plug-in developers, some of them need a way for the apps and plug-ins they're developing to consume and deliver Perforce SCM services when deployed. How does Perforce fit in here? Again, seamlessly – you could have the teams issue conventional p4 commands and interpret the answers, or you could use the P4Java API to present Perforce SCM services in a Java-natural way as Java methods, classes, objects, and interfaces, deployable anywhere there's a standard Java environment.

That's the vision. How do we get there from here?

Perforce's Java Products: Aims & Scope

Perforce's Java products have one overriding aim: *transparent integration*. What does this really mean in terms of Java and Perforce? Basically, two things:

1. That from a typical Java developer or team leader's perspective, using Perforce's SCM features effectively from within a Java development environment or Java app should require nothing special (or worse, "*Interesting*"...); and,

2. From an IT person's perspective, supporting Perforce in a Java environment should not require anything above and beyond normal Perforce support, or require any difficult Perforce-specific changes to existing Java environments.

Ideally, in the longer term what this means is that the typical Java developer should not necessarily even be aware that they're using Perforce (unless they want to be), and that that typical Java developer never needs to step outside the "Java bubble" to use Perforce effectively on a daily or weekly basis. Similarly, in the longer term, IT staff supporting Java developers and users with Perforce should barely notice that they're supporting Java development environments and staff as opposed to developers and development environments in general. In short, let the Java developers concentrate on Java development, and let the IT folks concentrate on Perforce support.

What does this take? We believe three integration points are crucial here:

- **Language integration** – our Java products must be "Java natural". They must conform to accepted Java norms and standards for things like deployment, documentation, standard package usage, design and use patterns, etc., and they shouldn't look like something grafted on from another development language or ecosystem.
- **Tools integration** – our Java products must work well with standard and common third party Java development and app tools (and increase the utility of both those tools and Perforce itself), and they must speak the same conceptual language as their users and the tools themselves.
- **Workflow integration** – our Java products must fit seamlessly into and support the typical workflows and development lifecycle models encountered and used in the Java world. It's true that little in this area is necessarily Java-specific, but unless our products support things like continuous integration, large-scale refactoring, or agile workflows without fuss and bother, they're not going to be as useful as they could be.

In the section that follows and the associated live examples, we'll highlight these integration points in practical detail.

The Perforce Java Product Line

Perforce's current and future Java products fall into the following broad product lines:

- **APIs and bindings:** Perforce's product line is built on top of the new P4Java API, a Java-native and Java-natural presentation of Perforce's features and services in Java. The P4Java API can itself be presented in different bindings on top of the underlying API; in particular, Perforce Ajax and web services bindings and deployments are possible for the future, and are in any case easily generated by third-party tools.
- **Tools and application plug-ins:** Perforce's flagship plug-in product, P4WSAD for Eclipse, has been entirely reworked and re-implemented on top of P4Java, and now closely integrates with Eclipse and Eclipse workflows. In addition, in the longer term Perforce will be providing or evaluating plug-ins for ant, cruise control, maven, etc., to access Perforce services from within these tools in "tool-natural" ways.
- **The Perforce Java Developer's Portal:** in addition to the various APIs and plug-ins themselves, a critical part of the longer-term Java strategy is likely to be the establishment of a Perforce Java developer's portal where worked examples of Perforce Java product usage and deployment will be presented along with tutorials, full documentation, and the latest releases of Perforce and partner plug-ins and apps (note that the details here are currently subject to many caveats and future contingencies).

The following sections concentrate on the most important parts of these product lines, P4Java and P4WSAD; examples of usage and features will be given live during the presentation. It should be noted in passing that the Perforce Java development team currently uses all our Java products on a daily basis, both to develop newer versions, and as part of an agile test and development workflow involving things like cruise control, Eclipse, junit, ant, etc. in conjunction with Perforce.

The P4Java API

The new P4Java API is aimed at Java plug-in and app developers who need to access Perforce SCM features

and services directly in Java-natural ways. The P4Java API presents Perforce functionality and associated objects (like files, versions, jobs, changelists, etc.) as Java interfaces, classes, and methods, and allows virtually any recent Java SE or J2EE application or plug-in to access Perforce services from within that app without leaving the Java world. P4Java now forms the basis for the new P4WSAD Eclipse plug-in, and is intended to be the heart of future ant, cruise control, maven, etc. plug-ins as well as the basis for client-side Ajax or J2EE server-side web services (etc.) bindings.

P4Java is not a repackaging of the current Perforce C++ API or the P4 command line interpreter; it is a completely new API written from the ground-up in Java, and departs from the C++ API's usage model considerably by (for example) presenting things like (say) a list of Perforce-managed files as a Java List of Java interfaces onto the underlying file data and operations. Similarly for things like changelists, which in P4Java are represented by a P4JChangeList interface onto the corresponding "real" changelist, meaning changelist elements and operations are accessible as first-class Java objects with predictable Java behaviors and usage patterns.

P4Java explicitly separates the underlying protocol implementation from the upper-level service and object presentation, allowing for several different implementations under the hood, including a fully-native Perforce RPC protocol implementation for use anywhere the standard JDK 5 network and file functionality is available, and a version built on top of the p4 command line interpreter for other environments.

The P4WSAD Eclipse plug-in

The new P4WSAD Eclipse plug-in has been redesigned and re-implemented from the ground up on top of the new P4Java API, giving us a more extensible and flexible platform for future development and deployment, and increasing the scope for much closer integration with the Eclipse application and plug-in model. In particular, we've gone to great lengths to make P4WSAD look and feel like a first-class Eclipse team player, while letting P4WSAD users also access extended Perforce functionality in ways that should feel natural to most Eclipse team functionality users.

For example, the new P4WSAD sync pane integrates the Perforce sync features into the well-tested and ubiquitous Eclipse team sync model, making the Perforce sync operations graphically available in ways that would be familiar to any Eclipse SVN or CVS user, but that allow for Perforce-specific extensions with the click of a mouse button or the selection of a menu.

Similarly for complex refactoring – while P4WSAD already supports refactoring according to Eclipse team infrastructure and plug-in rules, P4WSAD's new architecture allows us to intelligently optimize our responses to the underlying Eclipse operations and to implement things like extended cross-project and cross-server refactoring with the new Perforce move feature.

The primary aim here is to make the P4WSAD plug-in not just a natural part of the Eclipse ecology, but to make it powerful enough that the typical Java developer does not need to use any other Perforce tool on a daily or weekly basis. This doesn't mean that P4WSAD will be able to do *everything* – it's not likely ever to be able to do most admin tasks, for example – but it does mean that features available to Perforce users in other normal development contexts, such as smart merges and resolves, visual timelines and branching relationships, cross-provider jobs and bugfix support, etc., will also be available through P4WSAD.

A secondary aim is to make the Eclipse + P4WSAD combination suitable for integration into other products as-is, or as a stable platform for third-party development. For example, P4WSAD itself is now easily extendable using standard Eclipse plug-in extension points; future versions will extend this extensibility to support Perforce-specific and other extension points for things like pre- and post-submit processing, etc., with P4Java support built-in.