

Plastic SCM Triggers

A guide for the Plastic SCM triggers

Release version 3.0

© 2006-2010 Codice Software

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Codice Software cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Table of Contents

1	Introduction	2
1.1	Purpose	2
2	Trigger types	3
2.1	Server side triggers	3
2.2	Client side triggers	3
3	Trigger usage	4
3.1	Creating the first trigger	4
3.2	List, edit and delete triggers	5
4	Trigger reference	6
4.1	List of triggers	6
4.2	Trigger operations	7
4.2.1	Create trigger	7
4.2.2	Show trigger types	8
4.2.3	List triggers	8
4.2.4	Change trigger	9
4.2.5	Remove trigger	10
4.3	Trigger communication	10
4.3.1	Input	10
4.3.2	Output	11
4.3.3	Common environment variables	11
4.3.4	Server.conf variables	11
4.4	Detailed trigger reference	12
4.4.1	Add	12
4.4.2	Check in	13
4.4.3	Check out	15
4.4.4	Create branch	17
4.4.5	Create label	18
4.4.6	Create attribute	19
4.4.7	Create repository	20
4.4.8	Create workspace	21
4.4.9	Set selector	21
4.4.10	Update	22
4.4.11	Client checkout	23
4.4.12	Client checkin	24
5	Samples	26
5.1	Checkin	27
5.1.1	Apply code beautifier to .java files	27
5.1.2	Apply modifying action to items in block	28
5.1.3	Check that comments have been provided on checkin	29
5.1.4	Generate rss with changeset contents	30
5.2	Make Label	31
5.2.1	Validate that label name starts with 'release'	31
5.3	Client checkout	31
5.3.1	Update files before checkout	31

About this guide

This guide describes the trigger mechanism available in Plastic SCM.

Audience

This guide is targeted to developers and system administrators, assuming familiarity with Plastic SCM and operating system concepts.

Online documentation

Besides this document and the rest of the guides, Plastic SCM provides online reference throughout its different client frontends.

On the command line interface, both Windows and Linux, this reference can be obtained with the command:

```
cm help
```

For extended information on a specific command, type:

```
cm help {command}
```

The graphical interface provides online reference through the Help menu.

Conventions

Along this guide, the syntax for several commands is described. In those descriptions, the following conventions are used:

- **Bold** items are literal text strings
- *{argument}* denotes a required argument
- [argument] denotes an optional argument

Documentation errors

If you find any problem in this guide or any other part of the online reference, please report it using the following email address:

support@codicesoftware.com

1 Introduction

1.1 Purpose

The trigger system in Plastic SCM allows the execution of user commands at certain points in the client or server execution workflow, in the form of shell scripts or any other operating system executable.

Among others, the trigger system in Plastic SCM will allow the developer or administrator to perform the following tasks:

- Enforce branch creation policies like naming conventions or making sure that branch names always refer to a certain associated task.
- Introduce before-checkin rules to enforce coding standards or create formatting rules.
- Enforce that comments are introduced on checkin.

Plastic SCM supports the association of several scripts to any given trigger, being able to perform different actions in sequence. The sequence in which scripts are executed can be customized by the user.

2 Trigger types

2.1 Server side triggers

This will be the most common type of trigger. When operations are performed by clients, such as creating revisions, branches or changing workspace configurations, the server is capable of executing user scripts before and after the operation completes.

The “pre-operation” triggers will usually allow cancelling the operation, depending on the result code of the triggered scripts.

The following is the complete list of triggers executed in the server:

- Create attribute.
- Create branch.
- Create marker.
- Create repository.
- Add items to the source control.
- Check-in.
- Check-out.

2.2 Client side triggers

Some events that occur on the client can have scripts or programs associated. Currently, supported client triggers are the following:

- Update (before and after)
- Checkout (before and after)
- Checkin (before and after)
- Create Workspace.
- Set Selector, affecting “Switch to...” operations also.

3 Trigger usage

This section describes the basic steps to get started with triggers and the recommended usage patterns. For a detailed reference on all trigger parameters and supported features, please check the Trigger reference section.

3.1 Creating the first trigger

In order to associate a user script with a client or server event, a trigger must be created. A list of the possible events that scripts can bind in order to be obtained by the following command:

```
cm showtriggertypes
```

In order to create a trigger which validates label names, the names should be created according to a given naming standard; the user may use a command like this (on Windows-based server):

```
cm maketrigger before-mklabel "check label name" "ruby  
c:\plastic\triggers\validate-label.rb"
```

This is a sample ruby script (validate-label.rb) which checks that the label name starts with 'release'. Otherwise, it returns 1, which means that the trigger fails and it doesn't allow the mklabel operation to finish:

```
if (ENV['PLASTIC_LABEL_NAME'] !~ /^release/) then exit(1) end
```

The script picks the name of the label from the PLASTIC_LABEL_NAME environment variable and checks its contents against the regular expression '^release', which means 'match a string that starts with 'release'. If this is not the case ('!~' operator), the exit code returned would be 1, which is interpreted as a trigger failure.

3.2 List, edit and delete triggers

To see the trigger that was just created, list the trigger of the type used:

```
cm listtriggers before-mklabel
1 Validate label c:\tmp\triggers\validate-label.bat dave
```

To modify the script that this trigger is pointing to the changetrigger command can be used; it is necessary to indicate the trigger type and the trigger position; which is the first index printed by the list triggers command (1 in this case):

```
cm changetrigger before-mklabel 1 --script="c:\tmp\other-script.bat"
```

To remove the trigger just created, use the remove command, indicating the trigger type and position:

```
cm removetrigger before-mklabel 1
```

4 Trigger reference

4.1 List of triggers

This is a complete list of all available events that triggers can be binded to:

before-add after-add	Fires on item addition, only once per 'add' command. A list of the added items is provided to the trigger script.
before-checkout after-checkout	Fires on checkout. A list of the checked out items is provided.
before-checkin after-checkin	Fires on checkin. A list of the items to be checked in is provided.
before-mkbranch after-mkbranch	Fires on branch creation.
before-mklabel after-mklabel	Fires on label creation.
before-mkattribute after-mkattribute	Fires on attribute creation.
before-mkrep after-mkrep	Fires on repository creation.

before-mkworkspace after-mkworkspace	Fires on workspace creation.
before-setselector after-setselector	Fires on any workspace selector change, including explicitly setting the selector and also the 'switch to...' commands.
before-update after-update	This is a client-side trigger. The path to be updated is provided to the script.
before-clientcheckin after-clientcheckin	This is a client-side trigger. The paths of the files and directories to be checked in are provided to both the <i>before</i> and <i>after</i> operations.
before-clientcheckout after-clientcheckout	This is a client side trigger. The paths of the files and directories to be checked out are provided to the trigger scripts. The user can use it to perform customized operations before or after a file is checked out.

A list of all supported trigger types can be obtained on the command line client with the command:

```
cm showtriggertypes
```

4.2 Trigger operations

4.2.1 Create trigger

Triggers are created from the command line client (cm). This is the syntax for the trigger creation command:

```
cm maketrigger {type} {name} {script}
  [--position=value]
  [--server=server:port]
```

Where:

- Type is the trigger type, as listed in the table of the previous section, p.e. before-mkbranch, or mkbranch-before. This argument is required.
- Name is the name given to the new trigger. It is informational only and more than one trigger can be assigned the same name (triggers are uniquely identified by its type and position). This argument is required.
- Script is the full path to the script or program that will be executed. The path points to a file in the Plastic server, so it needs to be a valid file specification that the server (either Windows or Unix) can understand. This argument is required.
- Position refers to the position in the execution list of the given trigger type for the script. This parameter determines the execution order if several scripts are registered on a given trigger type. If the position is already being used by other script, an error is raised and the trigger will not be created. This argument is

optional, and if omitted, the trigger will be added at the end of the current list of scripts.

- Server is the server in which to create the trigger. If omitted, the trigger will be created on the default configured server. The syntax specifies a server host name, ':', and a port, by default 8084.

Position of the script in the trigger type is unique, meaning that a list is maintained for each trigger type and positions in that list are either used by a trigger or not, but only one trigger can be assigned to a given position. If no position is specified, the trigger will be added to the end of the list. The user will be able to change the position in the list later using the 'changetrigger' command.

If the trigger script doesn't exist, an error will be raised when the affected operation is executed (thus preventing it from completion in any case).

Here are some sample usages. To create a trigger that fires after setting a workspace selector, located at /home/scripts/plastic-backup at the server, and give it the name "backup":

```
cm maketrigger after-setselector backup /home/scripts/plastic-backup
Trigger created on position 1.
```

To create a trigger that fires before a label is created, called "validate-label.bat" at server "myserver" on port 8084, and calling it "Validate label".

```
cm mktrigger before-mklabel "Validate label" "c:\tmp\triggers\validate-
label.bat" --server=myserver:8084
Trigger created on position 1.

cm listtriggers before-mklabel
 2 Validate label  c:\tmp\triggers\validate-label.bat dave
```

To create a trigger that validates checkin contents before the checkin is actually performed in the repository, on a Windows server:

```
cm maketrigger before-checkin ensure-code-stds
"c:\plastic\triggers\checkcode.bat"
Trigger created on position 3.
```

Sample trigger scripts can be found on section "Samples" at the end of the document.

4.2.2 Show trigger types

To get a list of the available trigger types the 'showtriggertypes' command must be used. Here is the full syntax of the command:

```
cm showtriggertypes
```

This command is purely informational and just lists the possible trigger types, so it is independent from any server or client.

4.2.3 List triggers

It is possible to get a list of the registered triggers for any given trigger type. The syntax of the command is the following one:

```
cm listtriggers {type}
  [--server=server:port]
  [--format=formatstring]
```

Where:

- Type is the trigger type to get the list of associated scripts, as listed in the table of section "List of triggers". This argument is required.
- Server is the server in which to create the trigger. If omitted, the trigger will be created on the default configured server. The syntax specifies a server's host name, ':', and a port, by default 8084.
- Format is the usual format specifier used in Plastic commands. A reference of the available column values can be found below.

Sample usage to list the scripts associated with the before-checkin event:

```
cm listtriggers before-checkin
1 checkstyle c:\tmp\triggers\checkin-checkstyle.bat dave
```

This command will output one line for each trigger defined. This is the meaning of the output columns:

- 0.- Trigger position
- 1.- Trigger name
- 2.- Trigger script
- 3.- Trigger owner

The index of those columns can be used with the `--format` argument to print customized outputs, like in this sample:

```
cm listtriggers before-mklabel --format="{0} = {2}"
1 = c:\tmp\triggers\validate-label.bat
2 = c:\tmp\triggers\loglabels.bat
```

If no trigger type is provided, the `listtriggers` command will list the whole list of triggers present on the server.

4.2.4 Change trigger

Once a trigger has been created, its options can be altered without recreating it through the `changetrigger` command. The syntax is:

```
cm changetrigger {type} {existing-trigger-position}
  [--position=value]
  [--name=value]
  [--script=value]
  [--server=server:port]
```

Where:

- Type is the trigger type to get the list of associated scripts, as listed in the table of section "List of triggers". This argument is required.
- Existing trigger position is the index by which the trigger is referred to in the list of triggers associated to the trigger type. This value, together with the trigger type, uniquely identifies the script to be edited. This argument is required.

- Position is the new position for the trigger in the trigger list. Note that the destination position must not be used by another trigger or an error will be raised. This argument is optional.
- Name is the new name for the trigger. Note that the name is used only for readability. This argument is optional.
- Script is the new path for the script or program to run. The same restrictions that were described for trigger creation would apply here. This argument is optional.
- Server is the server in which to edit the trigger. If omitted, the trigger will be edited on the default configured server. The syntax specifies a server's host name, ':', and a port, by default 8084.

Sample for changing a trigger's name and target script:

```
cm listtriggers before-checkin
1 checkstyle c:\tmp\triggers\checkin-checkstyle.bat dave

cm changetrigger before-checkin 1 --name="codestyle"

cm listtriggers before-checkin
1 codestyle c:\tmp\triggers\checkin-checkstyle.bat dave
```

4.2.5 Remove trigger

Triggers can be removed from Plastic. Removing a trigger does not remove the associated trigger script or program on the file system, it simply instructs Plastic not to execute the script anymore. This is the syntax of the command:

```
cm removetrigger {type} {existing-trigger-position}
[--server=server:port]
```

Where:

- Type is the trigger type to get the list of associated scripts, as listed in the table of section "List of triggers". This argument is required.
- Existing trigger position is the index by which the trigger is referred in the list of triggers associated to the trigger type. This value, together with the trigger type, uniquely identifies the script to be removed. This argument is required.

Example:

```
cm listtriggers before-mklabel
1 Validate label c:\tmp\triggers\validate-label.bat dave
2 log labels c:\tmp\triggers\loglabels.bat dave

cm rmtrigger before-mklabel 2

cm listtriggers before-mklabel
1 Validate label c:\tmp\triggers\validate-label.bat dave
```

4.3 Trigger communication

4.3.1 Input

Plastic SCM will send information to the trigger script related to the executing operation. Two approaches will be used:

- Standard input: usually object references like revision specs involved in the triggered operation will be pushed to the trigger script using the standard input.
- Environment variables: general information, like the Plastic user which started the operation or the client machine. For a detailed description of the variables used in a given operation, check the specific operation in the reference below.

4.3.2 Output

The trigger script will communicate the result of its execution using the result code. Plastic will interpret these result codes:

- Zero (0): trigger finished correctly, operation can continue.
- Non-zero (>0): trigger failed, the operation cannot continue.

If the result is non-zero for a 'before' trigger, the operation is cancelled, and the error is displayed on the client.

If the result is non-zero for an 'after' trigger, the operation has been already performed and can't be rolled back, however the exception is displayed on the client too.

When a trigger fails (error code is non-zero) the standard output of the trigger will be sent to the client as an error message.

4.3.3 Common environment variables

This table details the environment variables that are available for every trigger script:

PLASTIC_USER	The user who started the operation in the client.
PLASTIC_CLIENTMACHINE	The client machine that started the operation.
PLASTIC_SERVER	The hostname of the Plastic server.

4.3.4 Server.conf variables

Variables can be defined on the server.conf file. Their value will be passed to the trigger script or program as environment variables. In order to define these variables, a section called 'TriggerVariables' needs to be added to the server.conf file available in the server installation folder. The following example shows a possible use of this file:

```
<?xml version="1.0"?>
<ServerConfigData>
  <Language>en</Language>
  <WorkingMode>UPWorkingMode</WorkingMode>
  <ServerType>ServerTypeAll</ServerType>

  <TriggerVariables>
    <TriggerVariable name="TRIGGERS_PATH" value="c:\triggers" />
  </TriggerVariables>
</ServerConfigData>
```

This sample defines a variable called 'TRIGGERS_PATH' with value 'c:\triggers'. This variable can be used when creating a trigger, in the 'script' field, like in this example:

```
cm createtrigger before-checkin "code checker"  
"@TRIGGERS_PATH\stylecheck.bat"
```

Note '@' to refer to the variable in this context. The variable is also passed as an environment variable to the trigger script, so it could be used also inside the script itself like the following:

```
@echo off  
  
set OUTPUT_FILE=%TRIGGERS_PATH%\label.log.txt  
  
echo %PLASTIC_REPOSITORY_NAME% %PLASTIC_LABEL_NAME% >> %OUTPUT_FILE%  
  
exit 0
```

4.4 Detailed trigger reference

The following sections will provide a detailed reference of all the triggers as well as input and output parameters. Samples are provided for the most common actions.

4.4.1 Add

4.4.1.1 Trigger names

```
before-add  
after-add
```

4.4.1.2 Description

It executes user scripts when items are added to the source control.

4.4.1.3 Runs on

Server.

4.4.1.4 Standard input

A list of files to be added.

4.4.1.5 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The branch will be created.
Non zero	The trigger reports failure. If the trigger is before-add, an error is reported and the items are not added to the repository. If the trigger is after-add an error is reported. However the items have already been added to the repository.

4.4.1.6 Environment variables

In addition to the variables defined in sections “Common environment variables” and “Server.conf variables”, these are also available:

PLASTIC_COMMENT	The comment given by the user on the add operation.
-----------------	---

4.4.2 Check in

4.4.2.1 Trigger names

before-checkin
after-checkin

4.4.2.2 Description

It executes user scripts when a check-in is performed on any client.

4.4.2.3 Comments

The before and after check-in triggers are invoked only once for all the items involved in the check-in. The standard input of the trigger will receive a list of the items involved.

This is one of the most complex and useful triggers. Some examples of usages: checking code before it is checked in on the repository against some validation / formatting tool or sending emails / updating rss feeds when new code is in the repository.

Revision contents can be accessed through the ‘cm cat’ command, specifying the revision specification supplied in the standard input. They can be validated, modified and then stored back into the server with the ‘cm shelve’ command. In case a ‘shelve’ command updates the contents of a revision in the repository, the client performing the checkin operation **will automatically update those items** so the contents of the workspace are always correct.

4.4.2.4 Runs on

Server.

4.4.2.5 Standard input

The standard input receives revision identifiers for all the items involved in the checkin operation, one per line. Each of them is a specially formatted string, containing the server’s path of the item (independent of any workspace) and the revision specification, so its contents are easily retrieved using the ‘cm cat’ command.

This is the format of the revision specifications, one per line:

<i>item_path#br:branch#rev_no;rev_id@rep:rep_name@repserver:server</i>
--

The meaning of the members in italic is detailed in the following table

item_path	The revision’s path in server format , which is independent of the client workspace and operating system.
branch	The branch of the revision.

rev_no	The revision number, inside the branch, of the revision.
rev_id	The revision unique identifier. Can be used to ease parsing when accessing revisions with 'cm cat' or 'cm shelve' in the trigger script, as string after the first semicolon uniquely identifies the revision inside the server.
rep_name	The repository name where the revision belongs.
rep_server	The repository server where the repository belongs.

This example shows standard input supplied to a checkin trigger:

```
/code/clean.bat#br:/main#CO;revid:12936@rep:default@repserver:DARKTOWER:8084;wk:trigger_test@DARKTOWER
```

4.4.2.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. Items will be checked in to the repository.
Non zero	The trigger reports failure. If the trigger is before-checkin, the checkin operation is stopped and the items are not checked in, neither changeset is created. An error message is reported to the client. If the trigger is after-checkin an error message is reported to the client. However the checkin operation has already been performed.

4.4.2.7 Environment variables

In addition to the variables defined in sections "Common environment variables" and "Server.conf variables", these are also available:

PLASTIC_COMMENT	The comment given at checkin time by the user.
PLASTIC_CHANGESET	The changeset or changesets that were created as a result of the checkin operation. Note that this variable is only available in the 'after-checkin' trigger. See notes below for the format of this variable.

The PLASTIC_CHANGESET variable contains the specifications for the changesets that were created, separated by semi-colons (;). This is a sample of a variable value with changesets created on two different repositories:

```
cs:23@br:/main@rep:default@repserver:DARKTOWER:8084;  
cs:19@br:/main@rep:secondrep@repserver:DARKTOWER:8084
```

4.4.2.8 Sample command line creation

```
cm mktrigger checkin-before "checkstyle" "c:\tmp\triggers\checkin-  
checkstyle.bat"  
Trigger created on position 1.
```

4.4.2.9 Sample trigger script

The following script simply reads all the standard input and redirects it to the 'c:\tmp\triggers\checkout.txt' file. The trick here is the use of the find.exe command in order to read the standard input in Windows command line 'cmd.exe':

```
@echo off  
  
for /f "tokens=*" %%g in ('find /V ""') do (  
    echo %%g >> c:\tmp\triggers\checkout.txt  
)  
  
exit 0
```

4.4.3 Check out

4.4.3.1 Trigger names

```
before-checkout  
after-checkout
```

4.4.3.2 Description

It executes user scripts when a check-out operation is performed on any client.

4.4.3.3 Comments

The before and after check-out triggers are invoked once for every item involved in the check-out. The standard input of the trigger will receive a list of the items involved.

4.4.3.4 Runs on

Server.

4.4.3.5 Standard input

The standard input receives revision identifiers for all the items involved in the checkout operation, one per line. Each of them is a specially formatted string, containing the server's path of the item (independent of any workspace) and the revision specification, so its contents are easily retrieved using the 'cm cat' command.

This is the format of the revision specifications, one per line:

```
item_path#br:branch#rev_no;rev_id@rep:rep_name@repserver:server
```

The meaning of the members in italic is detailed in the following table

<i>item_path</i>	The revision's path in server format , which is independent of the client workspace and operating system.
------------------	--

branch	The parent branch.
rev_no	The parent revision.
rev_id	The revision unique identifier. Can be used to ease parsing when accessing revisions with 'cm cat' or 'cm shelve' in the trigger script, as string after the first semicolon uniquely identifies the revision inside the server.
rep_name	The repository name where the revision belongs.
rep_server	The repository server where the repository belongs.

Sample standard input supplied to a checkout trigger:

```
/code/clean.bat#br:/main#1;revid:12936@rep:default@repserver:DARKTOWER:80
84;wk:trigger_test@DARKTOWER
```

4.4.3.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. Items will be checked out.
Non zero	<p>The trigger reports failure.</p> <p>If the trigger is before-checkout, the operation is stopped and the items are not checked out. An error message will reported to the client.</p> <p>If the trigger is after-checkout an error message is reported to the client. However the checkout operation has already been performed.</p>

4.4.3.7 Environment variables

In addition to the variables defined in sections "Common environment variables" and "Server.conf variables", the following variable is also available:

PLASTIC_COMMENT	The comment given by the user when the checkout operation is done.
-----------------	--

4.4.3.8 Sample command line creation

```
cm mktrigger checkout-before "checkstyle" "c:\tmp\triggers\checkout-
checkstyle.bat"
Trigger created on position 1.
```

4.4.3.9 Sample trigger script

The following script simply reads all the standard input and redirects it to the 'c:\tmp\triggers\checkout.txt' file. The trick here is the use of the find.exe command in order to read the standard input in Windows command line 'cmd.exe':

```
@echo off

for /f "tokens=*" %%g in ('find /V ""') do (
    echo %%g >> c:\tmp\triggers\checkout.txt
)

exit 0
```

4.4.4 Create branch

4.4.4.1 Trigger names

```
before-mkbranch

after-mkbranch
```

4.4.4.2 Description

It executes user scripts when a branch is created.

4.4.4.3 Comments

These triggers fire on branch creation.

4.4.4.4 Runs on

Server.

4.4.4.5 Standard input

No standard input is supplied to these triggers.

4.4.4.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The branch will be created.
Non zero	The trigger reports failure. If the trigger is before-mkbranch, an error is reported and the branch is not created. If the trigger is after-mkbranch an error is reported. However the branch has already been created.

4.4.4.7 Environment variables

In addition to the variables defined in sections "Common environment variables" and "Server.conf variables", these are also available:

PLASTIC_COMMENT	The comment given by the user at branch creation.
-----------------	---

PLASTIC_BRANCH_NAME	The branch that is being created.
PLASTIC_REPOSITORY_NAME	The repository name where the branch is being created.

4.4.5 Create label

4.4.5.1 Trigger names

```
before-mklabel
after-mklabel
```

4.4.5.2 Description

It executes user scripts when a label is created.

4.4.5.3 Comments

This trigger fires on label creation.

4.4.5.4 Runs on

Server.

4.4.5.5 Standard input

No standard input is supplied to these triggers.

4.4.5.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The label will be created.
Non zero	The trigger reports failure. If the trigger is before-mklabel, an error is reported and the label is not created. If the trigger is after-mklabel an error is reported. However the label has already been created.

4.4.5.7 Environment variables

In addition to the variables defined in sections "Common environment variables" and "Server.conf variables", these are also available:

PLASTIC_COMMENT	The comment given by the user at label creation.
PLASTIC_LABEL_NAME	The label that is being created.
PLASTIC_REPOSITORY_NAME	The repository name where the label is being created.

4.4.5.8 Sample command line creation

```
cm maketrigger before-mklabel "validate label"  
"c:\plastic\triggers\Validate-label.bat"  
Trigger created on position 1.  
  
cm listtriggers before-mklabel  
2 Validate label c:\tmp\triggers\validate-label.bat dave
```

4.4.5.9 Sample trigger script

The following script saves a record of created branches on the c:\plastic\triggers\labels.log.txt file.

```
@echo off  
  
echo %PLASTIC_REPOSITORY_NAME% %PLASTIC_LABEL_NAME% >>  
c:\plastic\triggers\labels.log.txt  
  
exit 0
```

4.4.6 Create attribute

4.4.6.1 Trigger names

```
before-mkattribute  
  
after-mkattribute
```

4.4.6.2 Description

Executes user scripts when an attribute is created

4.4.6.3 Comments

These triggers fire on attribute creation.

4.4.6.4 Runs on

Server.

4.4.6.5 Standard input

No standard input is supplied to these triggers.

4.4.6.6 Output result

The result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The attribute will be created.
Non zero	The trigger reports failure. If the trigger is before-mkattribute, an error is reported and the branch is not created. If the trigger is after-mkattribute an error is reported. However

	the branch has already been created.
--	--------------------------------------

4.4.6.7 Environment variables

In addition to the variables defined in sections “Common environment variables” and “Server.conf variables”, these are also available:

PLASTIC_COMMENT	The comment given by the user at attribute creation.
PLASTIC_ATTRIBUTE_NAME	The attribute that is being created.
PLASTIC_REPOSITORY_NAME	The repository name where the attribute is being created.

4.4.7 Create repository

4.4.7.1 Trigger names

```
before-mkrepository
after-mkrepository
```

4.4.7.2 Description

It executes user scripts when a repository is created

4.4.7.3 Comments

These triggers fire on repository creation.

4.4.7.4 Runs on

Server.

4.4.7.5 Standard input

No standard input is supplied to these triggers.

4.4.7.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The repository will be created.
Non zero	The trigger reports failure. If the trigger is before-mkrepository, an error is reported and the repository is not created. If the trigger is after-mkrepository an error is reported. However the repository has already been created.

4.4.7.7 Environment variables

In addition to the variables defined in sections “Common environment variables” and “Server.conf variables”, these are also available:

PLASTIC_REPOSITORY_NAME	The repository name where the attribute is being created.
-------------------------	---

4.4.8 Create workspace

4.4.8.1 Trigger names

<code>before-mkworkspace</code>
<code>after-mkworkspace</code>

4.4.8.2 Description

It executes user scripts when a workspace is created

4.4.8.3 Comments

These triggers fire on workspace creation.

4.4.8.4 Runs on

Client.

4.4.8.5 Standard input

No standard input is supplied to these triggers.

4.4.8.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The workspace will be created.
Non zero	The trigger reports failure. If the trigger is before-mkworkspace, an error is reported and the workspace is not created. If the trigger is after-mkworkspace an error is reported. However the workspace has already been created.

4.4.8.7 Environment variables

In addition to the variables defined in sections “Common environment variables” and “Server.conf variables”, these are also available:

PLASTIC_WORKSPACE_NAME	The name given to the new workspace.
PLASTIC_WORKSPACE_PATH	The path of the workspace on the client machine.

4.4.9 Set selector

4.4.9.1 Trigger names

```
before-setselector  
after-setselector
```

4.4.9.2 Description

Executes user scripts when a workspace selector is changed.

4.4.9.3 Comments

Selectors are modified either with the 'setselector' command or with the 'Switch workspace to branch / label' commands in both the command line interface or the GUI client.

4.4.9.4 Runs on

Client.

4.4.9.5 Standard input

The standard input in these triggers receives the selector contents that are being set by the client.

4.4.9.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. The selector will be created.
Non zero	The trigger reports failure. If the trigger is before-setselector, an error is reported and the selector is not modified. If the trigger is after-setselector an error is reported. However the operation has already been done.

4.4.9.7 Environment variables

In addition to the variables defined in sections "Common environment variables" and "Server.conf variables", these are also available:

PLASTIC_WORKSPACE_NAME	The name of the workspace which selector is being set.
PLASTIC_WORKSPACE_PATH	The client path of the workspace which selector is being set.

4.4.10 Update

4.4.10.1 Trigger names

```
before-update  
after-update
```

4.4.10.2 Description

It executes user scripts when a workspace is updated.

4.4.10.3 Comments

This trigger runs on the client, so the script locations are relative to the client machine filesystems. This is an important difference to note when creating this kind of triggers.

4.4.10.4 Runs on

Client.

4.4.10.5 Standard input

No standard input is supplied to these triggers.

4.4.10.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. Update operation will proceed.
Non zero	The trigger reports failure. If the trigger is before-update, an error is reported and the workspace is not updated. If the trigger is after-update an error is reported. However the workspace has already been updated.

4.4.10.7 Environment variables

PLASTIC_USER	The user who started the update operation.
PLASTIC_CLIENTMACHINE	The client machine in which the operation was started.
PLASTIC_UPDATE_PATH	The client path of the workspace being updated.

4.4.11 Client checkout

4.4.11.1 Trigger names

```
before-clientcheckout  
after-clientcheckout
```

4.4.11.2 Description

It executes user scripts on the client when a checkout operation is executed.

4.4.11.3 Comments

This trigger runs on the client, so the script locations are relative to the client machine filesystems. This is an important difference to note when creating this kind of triggers.

4.4.11.4 Runs on

Client.

4.4.11.5 Standard input

The standard input in these triggers receives the list of items specified by the user on the checkout operation.

4.4.11.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. Checkout operation will proceed.
Non zero	The trigger reports failure. If the trigger is before-clientcheckout, an error is reported and the operation is cancelled. If the trigger is after-clientcheckout, an error is reported. However the items have been already checked out.

4.4.11.7 Environment variables

PLASTIC_USER	The user who started the checkout operation.
PLASTIC_CLIENTMACHINE	The client machine in which the operation was started..
PLASTIC_COMMENT	The comment specified by the user on the checkout operation

4.4.12 Client checkin

4.4.12.1 Trigger names

```
before-clientcheckin  
after-clientcheckin
```

4.4.12.2 Description

It executes user scripts on the client when a checkin operation is executed.

4.4.12.3 Comments

This trigger runs on the client, so the script locations are relative to the client machine filesystems. This is an important difference to note when creating this kind of triggers.

4.4.12.4 Runs on

Client.

4.4.12.5 Standard input

The standard input in these triggers receives the list of items specified by the user on the checkin operation.

4.4.12.6 Output result

Result code from the trigger script or executable determines the success or failure of the operation:

0	The trigger completed successfully. Checkin operation will proceed.
Non zero	The trigger reports failure. If the trigger is before-clientcheckin, an error is reported and the checkin operation is aborted. If the trigger is after-clientcheckin an error is reported. The items are already checked in.

4.4.12.7 Environment variables

PLASTIC_USER	The user who started the checkin operation.
PLASTIC_CLIENTMACHINE	The client machine in which the operation was started.
PLASTIC_COMMENT	The comment specified by the user on the checkin operation

5 Samples

All the samples in this section can be found on the 'triggers' folder, within the server installation folder.

5.1 Checkin

5.1.1 Apply code beautifier to .java files

This sample will process all java files through a code beautifier (jindent used here, replace with your favorite tool). Script is written in Ruby.

```
#!/usr/bin/env ruby

# temp file that will be used for jindent
tmpfile = "c:\\tmp\\triggers\\trigger-validate.java"

# Process each line of stdin
STDIN.readlines.each_with_index do |line, index|

  # split into item, revspec and wkspec
  splitted = line.split(';')

  # pick item name from item spec
  filename = splitted[0].split('#')[0]

  # if it is a .java file, apply jindent
  if (filename =~ /\.java$/) then

    # revspec is after the first ;
    revspec = splitted[1];

    # extract revision content from repository to temp file
    res = system("cm cat #{revspec} --file=\"#{tmpfile}\"")

    # execute jindent on temp file (jindent should be on path)
    if (res) then res = system("jindent \"#{tmpfile}\"") end

    # if jindent failed, signal the trigger failed too
    if (!res || $? != 0) then exit(1) end

    # store the re-formatted file on Plastic repository
    if (res) then system("cm shelve #{revspec} --file=\"#{tmpfile}\"")
  end

  # delete the temp file
  if (res) then system("del \"#{tmpfile}\"") end

end #if

end #each
```

Sample trigger creation command (on Windows)

```
cm maketrigger before-checkin "apply jindent" "ruby
c:\\triggers\\jindent.rb"
```

5.1.2 Apply modifying action to items in block

This is the same example as before, but now all involved files are 'cat' and 'shelved' in block, greatly improving performance.

```
#!/usr/bin/ruby
tmpdir = 'c:\\tmp\\triggers\\'

$files = []
$cat_shelve_specs = []

# Apply command sending revision info
def commandOnSpecs(cmd)
  IO.popen(cmd, "w") do |io|
    $cat_shelve_specs.each do |spec|
      puts 'catting ' + spec
      io.puts spec
    end
  end
end

# Process stdin
STDIN.readlines.each do |line|
  itemspec, revspec, wkspec = line.split(';')
  filename, branchspec, revno = itemspec.split('#')

  # this may have problems with long paths
  filename.gsub!(/\\//, '_') # replace / with _ in filenames
  filename = tmpdir + filename # add tmpdir

  $files << filename
  $cat_shelve_specs << "#{revspec};#{filename}"
end

# cat files on temp directory
commandOnSpecs("cm cat -")

# Apply action on files
$files.each { |file| system("jindent \"#{file}\"") }

# shelve files
commandOnSpecs("cm shelve -")

# remove temp files
$files.each { |file| File.delete file }
```

Sample trigger creation command (on Windows)

```
cm maketrigger before-checkin "apply block jindent" "ruby
c:\\triggers\\jindent.rb"
```

5.1.3 Check that comments have been provided on checkin

Sample ruby script that checks the PLASTIC_COMMENT environment variable:

```
c = ENV['PLASTIC_COMMENT']  
if (c == nil || c == '') then exit(1) end
```

Sample trigger creation command:

```
cm maketrigger before-checkin "comment required" "ruby c:\triggers\check-  
comments.rb"
```

5.1.4 Generate rss with changeset contents

This ruby trigger combines all the techniques shown in the previous sections to provide a script capable of updating a .rss file which can be used by rss aggregators to notify new changes on the repository.

```
require 'rss/2.0'
require 'open-uri'
require 'rss/maker'

targetfile = "Z:\\cm\\tts\\plastic-changesets.rss"

# Read content if available

if (File.exists?(targetfile)) then
  content = ""
  open(targetfile) do |s| content = s.read end
  rss = RSS::Parser.parse(content, false)
else
  rss = RSS::Rss.new("2.0")
  channel = RSS::Rss::Channel.new
  channel.title = "Plastic updates feed"
  channel.link = http://www.plasticscm.com
  channel.description = ""
  channel.language = "en"
  rss.channel = channel
end

# Parse checkin item names from plastic

files = ''
STDIN.readlines.each do |line|
  itemspec, revspec, wkspec = line.split(';')
  filename, branchspec, revno = itemspec.split('#')
  files << filename << "<br/>"
end

# Add new rss item

item = RSS::Rss::Channel::Item.new
item.title = "#{ENV['PLASTIC_CHANGESET']} - #{ENV['PLASTIC_COMMENT']} by
#{ENV['PLASTIC_USER']}"
item.link = http://www.plasticscm.com
item.date = Time.now
item.description = files
rss.items << item

# Write the resulting file

File.open(targetfile, "w") do |f|
  f.write(rss)
end
```

This is an after-checkin trigger:

```
cm maketrigger after-checkin "generate rss" "ruby c:\triggers\rss-gen.rb"
```

5.2 Make Label

5.2.1 Validate that label name starts with 'release'

```
if (ENV['PLASTIC_LABEL_NAME'] !~ /^release/) then exit(1) end
```

Trigger creation command:

```
cm maketrigger before-mklabel "check label name" "ruby  
c:\plastic\triggers\validate-label.rb"
```

5.3 Client checkout

5.3.1 Update files before checkout

The ruby script would be as follows:

```
#!/usr/bin/ruby  
  
files = ''  
  
STDIN.readlines.each do |line|  
  files << " " << line  
end  
  
system("cm update #{files}")
```

Trigger creation command:

```
cm maketrigger before-clientcheckout "update before co" "ruby  
c:\plastic\triggers\update-before-co.rb"
```