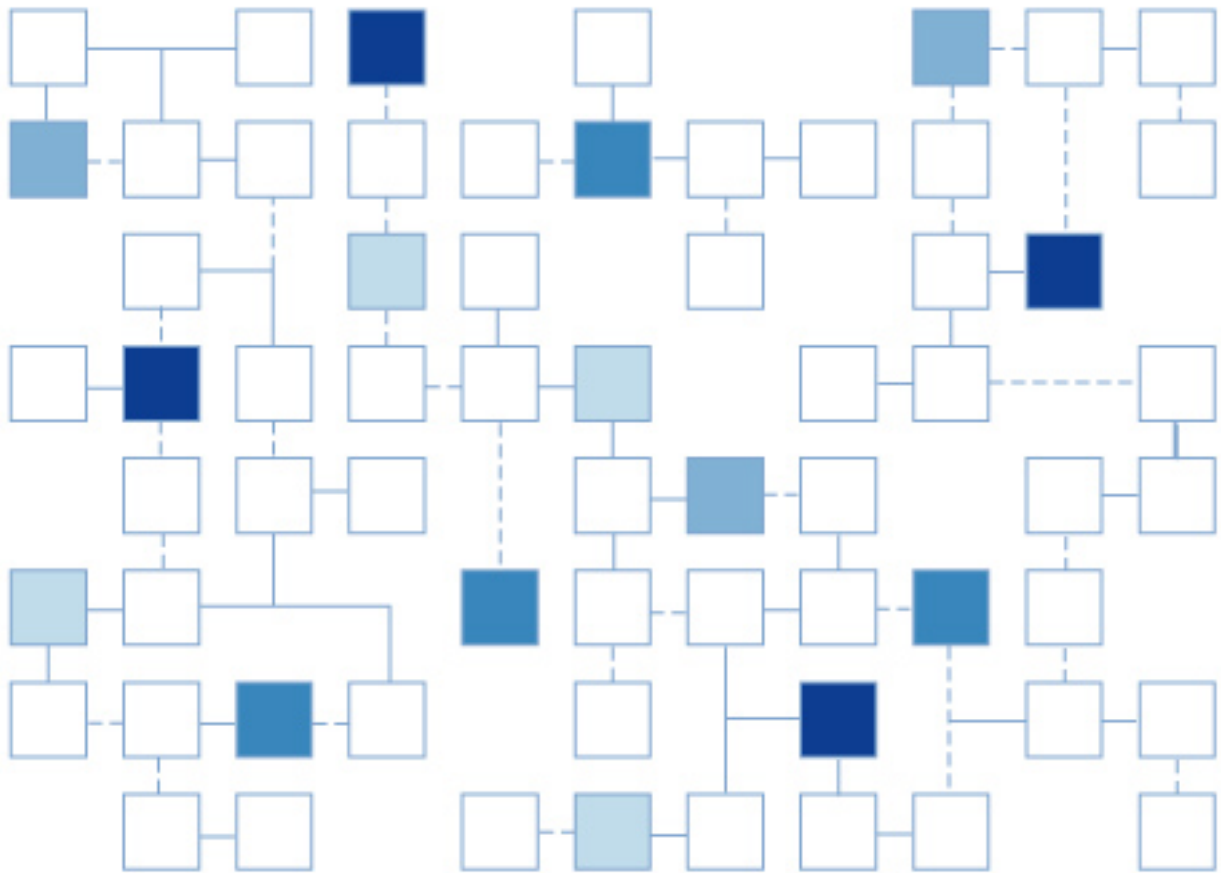




ExtraView – Perforce Integration Guide

Version 2.1





ExtraView Corporation
269 Mount Hermon Road, Suite 100
Scotts Valley, CA 95066

Telephone: (831) 461-7100
Fax: (831) 461-7104
Email: info@extraview.com
www.extraview.com
© 2004 - 2008 ExtraView Corporation
All rights reserved

Manual Name: ExtraView – Perforce Integration Guide
Revision Date: June 26, 2008

Perforce is a trademark of Perforce Software Inc.

Information contained in this document, and the software to which it refers is subject to change without notice. This includes URL's and any other web sites that may be mentioned. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise) or for any purpose, without the express written permission of ExtraView Corporation.

ExtraView Corporation may have patents, patent applications, trademarks, trademarks applied for, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from ExtraView Corporation, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

ABOUT THIS GUIDE

The ExtraView – Perforce Integration Guide is designed to give users of the ExtraView Web-based issue tracking and workflow management system the knowledge and proficiency needed to integrate ExtraView with the Perforce software configuration management system.

In writing this guide, we anticipate that the reader is at least somewhat familiar with many of the standard issue tracking functions found in ExtraView. We further expect that the reader is fully familiar with the set up, configuration and operation of Perforce.

Other ExtraView Guides available from ExtraView Corporation:

- ExtraView User's Guide
- ExtraView Administration Guide
- ExtraView CLI & API Guide
- ExtraView Installation Guide

Notational Conventions

This guide follows certain notational conventions that are explained below:

- Terminology that Administrators can customize in ExtraView will be marked in *italics*:

Select a *Product* from the list.

- Names of buttons, links, lists, or fields will appear in **bold**:

Select a value from the **Owner** dropdown list.

- Whenever there are multiple steps involved in achieving a certain result, these will be noted numerically:

1. Click the **Edit** button.
2. Select a value from the list.
3. Click the **Update** button.

Table of Contents

| | |
|---|-----------|
| INTRODUCTION..... | 6 |
| EXTRAVIEW PRODUCT DESCRIPTION | 6 |
| EVP4DJ..... | 6 |
| INTEGRATION CAPABILITIES | 7 |
| SAMPLE USE CASES | 9 |
| USE CASE 1 | 9 |
| USE CASE 2..... | 10 |
| USE CASE 3..... | 12 |
| INSTALLATION..... | 14 |
| INSTALL PERFORCE..... | 14 |
| INSTALL JAVA..... | 14 |
| ENVIRONMENT VARIABLES..... | 14 |
| INSTALL THE EVP4DJ FILES | 14 |
| INSTALLING EVP4DJ AS A WINDOWS SERVICE..... | 15 |
| CONFIGURATION..... | 16 |
| PERFORCE CONFIGURATION..... | 16 |
| <i>Create Client and User</i> | <i>16</i> |
| <i>Edit Jobspec</i> | <i>16</i> |
| EVP4DJ CONFIGURATION | 17 |
| <i>Create ExtraView EVP4DJ User.....</i> | <i>17</i> |
| <i>Create ExtraView Job Name Field.....</i> | <i>17</i> |
| <i>Modify ExtraView Layouts.....</i> | <i>17</i> |
| <i>Set Security Permissions for the Mapped Fields</i> | <i>18</i> |
| <i>Set Behavior Setting for Repeating Rows.....</i> | <i>18</i> |
| <i>Set Status Change Rules.....</i> | <i>18</i> |
| <i>ExtraView – Perforce Field Mapping.....</i> | <i>18</i> |
| <i>Fields with Special Treatment.....</i> | <i>24</i> |
| INTEGRATING WITH MULTIPLE PERFORCE SERVERS | 25 |
| RUNNING EVP4DJ..... | 27 |
| SUMMARY OF STEPS TO START EVP4DJ | 27 |
| START PERFORCE SERVER | 27 |
| START EVP4DJ | 27 |
| STATE FILE | 27 |
| ERROR LOG | 28 |
| APPENDIX A | 29 |
| COMMANDTXNPOLLER | 29 |
| COMMAND LINE OPTIONS FOR COMMANDTXNPOLLER | 29 |
| EVP4COMMANDLINE | 29 |

| | |
|--|-----------|
| <i>START Command</i> | 30 |
| <i>STOP Command</i> | 30 |
| <i>STATUS Command</i> | 30 |
| <i>QUIT Command</i> | 30 |
| APPENDIX B – P4 PROPERTIES FILE CONTENTS | 32 |
| <i>DEFAULT_LOG_LEVEL Values</i> | 36 |
| APPENDIX C – ARITHMETIC/BOOLEAN EXPRESSIONS | 37 |
| RULES OF EVALUATION | 37 |
| EXPRESSION EXAMPLES | 38 |
| APPENDIX D – REGULAR EXPRESSIONS | 39 |
| INDEX | 46 |

Introduction

ExtraView Product Description

ExtraView is a Web-based issue tracking system that is designed to meet the following objectives:

- Easy to install, configure and administer, minimizing your organization's setup and ongoing cost of ownership
- Provide functionality that is easily extensible over time
- Able to support your processes and your workflow, without major modification
- Scalable to support large numbers of users and issues
- Easily customized to reflect your company's terminology, and data hierarchies, and able to provide extensive validation for data that describes your organization, products, and services.

EVP4DJ

EVP4DJ is the name given to the ExtraView – Perforce Daemon. The J stands for the Java version. From Version 2.0, the daemon is written in Java, as opposed to the 1.0 release which was written in Perl.

ExtraView together with EVP4DJ has significant advantages over alternative methods of integrating Perforce with a bug or defect tracking tool. The key features and advantages of the EVP4DJ solution are:

- The integration is transparent in use to all users
- The integration is totally configurable. Like all other aspects of ExtraView, there is no set process or workflow or set of fields
- The integration can be configured to map multiple Perforce jobs to a single issue in ExtraView. This allows an implementation where multiple branches of code can all be tracked within a single issue
- The integration can be configured to map multiple ExtraView issues to a single job in Perforce
- A single instance of ExtraView may be configured to communicate seamlessly with multiple Perforce repositories
- There is total redundancy of transactions between ExtraView and Perforce. The integration does not depend on both the ExtraView and the Perforce servers being online continuously. The failure of one server will not jeopardize the installation, and when a server is restarted, all pending transactions will be executed
- The integration handles all flavors of the Perforce product – the P4V, P4 Win, P4 Web and command line interfaces are all equally

supported.

Integration Capabilities

Like all ExtraView features, integrating ExtraView with Perforce is configurable, and there is no fixed method for providing integration between the two products. ExtraView Corporation provides a sample integration with the code for the integration, based upon a summary of best practices seen across multiple customers. However, it is simple to adapt this best practice and adopt your own methods.

The following is a summary of how the integration may be used:

| ExtraView Activity | Effect |
|--------------------|--|
| Add Issue | Creation of new issues in ExtraView may trigger the creation of new Perforce jobs. Perforce job creation can be conditional upon the selection of one or more configurable field values in ExtraView. Alternatively, new ExtraView issues may be linked to existing jobs in Perforce. |
| Edit Issue | When issues in ExtraView are updated, the corresponding Perforce job (or jobs) is also updated, e.g. values entered in mapped fields are replicated in the Perforce job. |
| Close Issue | When the status of an ExtraView issue is changed to Closed, EVP4DJ can be easily configured to update the status of the corresponding Perforce job. This is an example of the use of the replication facilities rather than a standalone feature. |
| Reopen Issue | When the status of an ExtraView issue is changed from Closed back to Open, EVP4DJ can be easily configured to update the status of the corresponding Perforce job. This is an example of the use of the replication facilities rather than a standalone feature. |
| Change Status | Since submission of an associated changelist changes the status of a Perforce job to Closed, EVP4DJ captures ExtraView status other than Open and Closed by utilizing an ExtraView-specific Perforce jobspec field called EVstatus. Whenever the status of an ExtraView issue is updated to a new value, the Perforce job EVstatus field is updated to reflect the current ExtraView status. Note that EVStatus has no special meaning, it is simply a field that can be replicated. The field may be any available field. |

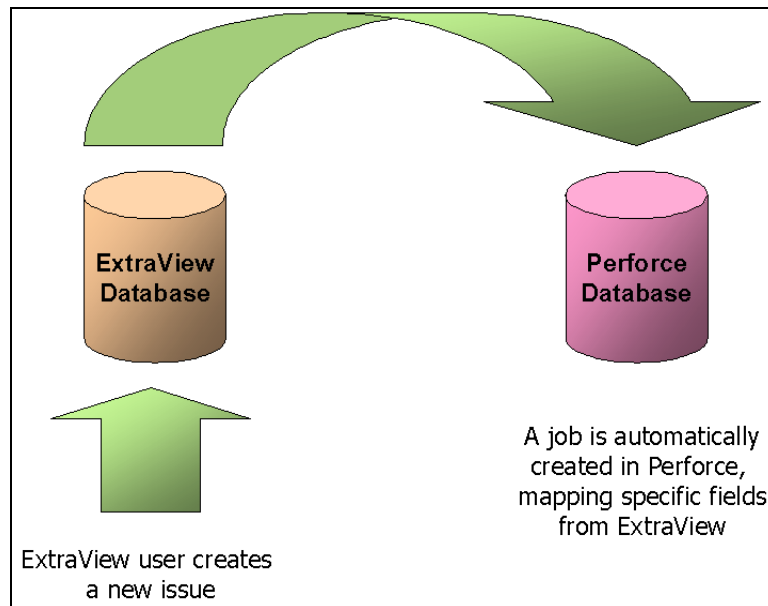
| Perforce Activity | Effect |
|-------------------|---|
| Create / Edit Job | When a job is created or edited in Perforce, it may be mapped to an existing issue in ExtraView, or a corresponding issue is automatically created in ExtraView if none already exists. If a new issue is created, the Perforce Job is subsequently updated to reflect the issue ID in ExtraView. |
| Submit Changelist | When a changelist is submitted that fixes one or more jobs, the corresponding ExtraView issues may be updated with the job status, the changelist description, other fields and resulting diffs. Diffs may be uploaded to ExtraView as an attachment to the affected issues or as comments. All field mappings are configurable, according to your needs. |
| Close Job | When a job is Closed in Perforce, EVP4D can be easily configured to update the status of the corresponding ExtraView issue. |
| Open Job | When a job status is set to Open in Perforce, EVP4D can be easily configured to update the status of the corresponding ExtraView issue. |
| Update EVstatus | When the EVstatus field of a Perforce job is edited to reflect another mapped status in ExtraView, the ExtraView issue status is updated accordingly. This is an example of the use of the replication facilities, rather than a standalone feature. |

Sample Use Cases

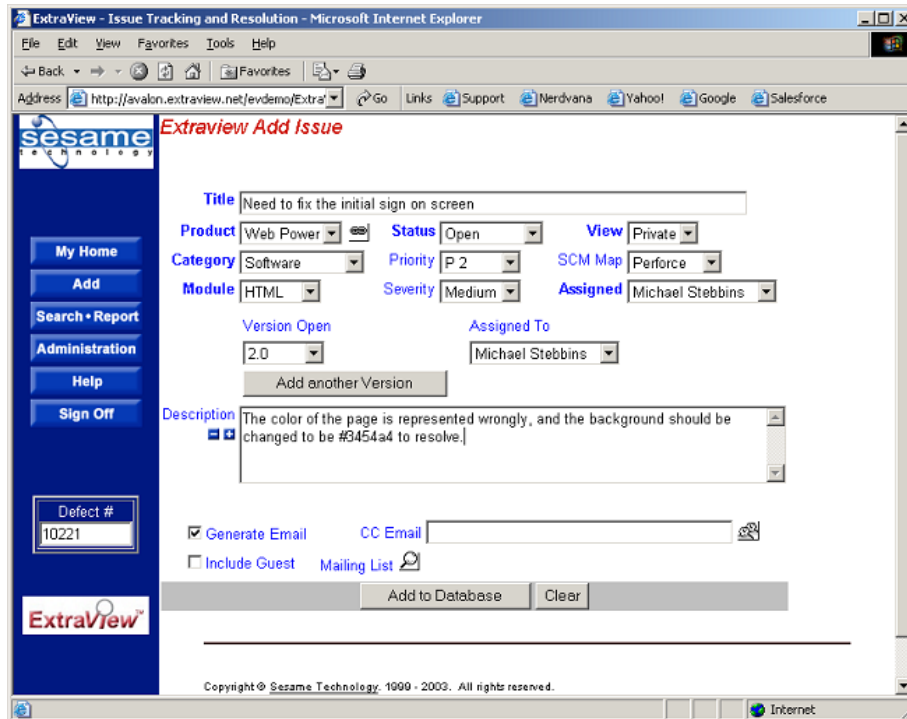
These use cases show how ExtraView integrates with Perforce from a logical perspective from a user's point of view.

Use Case 1

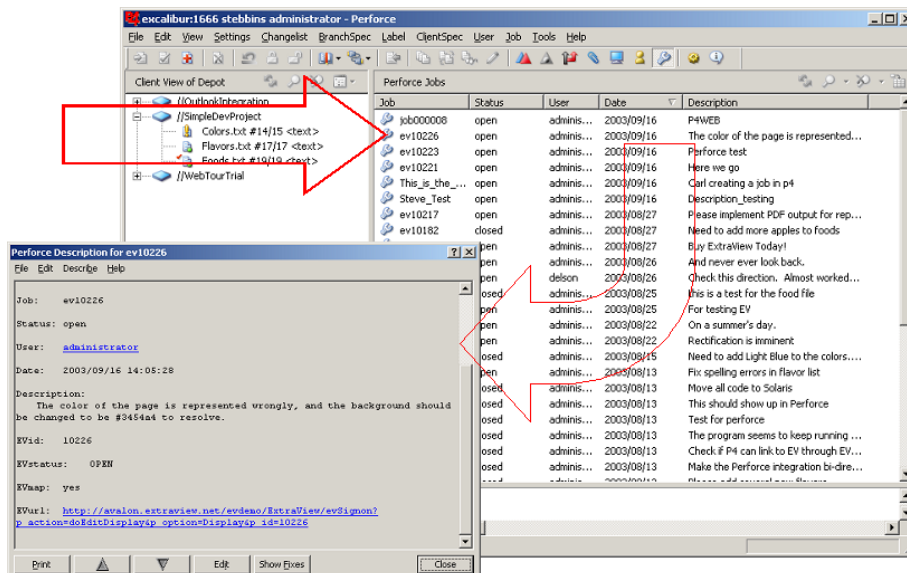
This case shows how a user will create an issue within ExtraView. The integration with Perforce will translate this into an action to create a new job within Perforce, mapping specific fields from ExtraView into the Perforce job.



Logical view



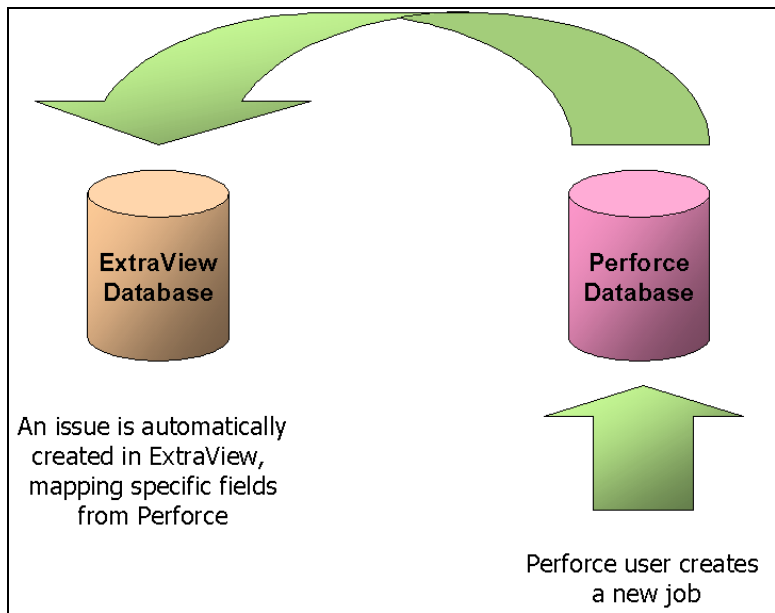
Adding the issue to ExtraView



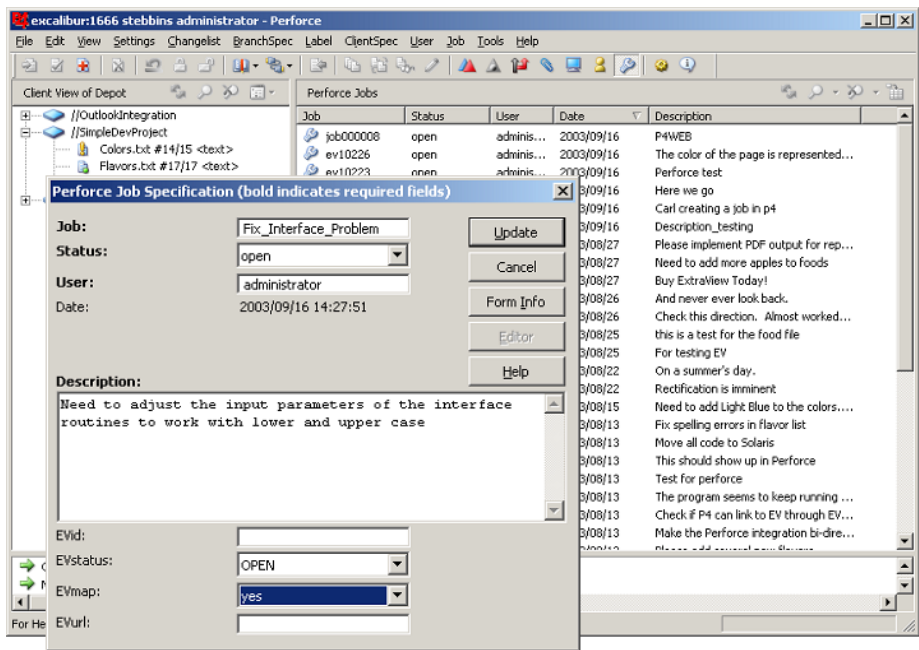
Viewing the new job within Performce

Use Case 2

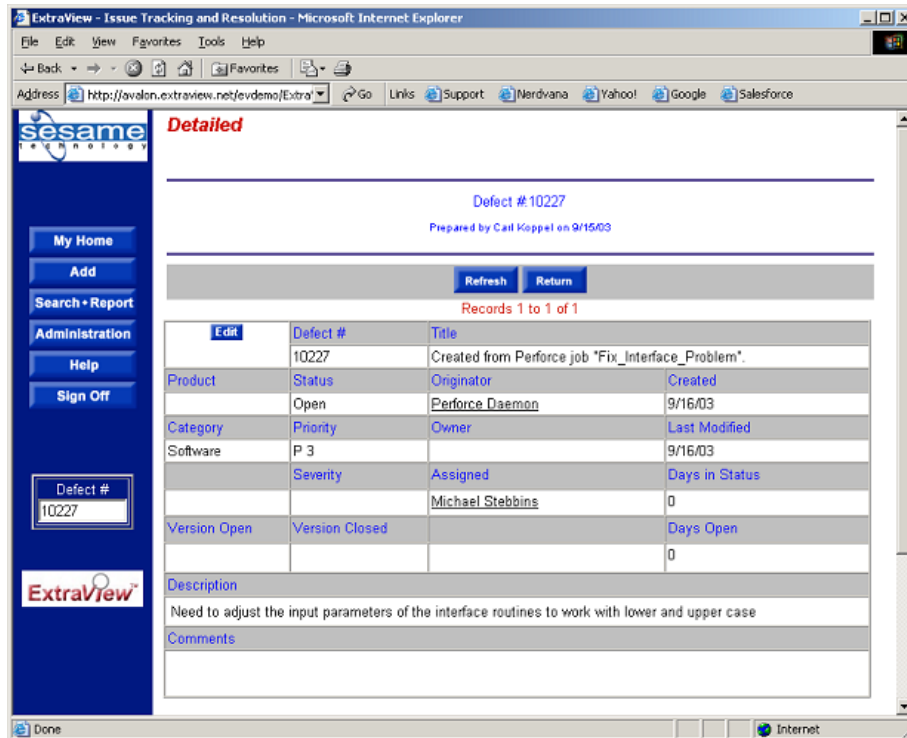
This use case is the inverse of the previous. A user adds a new job to Performce and the integration automatically creates the associated issue within ExtraView.



Logical View



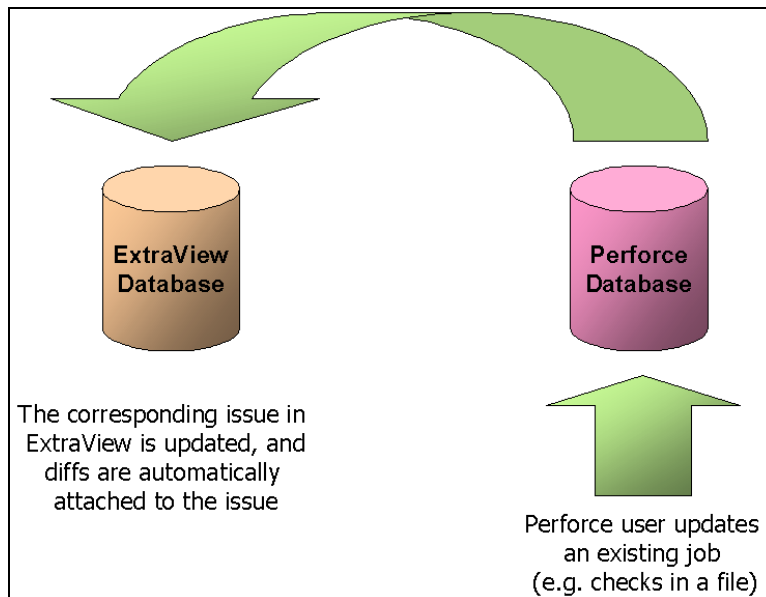
Creating the job in Performe



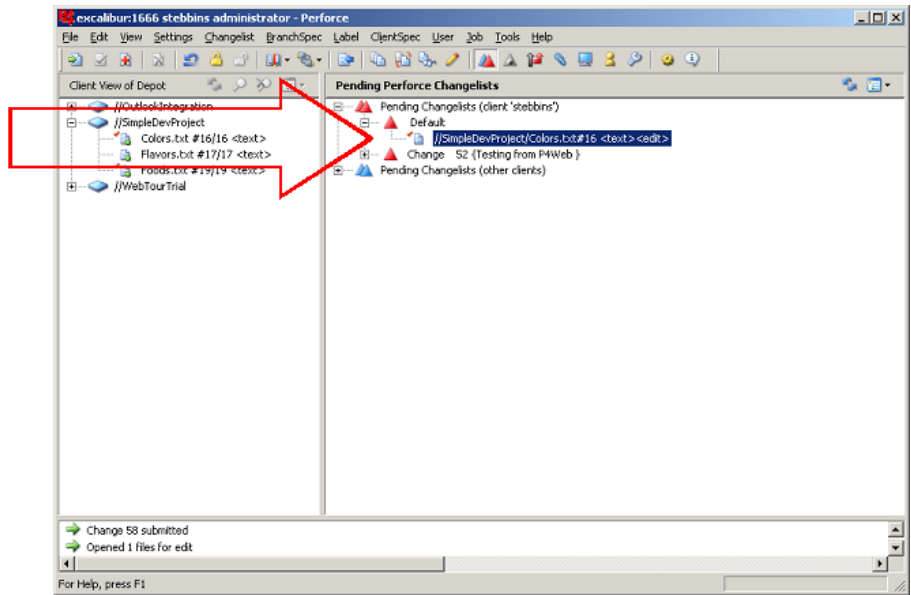
Viewing the issue that was automatically created in ExtraView

Use Case 3

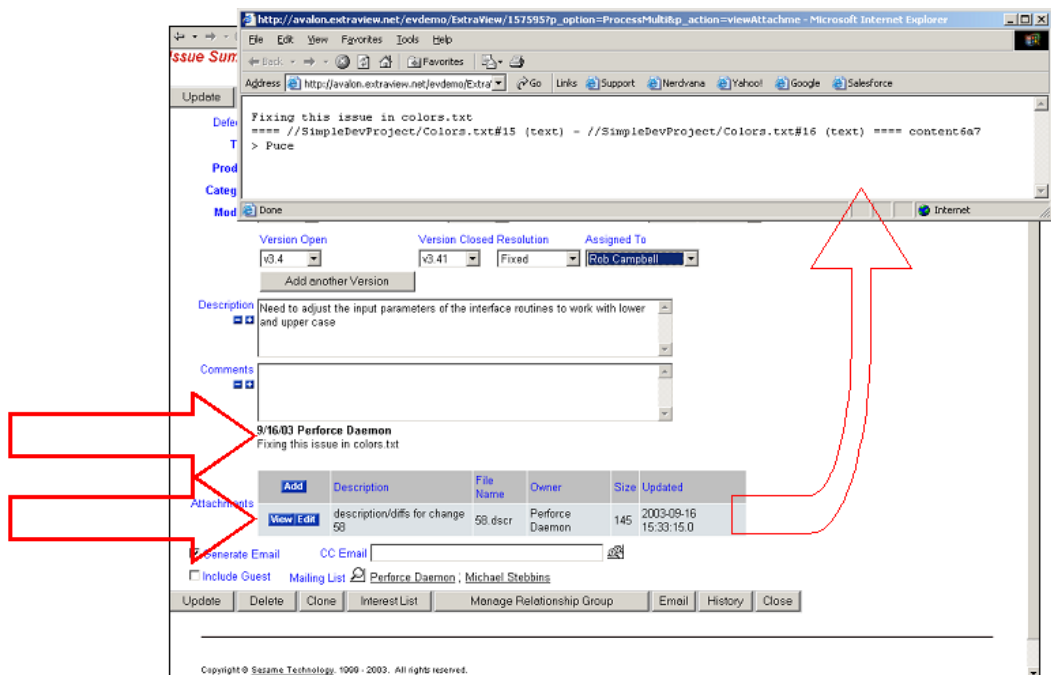
In this use case, the user updates a job in Perforce. The use case shows how the user can view the results of the update in ExtraView. The diffs from the update in Perforce may be mapped to a file attachment to the issue, and various fields defined in the integration (such as the Status of this issue), have their values mapped.



Logical View



Updating the job within Performe



Viewing the results of the Update within ExtraView

Installation

Suppose your company is using Perforce (P4) for source code control and ExtraView for problem tracking. You want to integrate them. That is, you want it so that whenever a P4 job changes, or whenever a P4 changelist is submitted, that the changes in P4 are reflected in ExtraView.

Install Perforce

Download Perforce from <http://www.perforce.com> and install it on your system according to the instructions provided on the Perforce website.

For example, download Perforce from <http://www.perforce.com/downloads/perforce/r04.2/bin.ntx86/perforce.exe> and run it to install Perforce on a computer with a Windows operating system.

Install Java

Download the Java 2 runtime from <http://java.sun.com> and install it according to the installer and instructions provided on the website.

For example, download either the J2SE SDK or JRE from <http://java.sun.com/j2se/1.4.2/download.html> and install it to a directory on your system.

Environment Variables

Set the following environment variables:

- `JAVA_HOME` – the Java 2 runtime environment installation directory

Install the EVP4DJ files

In the directory where ExtraView is installed, within the Tomcat webapps folder, you will find a folder named `evj/WEB-INF/data` (you may have installed ExtraView in a different place). There are two files according to the operating system upon which you are going to install EVP4DJ.

| | |
|---------------------------|----------------------------------|
| Windows operating systems | <code>evp4daemon_win.zip</code> |
| Linux, Solaris | <code>evp4daemon_unix.tar</code> |

Install all the EVP4DJ files in a directory on your system (referred as *EVP4DJ_HOME* in this document) by unzipping or untarring the appropriate file. The following will be the directory structure of the EVP4DJ files under Windows:

```
EVP4DJ_HOME/  
  Configuration/  
  Data/  
  de-installService.bat  
  evp4dj.bat  
  evp4dj.sh  
  ExtraViewService.exe  
  installEVP4DJService.bat  
  lib/  
  logs/  
  start_commands.txt
```

Installing EVP4DJ as a Windows Service

This section only applies if you are installing the EVP4DJ on a Microsoft Windows server, and wish to run it as a service.

- Locate the `installEVP4DJService.bat` file in your `EVP4DJ_HOME` folder after you have unzipped `evp4daemon_win.zip`

- Edit the `installEVP4DJService.bat` file

- Replace

```
D:\ExtraView\j2sdk1.4.2_13\jre\bin\server\jvm.dll
```

with your path to `jvm.dll`

- If you have multiple JVM's on your computer, use the one under `jre\bin\server`
- Replace all occurrences of `%EVP4D_HOME%` with the path to your `evp4d` folder. Make sure this includes replacing the following:
 - `%EVP4D_HOME%\start_commands.txt`
 - `%EVP4D_HOME%\err.log`
- Save the edited batch file, and run it one time. This will create a new service in Windows called `ExtraViewPerforceDaemon`. It defaults to starting automatically, but you can change its startup parameters by using the Services control panel in Windows.

Configuration

Perforce Configuration

Create Client and User

Create a Perforce client and a user that will be used by EVP4DJ to create and update jobs in Perforce.

Edit Jobspec

As a Perforce administrator user, edit the jobspec according to your requirements and use of ExtraView. If you are using the command line version of Perforce, use the `p4 jobspec` command to edit the jobspec.

For example, here is a sample jobspec that includes the following additional fields for mapping to ExtraView issues: `EVID`, `EVSTATUS`, `EVpriority`, `EVurl`, and `EVtitle`.

```
> p4 jobspec -o
# A Perforce Job Spec Specification.
#
# Updating this form can be dangerous!
# See 'p4 help jobspec' for proper directions.

Fields:
    101 Job word 32 required
    102 Status select 10 required
    103 User word 32 required
    104 Date date 20 always
    105 Description text 0 required
    110 EVID word 10 optional
    111 EVSTATUS select 10 default
    112 EVpriority select 10 required
    113 EVurl line 50 optional
    114 EVtitle line 50 default

Values:
    Status open/suspended/closed
    EVSTATUS OPEN/UNASSIGNED/CLOSED/NEW/REJECTED/DUPLICATE/FIXED
    EVpriority Critical/High/Medium/Low

Presets:
    Status open
    User $user
    Date $now
    Description $blank
    EVSTATUS UNASSIGNED
    EVpriority Medium
    EVtitle "Default Job Title"

Comments:
```

```
# A Perforce Job Specification.
#
# Job:          The job name. 'new' generates a sequenced job number.
# Status:      Either 'open', 'closed', or 'suspended'.
# User:        The user who created the job. Can be changed.
# Date:        The date this specification was last modified.
# Description: Comments about the job. Required.
# EVid:        ExtraView issue id. Optional.
# EVstatus:    Either OPEN, UNASSIGNED, CLOSED, NEW, REJECTED,
#             DUPLICATE, FIXED. Optional.
# EVpriority:  Either Critical, High, Medium, or Low. Required.
# EVurl:       Shortcut URL to ExtraView issue. Optional.
# EVtitle:     Short description. Optional.
```

EVP4DJ Configuration

Create ExtraView EVP4DJ User

Create a new ExtraView user that will be used by EVP4DJ to access the ExtraView API to create and update issues in ExtraView. This user must be assigned a role that has access to query and update mapped fields on ExtraView screens. See the Edit Configuration File section for the list of fields that are to be mapped between Perforce jobs and ExtraView issues.

Create ExtraView Job Name Field

In the Data Dictionary administration section of ExtraView, add a new UDF Text Field that will be used to contain the Perforce job name. For example, you can call the new job name field `PERFORCE_JOB_NAME`.

The job name is permitted to contain "-", "_", "\$", "." and alphanumeric characters in any position.

Modify ExtraView Layouts

In ExtraView, either modify existing layouts or create new layouts for the adding, editing, and searching of ExtraView issues that will be mapped to Perforce jobs. If your ExtraView site uses role-specific layouts, modify the layout that corresponds to the role of the new ExtraView user created for EVP4DJ use.

Note: When the integration triggers, the EVP4DJ daemon uses the ExtraView **Detailed Report** layout appropriate to the ExtraView user's area/project and role to retrieve the values for the issue being synchronized. This approach uses the user's permissions for access to the fields. Like all features in ExtraView, the Perforce integration is secure and users must have permissions for fields in order to be able to access them and to update them. The most important aspect of this is that the ExtraView fields you are mapping to Perforce fields must appear on the detailed report layout, and the user must have read and write permission to these fields within ExtraView.

Set Security Permissions for the Mapped Fields

In the Administration section of ExtraView, grant both add and edit field-level security permissions to read and write for the ExtraView issue fields that are mapped to Perforce job fields for the role given to the ExtraView EVP4DJ user.

Set Behavior Setting for Repeating Rows

If you intend to map Perforce jobs to repeating rows within ExtraView, you must make sure that the behavior setting in the Administration System Controls section named MULTI_RELEASE_XML is set to YES.

Set Status Change Rules

If you have the behavior setting named ENABLE_STATE_CHANGE_RULES set to YES, assign the appropriate status change rules according to your designed workflow for the ExtraView EVP4DJ user role. This behavior setting is found in the Workflow settings section of administration. Setting the status change rules is found in the same administration section.

ExtraView – Perforce Field Mapping

Determine which ExtraView issue fields will be mapped to Perforce job fields, and vice versa. This step will define a critical piece of the ExtraView – Perforce Integration. Define a mapping between new or existing Perforce Jobspec fields and new or existing ExtraView Data Dictionary fields. For example, using the sample Jobspec from the previous section, define a mapping between the Perforce `EVID` field and the ExtraView `ID` field. Another sample mapping may be between the Perforce `DESCRIPTION` field and the ExtraView `DESCRIPTION` field.

Mapping the Status fields

Both ExtraView and Perforce have a field named **STATUS** that has special treatment within each software product. However, ExtraView's treatment of the **STATUS** field is considerably more sophisticated than that of Perforce. This introduces some challenges in the integration process and the administrator must make some decisions about how the integration work flow should be managed. The most important point is that for ExtraView to successfully synchronize issues to their counterpart jobs in Perforce there must exist a single status value in each product that maps to each other. If you do not do this then updates between each product may be ambiguous as to their status. We suggest you use these guidelines –

- Create the same **STATUS** field values in both ExtraView and Perforce
- Map each value to its corresponding value in the other product (instructions follow in this section)

- If you have configured status change rules in ExtraView, these rules are not enforced in Perforce. **If your users make errors in the following of your workflow through the change in value of the STATUS field, this may have significant repercussions on your ability to synchronize issues.** For example, if a user is not allowed to move a *Closed* issue to *Fixed* in ExtraView, but makes this change in Perforce, the synchronization of the issue will fail, as ExtraView will not allow the transition to take place. Such errors are reported in the log file. If you are seriously concerned about this potential for lost updates, you may want to turn off the status change rules in ExtraView
- An alternative approach is to map the ExtraView **STATUS** field to a field you define in the Perforce Jobspec file.

Mapping user name fields

It is also recommended that you provide a one-to-one mapping of user field names between ExtraView and Perforce. This will ensure that no ambiguous updates are made.

Edit the Configuration File

The EVP4DJ configuration file is named `p4.properties` and is located in the `configuration/` directory of your EVP4DJ installation. Edit the `p4.properties` file and set the following settings. See Appendix B for a list and description of each configuration setting.

Perforce server name and port

```
p4.port=<server>:<port>
```

For example,

```
p4.port=localhost:1666
```

Perforce client, user, and password

```
p4.client=<client name>
```

```
p4.user=<user name>
```

```
p4.password=<user password>
```

For example,

```
p4.client=CLIENT1
```

```
p4.user=user1
```

```
p4.password=password1
```

ExtraView URL, user, and password

```
SERVER = http://<server>/evj/ExtraView/ev_api.action
```

```
USER = <user name>
```

```
PASSWORD = <user password>
```

For example,

```
SERVER = http://server.domain.com/evj/ExtraView/ev_api.action
USER = evp4dj_user
PASSWORD = evp4dj_password
```

Field Name Mapping

```
p4fn.<Perforce field> = evfn.<ExtraView field>
evfn.<ExtraView field> = p4fn.<Perforce field>
```

For example,

```
p4fn.Description=evfn.DESCRPTION
evfn.PERFORCE_JOB_NAME=p4fn.Job
```

Note: the ExtraView field names used here are not the same as the titles seen on the GUI; they are the “database names” as can be seen in the Data Dictionary utility.

Note: the “p4fn” and “evfn” fields can be on either side of the equals sign.

Field Value Mapping

```
p4v.<Perforce field>.<value> = evv.<ExtraView field>.<value>
evv.<ExtraView field>.<value> = p4v.<Perforce field>.<value>
```

For example,

```
p4v.EVStatus.NEW=evv.STATUS.New
p4v.user.user1=evv.user.evp4dj_user
```

Note: The field values used here are the field titles that are also seen in the GUI, not the “database names”.

Filter Perforce Jobs

This section explains how to filter updates to Perforce jobs, such that only the changes you want are propagated to ExtraView. The configuration file entries are:

```
MJ_FIELD = <Perforce field>
MJ_REGEX = <regular expression>
MJ_POLL_TRIGGER = <boolean/arithmetic expression>
```

For each modified job or job that is part of a submitted changelist, one or two tests are executed to determine whether or not to propagate the job values to ExtraView: the MJ_FIELD test and the MJ_POLL_TRIGGER test. If either test passes, then the job values are propagated to ExtraView for creation or modification of an item associated with that job. These tests are very closely similar to the MI_FIELD and MI_POLL_TRIGGER tests used for ExtraView issues.

The MJ_FIELD test requires two property settings: MJ_FIELD and MJ_REGEX. The MJ_FIELD identifies a Perforce job field. The MJ_REGEX is a regular expression, which is applied to the job field current value, and, if there is a match in the regular expression, the test passes. See Appendix D for the definition of regular expressions.

The MJ_POLL_TRIGGER test requires only the MJ_POLL_TRIGGER property, which is a text string containing a Boolean/arithmetic expression. If the expression evaluates to non-zero, then the test passes. Contained within the expression can be replaceable variable names: these are job field names from the Perforce job that are surrounded by “_”, for example, “_Status_”. As part of the expression evaluation, the replaceable variable names are replaced with the dynamic values obtained from the job under test. See Appendix C for the syntax of an arithmetic/Boolean expression.

For example,

```
MJ_FIELD = Status
MJ_REGEX = closed
MJ_POLL_TRIGGER = __EVpriority__ eq "Critical" ||
                 __EVpriority__ eq "High"
```

Filter ExtraView Issue Updates

This section explains how to filter updates to ExtraView, to only map the updates you want to Perforce:

```
MI_FIELD = <ExtraView field>
MI_REGEX = <regular expression>
MI_POLL_TRIGGER = <boolean/arithmetic expression>
```

For each modified item in the ExtraView histories, one or two tests are executed to determine whether or not to propagate the item modifications to Perforce: the MI_FIELD test and the MI_POLL_TRIGGER test. If either test passes, then the item is propagated to Perforce for creation or modification of a job associated with that item. These tests are very flexible, making them somewhat complex; it is for this reason that we describe them here in detail.

The MI_FIELD test requires two property settings: MI_FIELD and MI_REGEX. The MI_FIELD identifies an ExtraView field whose value is part of the item’s history. The MI_REGEX is a regular expression, which is applied to the field value, and, if there is a match in the regular expression, the test passes. See Appendix D for information of regular expressions.

The MI_POLL_TRIGGER test requires only the MI_POLL_TRIGGER property, which is a text string containing a Boolean/arithmetic expression. If the expression evaluates to non-zero, then the test passes. Contained within the expression can be replaceable variable names: these are field names from the ExtraView item that are surrounded by “_”, for example, “_priority_”. As part of the expression evaluation, the replaceable variable names are replaced with the dynamic values obtained from the ExtraView item under test. See Appendix C for the syntax of an arithmetic/Boolean

expression.

For example,

```
MI_FIELD = PRIORITY
MI_REGEX = [12]
MI_POLL_TRIGGER = __STATUS__ eq Open || __STATUS__ eq Fixed
||
                __STATUS__ eq Closed
```

JOBNAME_FIELD

JOBNAME_FIELD = <ExtraView Job Name field>

For example,

JOBNAME_FIELD = PERFORCE_JOB_NAME

EVID_FIELD

EVID_FIELD = <Perforce field mapped to ExtraView ID value>

For example,

EVID_FIELD = Evid

JOBNAME_PATTERN

The configuration property `JOBNAME_PATTERN` is optional. If set, it contains the pattern to be used to generate a job name in Perforce when there is no mapped job name from the issue or the mapped job name is invalid, such as all blanks.

The `JOBNAME_PATTERN` may contain mapped issue field names for variable substitution. These are, like those in the regular expressions used elsewhere in `p4.properties`, surrounded by double underline characters.

Note that it is important to include at least one field that makes the job names unique per issue. Otherwise, a one-to-many mapping of jobs to issues will occur, causing indeterminate replications.

You can use `__ID__` to guarantee uniqueness. The substituted value will be the ID of the issue, and, if the job name field is mapped to a repeating row, it will consist of the issue ID appended with "." and the repeating row id.

Example:

```
JOBNAME_PATTERN = evjob__ID__
```

This will produce job names like **evjob1123**, or **evjob1123.123456789**.

P4_INSERT_VALUES

You may use the configuration property `P4_INSERT_VALUES` to set default values for fields in Perforce jobs. The syntax is the same as that for `INSERT_VALUES`:

```
P4_INSERT_VALUES.<p4fieldname>=<value>
```

Triggering Notification on Synchronization

Each job can specify whether or not to send email notification when the ExtraView issue is updated by EVP4D. This is accomplished through an ExtraView pseudo-field called SEND_MAIL.

SEND_MAIL can take on either of two values: YES or NO. When the value is YES, notification is sent, when it NO, no email notifications are sent. If the value is not present or is anything other than YES, it is treated as NO. By default, the value is always null, unless specified by the Perforce interface mapping, so no notifications will be sent.

To send emails optionally based on values in the job fields, define a job field in the P4 job specification to map to the SEND_MAIL field in the ExtraView issue. This could be done as in the example p4.properties snippet as follows:

```
p4fn.GenerateEmail=evfn.SEND_EMAIL
p4v.GenerateEmail.Y=evv.SEND_EMAIL.YES
p4v.GenerateEmail.N=evv.SEND_EMAIL.NO
```

What this says is that the p4 job field "GenerateEmail" will map to the ExtraView item pseudo-field named SEND_EMAIL. For a value of "Y" in the GenerateEmail field in the P4 job, notification of email updates will occur, because this maps to the value of YES for the pseudo-field SEND_EMAIL. Otherwise, the update will not cause any notification.

Perforce Job Pseudo Fields

These are special fields with the following properties:

- These fields do not exist in any Perforce job and are not replicated there
- They have specific semantics that depend on the name of the pseudo-field
- The pseudo fields may be mapped to ExtraView fields like any "real" Perforce job field
- Pseudo fields behave as if they are a "word" or "text" type (not a user, line, or select type)

Two pseudo-fields are implemented:

| Pseudo Field | Explanation |
|----------------|---|
| p4_filelist | This contains the list of file names affected by this job. The list is developed from the list of changes in the job. The file names include the depot path and are separated by semicolons |
| p4_changelist: | This contains the list of changes derived from the job. Each change is a number, and the changes are separated by semicolons |

Example: The p4.properties file might contain:

```
p4fn.p4_filelist = evfn.fred_text_field
```

And, fred_text_field will be populated by something similar to:

```
//depot/42deltas.txt; //depot/ver1.0/got blanks/cfold.txt
```

Fields with Special Treatment

| | |
|--------------------------------|--|
| LogArea fields | <p>Fields with an ExtraView display type of LogArea, such as the ExtraView field named COMMENTS, receive special handling. The normal method of updating a comment is to append a new entry to existing comments. The EVP4D process emulates this in the following manner, in order to avoid duplication of the existing comments in ExtraView:</p> <p>LogArea fields that are mapped from ExtraView to/from Perforce will be replicated in their entirety, including the all the historic comments. When modifying a LogArea field in Perforce, the user should only add new text to the end of the field; only the new text will be appended to ExtraView and the resulting LogArea field will be replicated back to the Perforce job.</p> |
| User fields | <p>Defining a field to be a Perforce user field provides for a special kind of mapping where all mapped User ID's are shared across all Perforce user fields.</p> <p>For example, if you have two Perforce user fields, p4user1 and p4user2, then you might want to use the user ID named <i>FRED</i> in either or both of these. <i>FRED</i> should be mapped once for all Perforce user fields, rather than on a per-field basis, like other mappings. See the following:</p> <pre>p4user.p4user1 p4user.p4user2 evv.user.freddieEV=p4.user.fred</pre> <p>Now, having a value of <i>FRED</i> in either the field p4user1 or the field p4user2 maps directly to the freddieEV user ID in ExtraView. This is different from other types of mappings because the mapping is not specific to the field, but to the field type, thus saving the repeated mapping of the field for each Perforce user field.</p> |
| ExtraView Repeating row fields | <p>If you map a field on ExtraView's repeating row layout to the Perforce job, then the behavior of ExtraView is</p> |

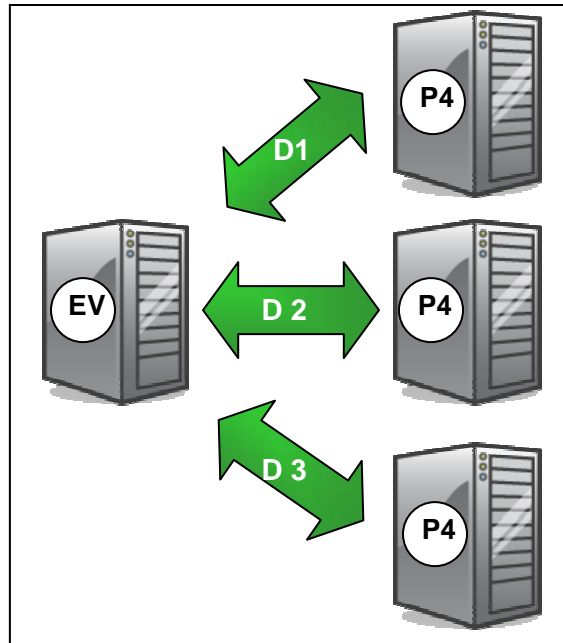
that it needs to know a different job number for each repeating row. The EVP4D process creates a job number for each repeating row mapped, of the format:

aaaaa . bbbbbbbbbb

where aaaaa is the ExtraView ID of the issue, and bbbbbbbbbb is the internal ID of the repeating row. This combination gives a unique mapping from each repeating row to each job.

Integrating with Multiple Perforce Servers

To integrate with multiple Perforce servers, you configure multiple daemons, each with its own properties file, and each pointing to a different Perforce repository. There is nothing special required, the same ExtraView user may be used within each properties file, or different users may be used.



Configuring multiple Perforce servers

The key to successfully using multiple Perforce servers is to configure the p4.properties files such that they will independently fire the triggers, based upon your needs.

For example, you may have your Perforce servers configured around products, such that one set of products reside within one server, another set of products within the next server, etc.

As explained in the last section, the following entries control which issues ExtraView communicates to the Perforce server:

```
MI_FIELD = <ExtraView field>
MI_REGEX = <regular expression>
```

```
MI_POLL_TRIGGER = <boolean/arithmetic expression>
```

You should set these entries in each of the different p4.properties files to be mutually exclusive, i.e. only one daemon will respond to one set of triggers. In our example, the field upon which you will trigger will be the product_name field within ExtraView with an entry similar to:

```
MI_POLL_TRIGGER = __product_name__ eq MyProduct
```

Of course, MyProduct is the title to the product selected for that daemon and other daemons will be configured with the titles of other products.

The daemons may all be configured on one server, or configured on separate servers. The entries within the p4.properties file on each daemon simply point to the appropriate Perforce and ExtraView servers.

Running EVP4DJ

Summary of Steps to Start EVP4DJ

The following steps are documented for a Microsoft Windows operating system installation. Using Linux or other operating system requires you to replace the names of the Windows specific files (such as .bat files) with their Linux counterpart (.sh or similar).

- Start the Perforce server as a Windows Service
- Start the daemon – EVP4DJ_HOME/evp4dj.bat

Start Perforce Server

Start Perforce according to the product documentation.

For example, on Windows start the Perforce Service to start Perforce. To stop Perforce, simply stop the Perforce Service in Windows.

Start EVP4DJ

Note that the following commands are wrapped into a single script that can be run as follows:

```
cd EVP4DJ_HOME
evp4dj.bat
```

State File

Certain dynamic runtime state variables are maintained in a persistent properties file, named `state.properties`. This file is maintained in the “data” directory and it must be available for writing by the application (e.g., privileges are set appropriately and the file must not be “in use by another application”).

The state file contains the state of progress data about the application, such as the latest ExtraView history already processed or the latest change number processed in Perforce.

Since state information is maintained internally during the operation of the daemon, it will do no good to modify the values in this file while the daemon is active. The values can be modified when the daemon is inactive, and the new values will be used when the daemon restarts.

The following keys and values are examples of state properties; note that all timestamps are maintained in a universal timestamp format, including time zone.

```
EV_TIMESTAMP=2004-11-27 14\ :26\ :49.000 -0800
P4_CHANGE=48
P4_TIMESTAMP=2004-11-29 13\ :05\ :38.000 -0800
```

Error Log

The file `error.log` is in the `logs` subdirectory of the directory in which EVP4DJ is installed.

The log file contains entries recounting EVP4DJ activity, including exceptions and other entries depending upon the log level setting.

Log information includes the P4 job names that matched and mapped to ExtraView issues, ExtraView issue ids that matched and mapped to P4 jobs, the names/ids of newly created jobs/issues, actions that failed, and notification of the updating of cutoff thresholds.

If the daemon shuts down due to a failure, the log will include as much information about the failure as it is able to ascertain.

Note: Configuration errors such as fields mapped incorrectly, may cause the daemon to shut down, and a log entry to be created. This is because continued operation may result in a loss of synchronization of the ExtraView and P4 databases.

APPENDIX A

CommandTXNPoller

This application manages the EVP4DJ application directly; it:

- Initializes transaction queues for EV-P4 communication
- Initializes message queues for command line interaction
- Starts the threads for EV interface and P4 Interface
- Provides for startup, shutdown, status, and configuration commands to both EV and P4 interfaces.

This is the primary application to initiate as a daemon for EV-P4 integration.

Command Line Options for CommandTXNPoller

The following options are available for invoking CommandTXNPoller:

| Option | Values | Significance |
|--------|------------|--|
| -f | File Name | Specify a file containing command lines (default is none) |
| -l | Log Level | Specify the level for logging daemon events (default is 6) |
| -q | Queue Name | Specify the message queue name for interprocess communications (default is "P4Commands") |

Note: the file containing commands has one command per line, e.g.

```
e:start  
p:start
```

EVP4CommandLine

This application runs as a separate application from EVP4DJ to provide a simple command-line, interactive interface to the CommandTXNPoller. It uses the JMS message queue functions to issue commands to, and read responses from the CommandTXNPoller.

There are four commands possible to the EVP4CommandLine:

- START
- STOP

- STATUS
- QUIT

The START, STOP and STATUS commands must be directed to one of the interface threads, either ExtraView or Perforce. To direct a command to an interface, the command is prepended with “E:” (for ExtraView) or “P:” for Perforce. Thus, to issue a status command to Perforce, one would type:

```
P : STATUS
```

START Command

This command is used to start the polling for the interface to which the command is directed. Thus, “E:START” starts the ExtraView interface polling for historical issue modifications that may be communicated to Perforce, and “P:START” starts the Perforce polling for changes or job modifications that may be communicated to ExtraView.

STOP Command

This command is used to stop the polling for the interface to which the command is directed. The polling is immediately terminated, possibly leaving some transactions in the intermediate queues in an unprocessed state.

STATUS Command

This command is used to determine the current state of the interface to which the command is directed. The possible states are:

- INIT: the thread is initialized, but the polling is not started
- RUN: the thread is running and the polling is active

The STATUS command takes one additional parameter on the command line: the name of a message queue that is used for receiving the status response from CommandTXNPoller. The returned status text is displayed on standard out. After issuing the status command to CommandTXNPoller, the command waits for 10 seconds for a response; if no response is received in 10 seconds, the command terminates; if a response is received, the command terminates after displaying the response.

QUIT Command

This command is not directed at any particular interface, but rather affects the CommandTXNProcessor itself. The QUIT command immediately terminates the CommandTXNProcessor and stops all processing of the child ExtraView and Perforce interface threads. The QUIT command should only be used

when the daemon is quiescent, such as after a STOP on both interface threads.

Appendix B – P4 Properties File Contents

| Key Name | Value(s) | Meaning |
|-------------------------------|----------------------|--|
| DEFAULT_LOG_LEVEL | Logging level number | Specifies which log messages are sent to the log file: higher numbers denote more detailed logs |
| evfn.<ev field name> | p4fn.<p4 field name> | Map the named ExtraView field to the named p4 field |
| p4fn.<p4 field name> | evfn.<ev field name> | Map the named ExtraView field to the named p4 field |
| EVID_FIELD | <p4 field name> | Identify the name of the p4 field containing the mapped ExtraView item ID |
| EVID_REPL | __<variable name>__ | Identify the ID replacement variable for EVURL_FIELD |
| EVURL_FIELD | <p4 field name> | Identify the name of the p4 field that contains the URL for drilldown to the associated EV item |
| FIXJOB_EVSTATUS | <ev field value> | The value to be set into the “status” and “release_status” fields when a changelist fixes a job. |
| INSERT_VALUES.<ev field name> | <ev field value> | The value to be used to set a required field in all item INSERT operations in ExtraView. |
| JOBNAME_FIELD | <ev field name> | Identify the ExtraView field that contains the associated job name |
| LOG_DATE_FORMAT | <valid date format> | The date format to use in generating log records. Default: yyyy-MM-dd |

| Key Name | Value(s) | Meaning |
|-----------------------------|-----------------------|--|
| LOG_TIME_FORMAT | <valid time format> | The time format to use in generating log records: Default kk:mm:ss |
| LOG_FILE_MANAGE_SCRIPT | <script name> | Name of a script to run when switching log files. Default: none |
| LOG_FILE_MAX_RETAINED | <number of log files> | This is the maximum number of old log files retained, when switching from one log file to the next. Default is -1 (infinite) |
| LOG_FILE_MAX_SIZE | Size of file in bytes | This is the maximum size that the logger will permit the log file to grow (default is 10MB). |
| LOG_FILE_DIR | <directory pathname> | The directory name to use for the log file (default is ./logs). |
| LOG_FILE_NAME | <file name> | The file name for the log file (default is "error.log") |
| LOG_FILE_DIR_ABSOLUTE | true or false | If true, the directory name for the log file is absolute, not relative to the current log file directory (./logs) |
| LOG_FILE_PATH_NAME_ABSOLUTE | true or false | If true, the file path name for the log file is not relative to the current log file directory (./logs). Default is false. |
| LOG_TIME_AT_CREATION | true or false | If true, the log file name is generated with a creation date/time string appended. Default is false. |
| MI_FIELD | <ev field name> | Identify the field to be inspected for the MI_REGEX evaluation to determine if an ExtraView item maps to p4 or not |

| Key Name | Value(s) | Meaning |
|-----------------|---|--|
| MI_POLL_TRIGGER | Boolean expression with replaceable variables | The expression that determines if an ExtraView item maps to p4 or not |
| MI_REGEX | Regular expression | The expression that is used with MI_FIELD value to determine if an ExtraView item maps to p4 or not |
| MJ_FIELD | <p4 field name> | Identify the field to be inspected for the MJ_REGEX evaluation to determine if a p4 job maps to ExtraView or not |
| MJ_POLL_TRIGGER | Boolean expression with replaceable variables | The expression that determines if a p4 job maps to ExtraView or not |
| MJ_REGEX | Regular expression | The expression that is used with MJ_FIELD value and MJ_YESMAP to determine if a p4 job maps to ev or not |
| MJ_YESMAP | Positive resultant value for MI_REGEX | The value for MJ_REGEX that determines if a p4 job maps to ExtraView or not |
| p4.client | Client for p4 | The client used for accessing p4 |
| p4.executable | Path name of executable for p4 | ** optional **Sets up the path to reach the p4 executable. The full path passed in must contain the executable or at least end in the system's file separator. |
| p4.logfile | <log file pathname> | The file name where the perforce log file is created. Default is:./logs/p4log.log |
| p4.log_level | none, split, or only | Specifies which log messages are sent to the p4 log. |
| p4.password | Password for p4 | The password used for accessing p4 |

| Key Name | Value(s) | Meaning |
|--------------------------------------|--------------------------------------|---|
| p4.port | Port for p4 | Port for p4 access; might be in the form "hostname:port#" |
| p4.sysdrive | Sysdrive for p4 | Sets system drive; only meaningful in Windows (see notes) |
| p4.sysroot | Sysroot for p4 | Sets system root; only meaningful in Windows (see notes) |
| p4.threshold | <msec for timeout> | Specifies the number of milliseconds to wait for a response from performe. Default is 10000. |
| p4.user | <p4 user id> | The p4 user id used for accessing p4 |
| P4DESC | <ev field name> | Where in ExtraView to place the description for a changelist from p4 |
| P4DESC_DIFFS | <ev field name>comment, file or none | Where in ExtraView to place the diffs for a changelist from p4. This allows you to store the diffs for historic purposes. You may use an ExtraView log area field, such as COMMENTS. In this way you will retain the diffs for all check ins within ExtraView |
| p4user.<p4 field name> | <any value> | Declares that this p4 field contains a user value |
| p4v.<p4 field name>.<p4 field value> | evv.<ev field name>.<ev field value> | Maps the named p4 field value to the named ExtraView field value |
| p4v.user.<p4 user id> | evv.user.<ev user id> | Maps the p4 user id to the ExtraView User ID (and vice-versa) |
| evv.user.<ev user id> | p4v.user.<p4 user id> | Maps the ExtraView User ID to the p4 user id (and vice-versa) |
| evv.<ev field name>.<ev field value> | p4v.<p4 field name>.<p4 field value> | Maps the named p4 field value to the named ExtraView field value |

| Key Name | Value(s) | Meaning |
|--------------|--------------------------------|--|
| PASSWORD | Password string | Password for access to EVAPI |
| SERVER | URL with replaceable variables | This string, with variables replaced, is stored into the p4 field named by EVURL_FIELD |
| SERVER_REPL | __<variable name>__ | Identify the server variable name for the EVURL_FIELD |
| USER | <ev user id> | The ExtraView User ID used for accessing EVAPI |
| XML_LOG_FLAG | true or false | Defines log format: false (plain text) is the default |

DEFAULT_LOG_LEVEL Values

The value set for the log level indicates that all log messages less than or equal to that level will be sent to the log file.

| | | |
|----------|----|---|
| ALERT | 1 | Site level threat or problem |
| CRITICAL | 2 | A configuration problem |
| ERROR | 3 | A component of the system in not operational |
| WARN | 4 | Example: A resource is getting low |
| NOTICE | 5 | For notices to operations staff |
| INFO | 6 | Normal program output like progress reporting |
| DEBUG | 7 | All debug messages are at DEBUG or below |
| DEBUG1 | 8 | |
| DEBUG2 | 9 | |
| DEBUG3 | 10 | |
| DEBUG4 | 11 | |
| DEBUG5 | 12 | |

Appendix C – Arithmetic/Boolean Expressions

In the Perl version of EVP4D (Version 1.0), the Perl “eval” function was used to evaluate a generalized expression used in the `xx_POLL_TRIGGER` property. Since Perl is no longer used or required, a substitute expression evaluation method is provided. We describe the expression evaluation syntax and semantics here.

An expression consists of one or more terms separated by infix operators:

```
<expression> = <term>[<infix op><term>]*
```

A term is a literal or a replaceable variable or a term preceded by a prefix operator or a parenthesized expression.

A term may also be an expression surrounded by parentheses:

```
<term> = <literal> | <replaceable variable>
        | ( <expression> ) | <prefix op>*<term>
```

A literal is a string literal or a number literal:

```
<literal> = "<inside quote string>" | <inside quote string>
           | <floating point number>
```

where `<inside quote string>` follows the usual rules for quoted strings.

Infix operators include both Boolean and arithmetic operators:

```
<infix op> = + | - | "*" | / | "&&" | "||" | .
           | eq | ne | gt | ge | lt | le | "=="
```

Only one prefix operator, the boolean NOT, is currently supported:

```
<prefix op> = !
```

Rules of Evaluation

Evaluation proceeds left to right, in order of infix precedence, with parenthesized expressions evaluated before the containing term is evaluated.

When mixing arithmetic and Boolean values, Boolean expressions are converted to 1.0 for true and 0.0 for false. Any non-zero arithmetic value is equivalent to true in a Boolean expression. All arithmetic computation uses floating point.

The infix precedence is assigned to make the higher-valued operations more tightly binding. The values are as follows:

- 7 – for the prefix “not” operator (!)

- 6 – for multiplication and division (* and /)
- 5 – for the string concatenation operation (.)
- 3 – for plus, minus and the relational operators (+, -, eq, ne, gt, ge, lt, le, ==)
- 1 – for the Boolean operators (||, &&)

Hence, an expression like:

`a + b ge (c+e)*2 || !d == e`

would be evaluated in the following order:

```
!d
(c+e)
*2
a + b
ge
== e
||
```

Expression Examples

```
(__priority__ le 1) || ( __status__ eq "fixed")
```

```
5/9 * ( __temperature__ - 32) gt 0
```

Appendix D – Regular Expressions

Summary of regular-expression constructs

| Construct | Matches |
|-------------------------------------|---|
| Characters | |
| <code>x</code> | The character <code>x</code> |
| <code>\\</code> | The backslash character |
| <code>\0n</code> | The character with octal value <code>0n</code> ($0 \leq n \leq 7$) |
| <code>\0nn</code> | The character with octal value <code>0nn</code> ($0 \leq n \leq 7$) |
| <code>\0mnn</code> | The character with octal value <code>0mnn</code> ($0 \leq m \leq 3, 0 \leq n \leq 7$) |
| <code>\xhh</code> | The character with hexadecimal value <code>0xhh</code> |
| <code>\uhhhh</code> | The character with hexadecimal value <code>0xhhhh</code> |
| <code>\t</code> | The tab character (<code>"\u0009"</code>) |
| <code>\n</code> | The newline (line feed) character (<code>"\u000A"</code>) |
| <code>\r</code> | The carriage-return character (<code>"\u000D"</code>) |
| <code>\f</code> | The form-feed character (<code>"\u000C"</code>) |
| <code>\a</code> | The alert (bell) character (<code>"\u0007"</code>) |
| <code>\e</code> | The escape character (<code>"\u001B"</code>) |
| <code>\cx</code> | The control character corresponding to <code>x</code> |
| Character classes | |
| <code>[abc]</code> | a, b, or c (simple class) |
| <code>[^abc]</code> | Any character except a, b, or c (negation) |
| <code>[a-zA-Z]</code> | a through z or A through Z, inclusive (range) |
| <code>[a-d[m-p]]</code> | a through d, or m through p: <code>[a-dm-p]</code> (union) |
| <code>[a-z&&[def]]</code> | d, e, or f (intersection) |
| <code>[a-z&&[^bc]]</code> | a through z, except for b and c: <code>[ad-z]</code> (subtraction) |
| <code>[a-z&&[^m-p]]</code> | a through z, and not m through p: <code>[a-lq-z]</code> (subtraction) |
| Predefined character classes | |
| <code>.</code> | Any character (may or may not match line terminators) |
| <code>\d</code> | A digit: <code>[0-9]</code> |
| <code>\D</code> | A non-digit: <code>[^0-9]</code> |
| <code>\s</code> | A whitespace character: <code>[\t\n\r\b\f]</code> |

| Construct | Matches |
|--|---|
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z_0-9] |
| \W | A non-word character: [^\w] |
| POSIX character classes (US-ASCII only) | |
| \p{Lower} | A lower-case alphabetic character: [a-z] |
| \p{Upper} | An upper-case alphabetic character:[A-Z] |
| \p{ASCII} | All ASCII:[\x00-\x7F] |
| \p{Alpha} | An alphabetic character:[\p{Lower}\p{Upper}] |
| \p{Digit} | A decimal digit: [0-9] |
| \p{Alnum} | An alphanumeric character:[\p{Alpha}\p{Digit}] |
| \p{Punct} | Punctuation: One of !"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~ |
| \p{Graph} | A visible character: [\p{Alnum}\p{Punct}] |
| \p{Print} | A printable character: [\p{Graph}] |
| \p{Blank} | A space or a tab: [\t] |
| \p{Cntrl} | A control character: [\x00-\x1F\x7F] |
| \p{XDigit} | A hexadecimal digit: [0-9a-fA-F] |
| \p{Space} | A whitespace character: [\t\n\x0B\f\r] |
| Classes for Unicode blocks and categories | |
| \p{InGreek} | A character in the Greek block (simple block) |
| \p{Lu} | An uppercase letter (simple category) |
| \p{Sc} | A currency symbol |
| \P{InGreek} | Any character except one in the Greek block (negation) |
| [\p{L}&&[^\p{Lu}]] | Any letter except an uppercase letter (subtraction) |
| Boundary matchers | |
| ^ | The beginning of a line |
| \$ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | The end of the input but for the final terminator, if any |
| \z | The end of the input |
| Greedy quantifiers | |

| Construct | Matches |
|---|---|
| $X?$ | X , once or not at all |
| X^* | X , zero or more times |
| X^+ | X , one or more times |
| $X\{n\}$ | X , exactly n times |
| $X(n,)$ | X , at least n times |
| $X\{n,m\}$ | X , at least n but not more than m times |
| Reluctant quantifiers | |
| $X??$ | X , once or not at all |
| $X*?$ | X , zero or more times |
| $X+?$ | X , one or more times |
| $X\{n\}?$ | X , exactly n times |
| $X(n,)?$ | X , at least n times |
| $X\{n,m\}?$ | X , at least n but not more than m times |
| Possessive quantifiers | |
| $X?+$ | X , once or not at all |
| $X*+$ | X , zero or more times |
| $X++$ | X , one or more times |
| $X\{n\}+$ | X , exactly n times |
| $X(n,)+$ | X , at least n times |
| $X\{n,m\}+$ | X , at least n but not more than m times |
| Logical operators | |
| XY | X followed by Y |
| $X Y$ | Either X or Y |
| (X) | X , as a capturing group |
| Back references | |
| $\backslash n$ | Whatever the n^{th} capturing group matched |
| Quotation | |
| \backslash | Nothing, but quotes the following character |
| $\backslash Q$ | Nothing, but quotes all characters until $\backslash E$ |
| $\backslash E$ | Nothing, but ends quoting started by $\backslash Q$ |
| Special constructs (non-capturing) | |
| $(?:X)$ | X , as a non-capturing group |

| Construct | Matches |
|-----------------------------|---|
| (? <i>idmsux-idmsux</i>) | Nothing, but turns match flags on - off |
| (? <i>idmsux-idmsux:X</i>) | X, as a non-capturing group with the given flags on - off |
| (? <i>=X</i>) | X, via zero-width positive lookahead |
| (? <i>!X</i>) | X, via zero-width negative lookahead |
| (? <i><=X</i>) | X, via zero-width positive lookbehind |
| (? <i><!X</i>) | X, via zero-width negative lookbehind |
| (? <i>>X</i>) | X, as an independent, non-capturing group |

Backslashes, escapes, and quoting

The backslash character (`\`) serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression `\\` matches a single backslash and `\{` matches a left brace.

It is an error to use a backslash prior to any alphabetic character that does not denote an escaped construct; these are reserved for future extensions to the regular-expression language. A backslash may be used prior to a non-alphabetic character regardless of whether that character is part of an unescaped construct.

Backslashes within string literals in Java source code are interpreted as required by the Java Language Specification as either Unicode escapes or other character escapes. It is therefore necessary to double backslashes in string literals that represent regular expressions to protect them from interpretation by the Java bytecode compiler. The string literal `"\b"`, for example, matches a single backspace character when interpreted as a regular expression, while `"\\b"` matches a word boundary. The string literal `"\hello"` is illegal and leads to a compile-time error; in order to match the string `(hello)` the string literal `"\\hello\\"` must be used.

Line terminators

A *line terminator* is a one- or two-character sequence that marks the end of a line of the input character sequence. The following are recognized as line terminators:

- A newline (line feed) character (`\n`),
- A carriage-return character followed immediately by a newline character (`\r\n`),
- A standalone carriage-return character (`\r`),

- A next-line character (`\u0085`),
- A line-separator character (`\u2028`), or
- A paragraph-separator character (`\u2029`).

If `UNIX_LINES` mode is activated, then the only line terminators recognized are newline characters.

The regular expression `.` matches any character except a line terminator unless the `DOTALL` flag is specified.

Groups and capturing

Capturing groups are numbered by counting their opening parentheses from left to right. In the expression `((A)(B(C)))`, for example, there are four such groups:

- | | |
|---|--------------------------|
| 1 | <code>((A)(B(C)))</code> |
| 2 | <code>(A)</code> |
| 3 | <code>(B(C))</code> |
| 4 | <code>(C)</code> |

Group zero always stands for the entire expression.

Capturing groups are so named because, during a match, each subsequence of the input sequence that matches such a group is saved. The captured subsequence may be used later in the expression, via a back reference, and may also be retrieved from the matcher once the match operation is complete.

The captured input associated with a group is always the subsequence that the group most recently matched. If a group is evaluated a second time because of quantification then its previously-captured value, if any, will be retained if the second evaluation fails. Matching the string "aba" against the expression `(a(b)?)+`, for example, leaves group two set to "b". All captured input is discarded at the beginning of each match.

Groups beginning with `(?` are pure, *non-capturing* groups that do not capture text and do not count towards the group total.

Unicode support

This class follows *Unicode Technical Report #18: Unicode Regular Expression Guidelines*, implementing its second level of support though with a slightly different concrete syntax.

Unicode escape sequences such as `\u2014` in Java source code are processed as described in §3.3 of the Java Language Specification. Such escape sequences are also implemented directly by the regular-expression parser so that Unicode escapes can be used in expressions that are read

from files or from the keyboard. Thus the strings "\u2014" and "\u2014", while not equal, compile into the same pattern, which matches the character with hexadecimal value 0x2014.

Unicode blocks and categories are written with the \p and \P constructs as in Perl. \p{*prop*} matches if the input has the property *prop*, while \P{{*prop*} does not match if the input has that property. Blocks are specified with the prefix In, as in InMongolian. Categories may be specified with the optional prefix ls: Both \p{L} and \p{lsL} denote the category of Unicode letters. Blocks and categories can be used both inside and outside of a character class.

The supported blocks and categories are those of *The Unicode Standard, Version 3.0*. The block names are those defined in Chapter 14 and in the file Blocks-3.txt of the Unicode Character Database except that the spaces are removed; "Basic Latin", for example, becomes "BasicLatin".

Comparison to Perl 5

Perl constructs not supported by this class:

- The conditional constructs (?{*X*}) and (?(*condition*)*X*|*Y*),
- The embedded code constructs (?{*code*}) and (??{*code*}),
- The embedded comment syntax (?#comment), and
- The preprocessing operations \l \u, \L, and \U.

Constructs supported by this class but not by Perl:

- Possessive quantifiers, which greedily match as much as they can and do not back off, even when doing so would allow the overall match to succeed.
- Character-class union and intersection. Character classes may appear within other character classes, and may be composed by the union operator (implicit) and the intersection operator (&&). The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that is in both of its operand classes.

The precedence of character-class operators is as follows, from highest to lowest:

| | | |
|---|----------------|----------------|
| 1 | Literal escape | \x |
| 2 | Grouping | [...] |
| 3 | Range | a-z |
| 4 | Union | [a-e][i-u] |
| 5 | Intersection | [a-z&&[aeiou]] |

Notable differences from Perl:

- In Perl, `\1` through `\9` are always interpreted as back references; a backslash-escaped number greater than 9 is treated as a back reference if at least that many subexpressions exist, otherwise it is interpreted, if possible, as an octal escape. In this class octal escapes must always begin with a zero. In this class, `\1` through `\9` are always interpreted as back references, and a larger number is accepted as a back reference if at least that many subexpressions exist at that point in the regular expression, otherwise the parser will drop digits until the number is smaller or equal to the existing number of groups or it is one digit.
- Perl uses the `g` flag to request a match that resumes where the last match left off. This functionality is provided implicitly by the `Matcher` class: Repeated invocations of the `find` method will resume where the last match left off, unless the matcher is reset.
- In Perl, embedded flags at the top level of an expression affect the whole expression. In this class, embedded flags always take effect at the point at which they appear, whether they are at the top level or within a group; in the latter case, flags are restored at the end of the group just as in Perl.
- Perl is forgiving about malformed matching constructs, as in the expression `*a`, as well as dangling brackets, as in the expression `abc]`, and treats them as literals. This class also accepts dangling brackets but is strict about dangling metacharacters like `+`, `?` and `*`, and will throw a `PatternSyntaxException` if it encounters them.

For a more precise description of the behavior of regular expression constructs, please see *Mastering Regular Expressions*, Jeffrey E. F. Friedl, O'Reilly and Associates, 1997.

Index

attachment, 8, 12
changelist, 7, 8, 14
Changelist, 8
CLI, iii
CommandTXNPoller, 29, 30
diffs, 8, 12, 35
Environment Variables, 14
error.log, 28
EVID_FIELD, 22, 32
EVP4DJ, 6, 7, 14, 15, 16, 17, 18, 19, 27, 28, 29
EVstatus, 7, 8
ExtraView, i, iii, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 25, 27, 28, 30, 32
Field Name Mapping, 20
Field Value Mapping, 20
Java, 6, 14, 42, 43
JOBNAME_FIELD, 22, 32
jobspec, 7, 16, 18
Jobspec, 16, 18
log file, 28
Map Job Settings, 20
MI_FIELD, 20, 21, 22, 25, 33, 34
MI_POLL_TRIGGER, 20, 21, 22, 26, 34
MI_REGEX, 21, 22, 25, 33, 34
MJ_FIELD, 20, 21, 34
MJ_POLL_TRIGGER, 20, 21, 34
Multiple Perforce Servers, 25
Perforce, i, iii, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 27, 30
Security Permissions, 18
Status Change Rules, 18
Windows Service, 15