Dipl.-Ing. Frank Sell
System Engineer Unix

: www.xpile.de
: fsell@xpile.de

Tips und Kniffe

im Umgang mit ClearCase

Dokument: clearcase_tips

Erstellungsdatum: 19. Oktober 2004
Letzte Bearbeitung: 14. Januar 2014
Version: 57

Inhaltsverzeichnis

1 Einleitung

Versionsverwaltungen sind bekanntlich ein unverzichtbarer Bestandteil einer jeden Entwicklungsarbeit. Sie gewährleisten, dass alle Versionen eines Files im Repository gesichert sind und mit Hilfe einer konkreten Sicht auch wieder verfügbar sind.

Eine der zur Zeit gängigsten Versionsverwaltungen ist derzeit ClearCase.

Nachfolgender Text beschäftigt sich in einer losen Sammlung mit Aspekten rund um ClearCase.

2 Erzeugen von Views

Views sind eigentlich nur Container, die Platz für eine Sicht auf das ClearCase-Repository bereitstellt. Was letztendlich gesehen wird, bestimmt die Config-Spec.

Views müssen explizit angelegt werden.

Dazu dient folgendes Kommando:

cleartool mkview -tag <viewname> path_zum_storage/<viewname.vws>

In der Folge geht es um den Pfad zum View-Storage.

Bei normalen Installationen kann das View-Storage in einem beliebigen Verzeichnis liegen. Folgende Einschränkungen gelten dabei:

- Das Verzeichnis muss schreib bar sein
- Das Volume, in dem das Verzeichnis existiert muss lokal gemountet sein

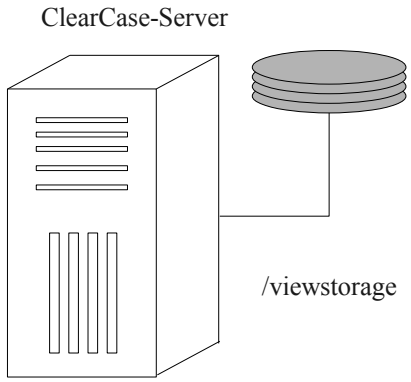

Bild 1: Beispiel für ein lokales Volume

oder

- Das Verzeichnis muss schreib bar sein.
- Das Verzeichnis muss per NFS von einer Maschine kommen, auf der ebenfalls ClearCase installiert ist und das exportierte Volume ist lokal.

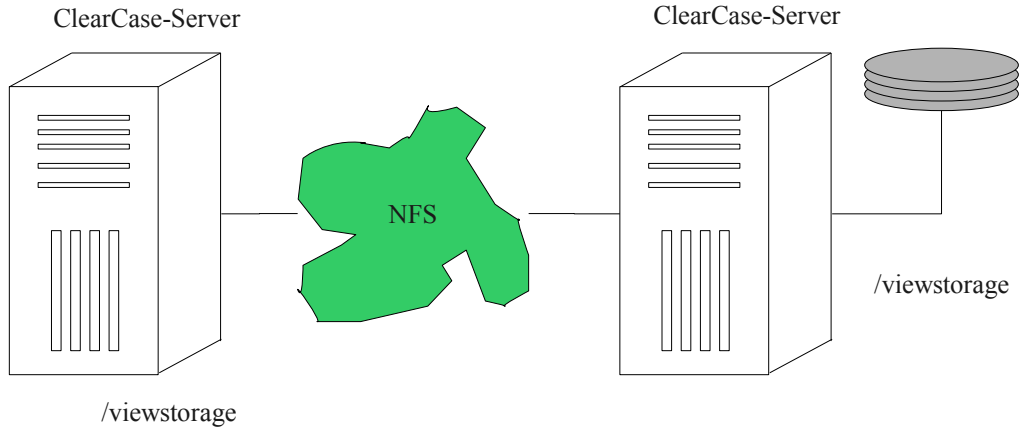

Bild 2: Beispiel für ein per NFS gemountetes Volume

Anders sieht es aus, wenn das Volume für das Storage nicht physikalisch auf den ClearCase-Servern liegt. Diese NAS-Volume müssen im ClearCase bekannt gemacht werden.

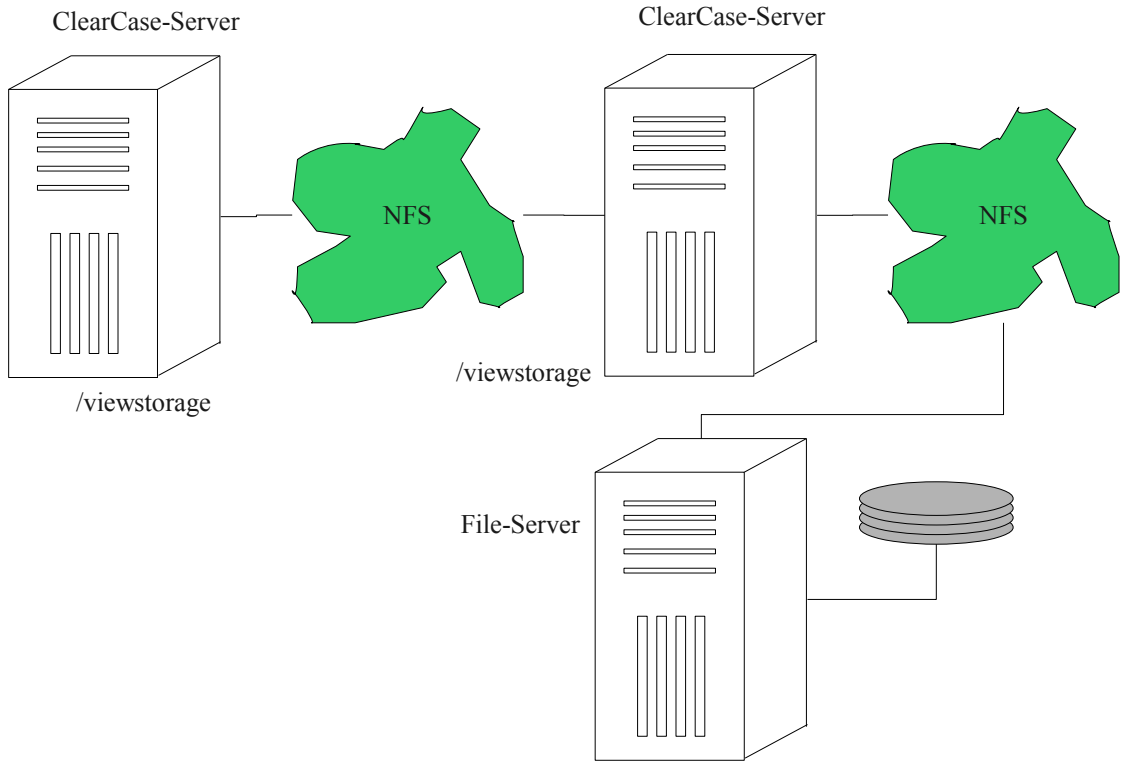

Bild 3: Beispiel für ein NAS Volume

Dabei ist folgendes zu beachten:

- der NFS-Mount des Servers, der das NAS-Device mountet, muss hard sein
- alle weiteren Mounts können wieder soft sein.

Das NAS-Volume wird folgendermaßen bekannt gemacht:

```
cleartool mkstgloc -view -host cc-Server -gpath /viewstorage -hpath /viewstorage \
ccnasviewstg /viewstorage
```

cc-Server ist hierbei der Server, der das NAS-Device importiert.

Die Views müssen jetzt auf andere Weise angelegt werden:

```
cleartool mkview -tag <viewname> -stgloc ccnasviewstg
oder
cleartool mkview -tag <viewname> -stgloc -auto
```

3 Programme und Scripte unter Versionskontrolle halten

Nützliche Programme und Scripte (keine Standard-Software) sollten unbedingt einer Versionskontrolle unterliegen.

Das bedeutet in der Regel, die betreffenden Teile in die Versionsverwaltung aufzunehmen und dann auf den Maschinen, auf welchen sie verfügbar sein sollen, zusätzlich zu installieren.

Eleganter und weniger fehlerbehaftet ist es, folgenden Weg einzuschlagen:

- als root einen View definieren
  (z.B. cleartool mkview -tag support_<host> /var/share/views/support_<host>.vws
- geeignete Config-Spec definieren
  (z.B. „element * /main/LATEST -nocheckout“)
- diesen View über die Start-Up-Scripte der Maschine starten
  (z.B. ..... cleartool startview support_<host> ...)
- Link erzeugen
  (z.B. ln -s /view/support_<host>/vob/... /usr/tools)

Natürlich ist dieses Vorgehen nur für Maschinen sinnvoll, auf denen auch ClearCase installiert ist.

In den obigen Beispielen wird ein View pro Maschine eingesetzt. Durch geschickte Gestaltung der Config-Spec kann eine individuelle Sicht für jede Maschine auf die versionierten Programme / Scripte eingestellt werden.

Natürlich kann auch nur ein View Verwendung finden. Dazu muss allerdings das Viestorage auf einem von allen Maschinen sichtbaren Verzeichnis liegen. Weiterhin gilt die eingestellt Config-Spec für alle Maschinen.

Durch diesen Trick ist es möglich, Software in Bereichen zur Verfügung zu stellen, in denen normale User keine Schreib rechte haben. Der verlinkte View hat keinerlei ausgecheckten Elemente. Damit ist auch alles schreibgeschützt.

Trotzdem ist es möglich neue Software verfügbar zu machen, ohne 'root' zu bemühen.

- View mit passender Config-Spec erzeugen
- entsprechendes Element auschecken und bearbeiten
- Nach checkin werden die Änderungen auf den Maschinen sichtbar

4 Nützliche Trigger im Einsatz mit ClearCase

ClearCase bietet den Einsatz von Triggern an, um bestimmte Aktionen in der Versionsverwaltung mit zusätzlichen Funktionen zu hinterlegen.

Folgende Funktionalitäten werden als nützlich erachtet:

Funktionalität	**ClearCase-Aktion**
Leerzeichen in Element-Namen abweisen (automatische Bearbeitung mittels Scripte gestaltet sich durch fehlendes Quoting wesentlich einfacher)	lnname
Erzeugen von Elementen, welche bereits in anderen Branches vorliegen, abweisen	mkelem, lnname
Leere Branches entfernen	rmver, rmbranch, uncheckout
Destroy von Elementen unterbinden	rmelem
RCS-Keywords ersetzen (siehe nächstes Kapitel)	checkout, checkin

Günstiger weise fasst man die ersten beiden Funktionalitäten in einem Trigger zusammen. Allerdings sollte die Prüfung auf Elemente in anderen Branches für Links (cleartool ln [-s] ...) aufgeweicht werden. Links können auf voll qualifiziert werden. Dann macht es wenig Sinn, immer neue Namen vergeben zu müssen.

Damit ergeben sich drei Trigger, welche pro Vob zu installieren sind.

5 ClearCase und Identifikation ausgelieferter Dateien

Im Workflow einer Softwareentwicklung klafft eine Lücke, die in der realen Welt nicht richtig dargestellt werden kann.

Diese Lücke betrifft die Schnittstelle Versionsverwaltung und Installations-Umgebung.

Mit den heutigen Tools ist es mittlerweile möglich den Entwicklungsfluss von der Beauftragung bis hin zur Fertigmeldung lückenlos zu erfassen und nachvollziehbar zu machen.

Produktive Systeme oder große Test-Umgebungen sind aber in der Regel von den Entwicklungs-Umgebungen und damit von der Versionsverwaltung abgekoppelt.

Damit stellt ein neuer Release-Stand in der Versionsverwaltung nicht den 'Produktionsstand' dar, sondern bestenfalls den Stand 'zur Produktion vorgesehen'. Das heißt, der gelieferte Release-Stand kann korrekt installiert sein, muss aber nicht.

In Bezug auf die Qualitätssicherung ist es sehr hilfreich, anhand von in die ausgelieferten Dateien eingelagerten Informationen, eindeutig die Version der Datei in der Versionsverwaltung identifizieren zu können.

5.1 Einsatz von Clearmake

Clearmake ist in der Lage, die zum Build benutzte Config-Spec und diverse andere Informationen im Compilat zu verankern.

Leider wird bei der Auslieferung von z.B. Shell-Scripts oder Text-Dokumente kein Build angestoßen.

Damit erübrigt sich hier eine weitere Betrachtung.

5.2 Einsatz von RCS-Keywords

RCS-Keywords sind ein gutes Mittel um ausgelieferte Dateien zu identifizieren.

Um die in den RCS-Keywords verankerten Informationen verfügbar zu machen, bieten sich Tools wie z.B. 'what' oder 'ident'.

Das Tool 'ident' durchsucht eine Datei nach dem Auftreten von Keywords, wie sie durch die Versionsverwaltungen 'RCS' und 'CVS' in der Form: '$keyword: text $'definiert sind.

Das Tool 'what' arbeitet ähnlich. Allerdings sucht dieses Programm nach dem Vorkommen des SCCS-Pattern '@(#)'.

Da 'what' und 'ident' nicht auf allen Umgebungen verfügbar sind, sollten beide o.g. Pattern in den zu versionierenden Files verankert werden.

Leider unterstützt ClearCase von Hause aus die Substitution der Pattern nicht.

Man kann sich aber mit einem Trigger behelfen. Dieser Trigger muss dann nach dem Checkout und vor dem Checkin der entsprechenden Files tätig werden. Dieser Trigger hat dann die Aufgabe, den RCS-String entsprechend anzupassen.

Lösungen sind bereits verfügbar bzw. können leicht implementiert werden.

Für brauchbare Information, bieten sich folgende RCS-Keywords an:

$RCSfile$	Name des Files ohne Pfadanteil
$Source$	Name des Files mit Pfadanteil
$Revision$	Revisionnummer des Files
$Header$	Voller Pfadname des Files, Revisionsnummer und diverse andere Angaben
Id	Nur Name des Files, Revisionsnummer und diverse andere Angaben

Für eine eindeutige Identifikation sollten die Keywords: '$Source$' und '$Revision$' preferiert werden. Somit ergibt sich dann folgender Keyword-String:

```
"@(#) $Source$ $Revision$"
```

Das nachfolgende Beispiel für eine C-Quelle 'bla.c' veranschaulicht die Funktionalität.

```
static char bla_c_rcsid[] = "@(#) $Source$ $Revision$";
```

wird z.B. expandiert zu

```
static char bla_c_rcsid[] = "@(#) $Source: /vob/src/bla.c $ $Revision: /main/2 $";
```

Natürlich kann man auf das Keyword $Revision$ verzichten und den Trigger so gestalten, dass er das Keyword $Source$ mit der Versionsinformation zu expandieren.

```
static char bla_c_rcsid[] = "@(#) $Source: /vob/src/bla.c@@/main/2 $";
```

Der Trigger kann auch ab prüfen, ob in dem einzucheckenden File die Keywords richtig eingetragen sind. Und wenn dem nicht so ist, kann der Checkin mit einer entsprechenden Fehlermeldung abgewiesen werden.

Wie im obigen Beispiel zu sehen ist, wird die statische Charakter-Variable nicht einfach nur 'rcsid' genannt. Sie sollte für jede Datei eindeutig benannt werden.

Damit werden von vornherein Konflikte z.B. mit Variablennamen includierter Headerfiles ausgeschlossen.

Für ein Shell-Script würde sich dann folgender Keyword-String anbieten:

```
# @(#) $Source$ $Revision$
```

Natürlich kann man nicht in jedem File RCS-Keywords verankern. Besonders Binaries vertragen dieses nicht besonders gut.

Deshalb ist es notwendig, automatisch zu entscheiden, ob ein Keyword gefordert ist.

Sinnvoll ist dieser Ersetzungsmechanismus z.B für C/C++-Files, Perl- und Shell-Scripte, sql-Files, Java-Sourcen etc.

Leider handelt man sich mit dem Einsatz von RCS-Keywords in ClearCase zwangsläufig Merge-Konflikte ein.

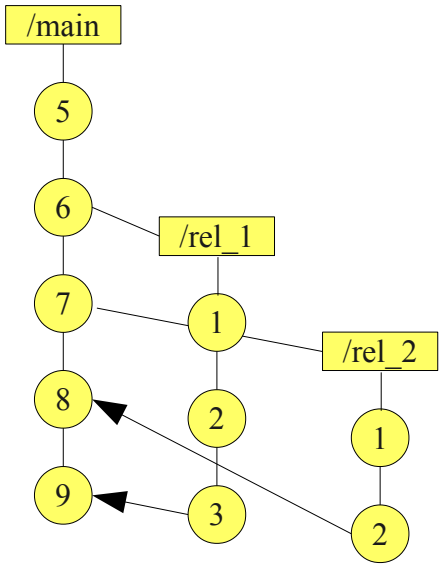

Bild 4: Mergebeispiel

Jeder der im obigen Beispiel abgebildeten Versionen beinhaltet einen anderen Versions-String. Da er in der Datei immer an der gleichen Stelle steht, kommt es beim Mergen zum Mergekonflikt und ein Automatischer Merge ist somit nicht mehr durchführbar.

Hier ist es unbedingt nötig, das normale Merge-Verhalten von ClearCase zu manipulieren. Ein Trigger für das Pre-Checkin und Post-Checkout hilft auch hier nicht wirklich.

Vielmehr muss hier der ClearCase-Typemanager manipuliert werden.

Dieser Typemanager kann für verschiedene Aktionen aufgerufen werden. An dieser Stelle ist es sinnvoll, einen angepassten Manager für die Aktionen 'merge' und 'xmerge' zu benutzen.

Dieser Manager bereinigt die RCS-Keywords vor dem eigentlichen Mergen der Dateien.

In gängigen Implementationen ist es so gelöst, daß sich der angepasste Manager von der Schnittstelle her wie der originale verhält. D.h. es werden alle Aufrufparameter vom ClearCase-System übernommen.

Anschließend wird das originale zu mergende File in ein temporäres File kopiert, wobei alle Keywords und ihre Erweiterungen bis auf sich selbst zurückgeführt werden. Anschließend kann der originale Merge-Manager aufgerufen werden. Dieser behandelt die zu mergenden Files in der gewohnten Art und Weise.

Der Merge-Manager darf nicht jede Art von Files in der oben beschriebenen Art und Weise behandeln.

Diese Verfahrensweise ist nur auf Files anzuwenden, welche auch durch die oben beschriebenen Trigger behandelt werden.

Um dieses Verhalten für bestimmte Files zu erreichen, sollte ein neuer File-Typ im entsprechenden Vob bekannt gemacht werden.

Z.B. ein Type 'keyed_text_file':

```
cleartool mkeltype -supertype text_file -manager keyed_text_file_delta \
-c "RCS keyword type" eltype:keyed_text_file@vob:<your-vob>
```

Der dazugehörige Manager 'keyed_text_file_delta' muss dann natürlich auch entsprechend

installiert werden.

Dazu wird im Verzeichnis '$ATRIAHOME/lib/mgrs' ein neues Verzeichnis 'keyed_text_file_delta' mit folgendem Inhalt angelegt:

```
manager-exe
annotate -> ../text_file_delta/annotate
construct_version -> ../text_file_delta/construct_version
create_branch -> ../text_file_delta/create_branch
create_element -> ../text_file_delta/create_element
create_version -> ../text_file_delta/create_version
delete_branches_versions -> ../text_file_delta/manager-exe
get_cont_info -> ../text_file_delta/manager-exe
compare -> manager-exe
merge -> manager-exe
xcompare -> manager-exe
xmerge -> manager-exe
```

Das File 'manager-exe' ist das Binary des angepassten Merge-Managers. Alle geforderten Funktionalitäten verweisen per Link auf ihn. Alle anderen verweisen auf die original-Manager im Verzeichnis '$ATRIAHOME/lib/mgrs/text_file_delta'

Der Type 'keyed_text_file' kann jetzt auch für bestimmte Files konfiguriert werden. Dazu verfügt ClearCase über einen Magic-File-Mechanismus, der anhand einer Konfigurationsdatei 'default.magic' den richtigen Filetype beim Anlegen eines Files im Repository ermittelt.

Die Datei '/$ATRIAHOME/config/magic/default.magic' kann dann beispielsweise wie folgt erweitert werden:

```
# Erweiterungen fuer die RCS Keywords
keyed_text_file text_file :
            -name "*.[sS][hH]"
          | -name "*.[kK][sS][hH]"
          | -name "*.[bB][aA][sS][hH]"
          | -name "*.[pP][lL]"
          | -name "*.[pP][mM]"
          | -name "*.[sS][qQ][lL]" | -name "*.[pP][lL][sS][qQ][lL]"
          | -name "*.[bB][aA][tT]"
          | -name "*.[tT][xX][tT]"
          | -name "*.[cC][bB][sS]"
          | -name "*.[hH]" | -name "*.[hH][pP][pP]"
          | -name "*.[cC]" | -name "*.[cC][pP][pP]" | -name "*.[eEpP][cC]"
          | -name "*.4[gG][lL]"
          | -name "*.[pP][eE][rR]"
          | -name "*.java" | -name "*.JAVA" ;
```

An dieser Stelle noch ein Hinweis:

Einige Programme, die im html- oder xml-Umfeld erzeugen Dateien mit Zeilenlänge > 8000 Charakter. Diese Dateien legt ClearCase in der Regel als 'text_file' an. Leider kann der dazu gehörende Mergemanager solche Files nicht mehr mergen. Es bietet sich an, solche Files zum Type 'compressed_file' zu konvertieren oder gleich in diesem Type anlegen zu lassen:

```
# generierte mdl-Files koennen durchaus mehr als 8000 Character in der
# Zeile haben
compressed_file : -name "*.mdl" ;
```

Die Einführung eines eigenen File-Types und des angepassten Mergemanagers schützt natürlich

nicht vor Merge-Konflikten an anderen Stellen. Aber die Wahrscheinlichkeit, das zwei Entwickler das File an genau der gleichen Stelle bearbeiten ist relativ gering.

Der weiter oben beschriebene Trigger zur Behandlung der RCS-Keywords kann folgendermaßen für Checkin- und Checkout-Aktionen installiert werden:

```
cleartool mktrtype -nc -replace -element -all -preop checkin -eltype keyed_text_file \
-execunix "/usr/atria/bin/Perl /<PATH-ZU-TRIGGER>/rcs_keywords.pl" \
-execwin "\\\\<maschine>\\PATH-ZU-PERL>\\ccperl.exe \\\\<maschine>\\<PATH-ZU-
TRIGGER>\\rcs_keywords.pl" \
RCS_KEYWORDS_PRE@@/vob/<vob-to-install>

cleartool mktrtype -nc -replace -element -all -postop checkout -eltype keyed_text_file \
-execunix "/usr/atria/bin/Perl /<PATH-ZU-TRIGGER>/rcs_keywords.pl" \
-execwin "\\\\<maschine>\\PATH-ZU-PERL>\\ccperl.exe \\\\<maschine>\\<PATH-ZU-
TRIGGER>\\rcs_keywords.pl" \
RCS_KEYWORDS_POST@@/vob/<vob-to-install>
```

Die Trigger werden bei den Aktionen 'chekout' bzw 'checkin' ausschließlich für Files vom Type 'keyed_text_file' aufgerufen.

Das vom Trigger aufgerufene Perlscript 'rcs_keywords.pl' führt die eigentliche Arbeit durch. Seine Implementation ist von den Anforderungen abhängig.

Wie an der Installation der Trigger zu sehen ist, müssen auch Windows-ClearCase-Clients mit den RCS-Keywords umgehen können.

Auf für Windows ist es nötig einen entsprechenden Merge-Manager zu installieren. Dieses Executable (z.B. manager.exe) ist in das \bin-Verzeichnis des Clients zu kopieren.

Weiterhin muss der Manager bekannt gemacht werden. Dieses erfolgt mittels einer Map-Datei (...\ClearCase\lib\mgrs\map). Die Datei muss folgende zusätzliche Einträge erhalten:

```
keyed_text_file_delta    construct_version          ..\..\bin\tfdmgr.exe
keyed_text_file_delta    create_branch              ..\..\bin\tfdmgr.exe
keyed_text_file_delta    create_element             ..\..\bin\tfdmgr.exe
keyed_text_file_delta    create_version             ..\..\bin\manager.exe
keyed_text_file_delta    delete_branches_versions   ..\..\bin\tfdmgr.exe
keyed_text_file_delta    compare                    ..\..\bin\cleardiff.exe
keyed_text_file_delta    xcompare                   ..\..\bin\cleardiffmrg.exe
keyed_text_file_delta    merge                      ..\..\bin\manager.exe
keyed_text_file_delta    xmerge                     ..\..\bin\manager.exe
keyed_text_file_delta    annotate                   ..\..\bin\tfdmgr.exe
keyed_text_file_delta    get_cont_info              ..\..\bin\tfdmgr.exe
```

Um nun auch den Windows-Client zu veranlassen, auch den richtigen Element-Typ bei neuen Files zu erzeugen, wird auch die angepasste Datei 'default.magic' benötigt.

Sie wird ins Verzeichnis ...\ClearCase\config\magic\ kopiert.