

# Introduction au gestionnaire de source Rational Clearcase


par Loïc Mathieu ([Accueil](#))

Date de publication : 13/03/2008

Dernière mise à jour : 13/03/2008

Cet article est une courte introduction à Rational Clearcase, un gestionnaire de source ('source control' ou 'version control' ou juste SCM en anglais). Rational ayant été racheté par IBM, les informations produits se trouvent sur son site (voir en section Liens). Utilisant la version 2003.06.10, l'article se base sur celle-ci.

- I - Introduction
- II - Les principes de base
  - II-A - Vocabulaire ClearCase
  - II-B - Les premiers pas
  - II-C - Autres fonctionnalités
  - II-D - Comparaison avec CVS et Subversion
- III - Les principaux outils
  - III-A - ClearCase Explorer
  - III-B - ClearCase Project Explorer
  - III-C - Cleartool
- IV - Intégration
  - IV-A - Dans eclipse
  - IV-B - Autres IDE
  - IV-C - Dans l'explorateur Windows
  - IV-D - Dans ANT
  - IV-E - Dans MAVEN
  - IV-F - Travail depuis Eclipse avec ClearCase
- V - Aller plus loin : les références entre projets
- VI - Liens
- VII - Remerciements

 *Certaines personnes utilisent peut-être la version LT qui est simplifiée et n'ont donc pas accès à toutes les fonctionnalités décrites ci-dessous. Il existe aussi une version multisite optimisée pour les équipes séparées géographiquement parlant, les fonctionnalités de celle-ci peuvent différer.*

## I - Introduction

ClearCase est reconnu comme un des SCM les plus puissants du marché, ayant travaillé avec CVS et Subversion je préfère largement ces derniers qui sont beaucoup moins lourds et plus rapide d'utilisation. Les possibilités de ClearCase sont très étendues, ce qui augmente énormément sa complexité. De plus, son mode de fonctionnement est assez éloigné de CVS et Subversion. Donc, un conseil, n'utilisez ClearCase que dans le cas où vous faites partie d'une équipe importante gérant un grand nombre de projets différents, c'est uniquement dans ce cas que ClearCase prendra tout son intérêt. Pour finir, il est conseillé de suivre une formation aux techniques de gestion et de merge de ClearCase pour en tirer entièrement profit.

Je vais ici vous faire une brève introduction à ClearCase en vous expliquant ses principes de base, son intégration dans les environnements de développement et ses outils pour finir sur une des fonctionnalités les plus puissantes: les références entre projets.

## II - Les principes de base

### II-A - Vocabulaire ClearCase

ClearCase a un vocabulaire très riche pour nommer ses différents composants et ses fonctionnalités. Je vais donc commencer par énumérer ici les principaux termes qui définissent les composants et les fonctionnalités de ClearCase. Ce n'est pas grave si vous ne les comprenez pas tous et que vous ne saisissez pas les dépendances entre eux, vous trouverez dans la section suivante des explications pas à pas qui devraient vous éclairer.

- **VOB** : Espace de stockage physique des sources, tous les projets d'un même VOB partagent le même espace de stockage (la même base de données)
- **Projet** : On définit ici un emplacement pour mettre les sources d'un projet (d'une application). Un projet regroupe les sources qui pourront être accédées depuis une vue.
- **Stream** : Espace de travail, on définit une stream sur un projet (comme une branche Subversion) et c'est sur cette dernière qu'on pourra accéder aux fichiers. On peut créer autant de stream que l'on veut sur un projet.
- **Integration Stream** : Stream 'parent' de chaque Projet dont le but est de permettre l'intégration de toutes les sources. Il n'en existe qu'une par projet.
- **Developpement Stream** : Toute stream enfant est une Developpement Stream. C'est sur une de ces streams que l'on travaille.
- **Child Stream** : On peut définir des streams enfant de toutes streams pour définir une organisation hiérarchique des streams.
- **View** : Littéralement: 'vue' sur une stream. Chaque personne travaillant sur une stream doit se créer une vue sur celle-ci pour accéder aux données.
- **Dynamic View** : Vue dynamique, accès direct aux sources via un lecteur réseau. Toutes les modifications de la stream faites par les autres développeurs sont directement accessibles.
- **Snapshot View** : Vue snapshot, les fichiers sont 'copiés' sur le disque local, le temps d'accès aux fichiers est donc plus rapide. Un checkin/checkout se synchronise toujours avec la stream mais un update est nécessaire pour récupérer les changements des autres développeurs.
- **Baseline** : Une baseline est un ensemble de modifications d'une stream (sorte de snapshot des versions de fichiers à un moment). Toute stream est basée sur une baseline de fondation pour sa création. On définit une baseline recommandée qui représente la version des sources stables et toute stream enfant est créée sur cette baseline (il est aussi conseillé de rebaser les streams enfants avec cette baseline)
- **Activity** : Nom donné à un ensemble de modifications (une sorte de tag), à chaque modification on doit définir ou réutiliser une activité.
- **Checkout** : Action de prendre le control d'un fichier/répertoire pour le modifier. Par défaut, un checkout est 'reserved', un checkout 'reserved' étant unique sur une stream. L'idée étant d'éviter que plusieurs personnes fassent un checkout du même fichier en même temps. On peut aussi faire un checkout 'unreserved', mais dans ce cas là, le checkout 'reserved' à la priorité (entre autre, on ne peut faire un checkin tant qu'il y a un checkout reserved sur ce fichier)
- **Checkin** : Action de rendre le control d'un fichier et d'envoyer ses changements sur la stream pour qu'ils soient accessibles par les autres développeurs de la stream (comme le commit de CVS)
- **Rebase** : Action de mettre à jour une stream avec la baseline par défaut de la stream parent.
- **Deliver** : Action de délivrer ses modifications d'une stream à l'autre. Par défaut on délivre vers la stream parente. On peut soit délivrer une baseline (conseillé), soit un ensemble d'activités.
- **Update** : Pour les vues snapshot, permet de récupérer les modifications faites par les autres développeurs sur la stream.
- **Hijacked** : Etat problématique d'un fichier que l'on obtient parfois en vue snapshot: sorte d'état 'modifié mais pas en checkout'. On peut alors soit faire un checkout puis checkin, soit faire un undo hijacked, mais dans ce cas là on perd ses changements. En fait, cela arrive quand un fichier est modifié sans que le serveur ne le sache. On peut aussi hijacker un fichier sciemment, si on veut pouvoir le modifier sans que le serveur ne le sache, donc sans gêner les autres développeurs. Personnellement, je conseillerais plutôt de faire un checkout unreserved dans ce cas là.

- **Eclipsed** : Etat problématique d'un fichier dû le plus souvent à une suppression ou lorsqu' un lien symbolique est créé dans un répertoire et qu'un fichier du même nom existait déjà.
- **Evil Twin** : Erreur classique quand on essaye de créer un fichier du même nom qu'un fichier précédemment supprimé (ce qui est interdit par ClearCase)

## II-B - Les premiers pas

Après avoir installé et configuré Clearcase sur votre machine (installation classique d'un logiciel pour Windows, configuration via le panneau de configuration des serveurs ClearCase, création d'une variable d'environnement pour la 'Région' de ClearCase qui définit les accès à Clearcase et les VOB vues; tout ceci doit être défini par un administrateur), vous voilà prêt à l'utiliser.

**Etape 1** : créer une vue pour accéder aux fichiers source. Pour cela, plusieurs façons : un wizard 'Join project' existe qui vous permet de joindre un projet et de créer une vue ou alors (et c'est ce que je vous conseille de faire), vous pouvez naviguer dans l'arborescence des projets grâce à l'outil ClearCase Project Explorer et créer une vue depuis un simple clic droit sur une stream. Votre vue est maintenant créée, qu'elle soit dynamique ou snapshot, vous pouvez maintenant réaliser des actions sur les fichiers et répertoires qui la composent.

**Etape 2** : travailler sur la stream. Pour cela, il suffit de parcourir les fichiers et de faire un 'checkout' du fichier voulu, de le modifier puis de faire un 'checkin' si tout est OK ou un 'undo checkout' pour abandonner ses modifications. Vos modifications seront directement visibles par les autres développeurs travaillant sur votre stream si vous travaillez en vue dynamique, ou visibles après un update si vous travaillez en vue snapshot. Si un autre développeur travaillant sur la même stream a déjà fait un checkout d'un fichier, alors vous ne pouvez pas en faire un vous même à moins de faire ce qu'on appelle un 'checkout unreserved'. Un 'checkout unreserved' équivaut à dire 'je fais un checkout mais je n'ai pas la priorité', les changements du checkout normal (checkout reserved) seront prioritaires, il faudra donc attendre que la personne ayant un checkout reserved fasse un checkin avant de pouvoir faire le votre, un merge sera alors nécessaire. A chaque modification d'un fichier, il vous sera demandé d'entrer une activité (un label pour la modification) ou d'en sélectionner une existante. Ne pas oublier, si vous êtes en vue snapshot, de faire un update régulier de votre vue pour récupérer les changements des autres développeurs.

**Etape 3** : délivrer son travail. Ca y est, votre ensemble de modifications est OK, il vous faut alors le partager avec les développeurs travaillant sur les autres streams. Vous allez donc faire un 'deliver' de votre code vers une autre stream (la stream parent, habituellement la stream d'intégration dont c'est le rôle), cela permet d'intégrer ses changements avec ceux des autres. Deux possibilités : créer une baseline et la délivrer, ou délivrer un ensemble d'activités. La première solution, en forçant la création d'une baseline, permet de garder l'ensemble des modifications faites sur votre stream à ce moment là, mais oblige à délivrer tous les changements. Pour la deuxième solution, vous pourrez choisir les activités qui seront délivrées. Mais vous devez savoir que certaines activités sont dépendantes d'autres (modification d'un même fichier dans deux activités, ou création d'une baseline qui rend dépendante toutes les activités ayant participé à cette baseline) ce qui rend cette option assez limitée.

**Etape 4** : récupérer les changements des autres streams. Vous travaillez sur une stream A et des développeurs travaillant sur une stream B viennent de faire un deliver de leurs changements sur la stream d'intégration (ou sur une stream parent que vous partagez). Vous voulez récupérer ces changements. Il faut donc faire une baseline sur la stream d'intégration (ou la stream parent commune) et mettre cette baseline en recommended. Après cela, il suffit de faire un rebase de votre stream pour récupérer les changements (clic droit, rebase sur la stream). Et voilà, si nécessaire, quelques merges seront faits et votre stream sera à jour.

## II-C - Autres fonctionnalités

**Version Tree** : Clic droit -> Version Tree sur un fichier ou un répertoire vous permet de voir l'historique des modifications d'un fichier sous forme d'arbre contenant toutes les modifications sur toutes les streams. On peut alors

comparer des versions, merger des versions, faire des checkout/checkin directement depuis l'arbre. A noter qu'on peut aussi merger des versions de répertoire, ce qui est très pratique pour récupérer des fichiers supprimés par erreur.

**Find Checkouts** : Clic droit -> Find Checkout sur la racine d'une vue ou sur un répertoire. C'est LA commande la plus utile qui permet de lister tous les checkouts faits sur une vue (et ensuite de pouvoir faire un checkin sur ces modifications).

## II-D - Comparaison avec CVS et Subversion

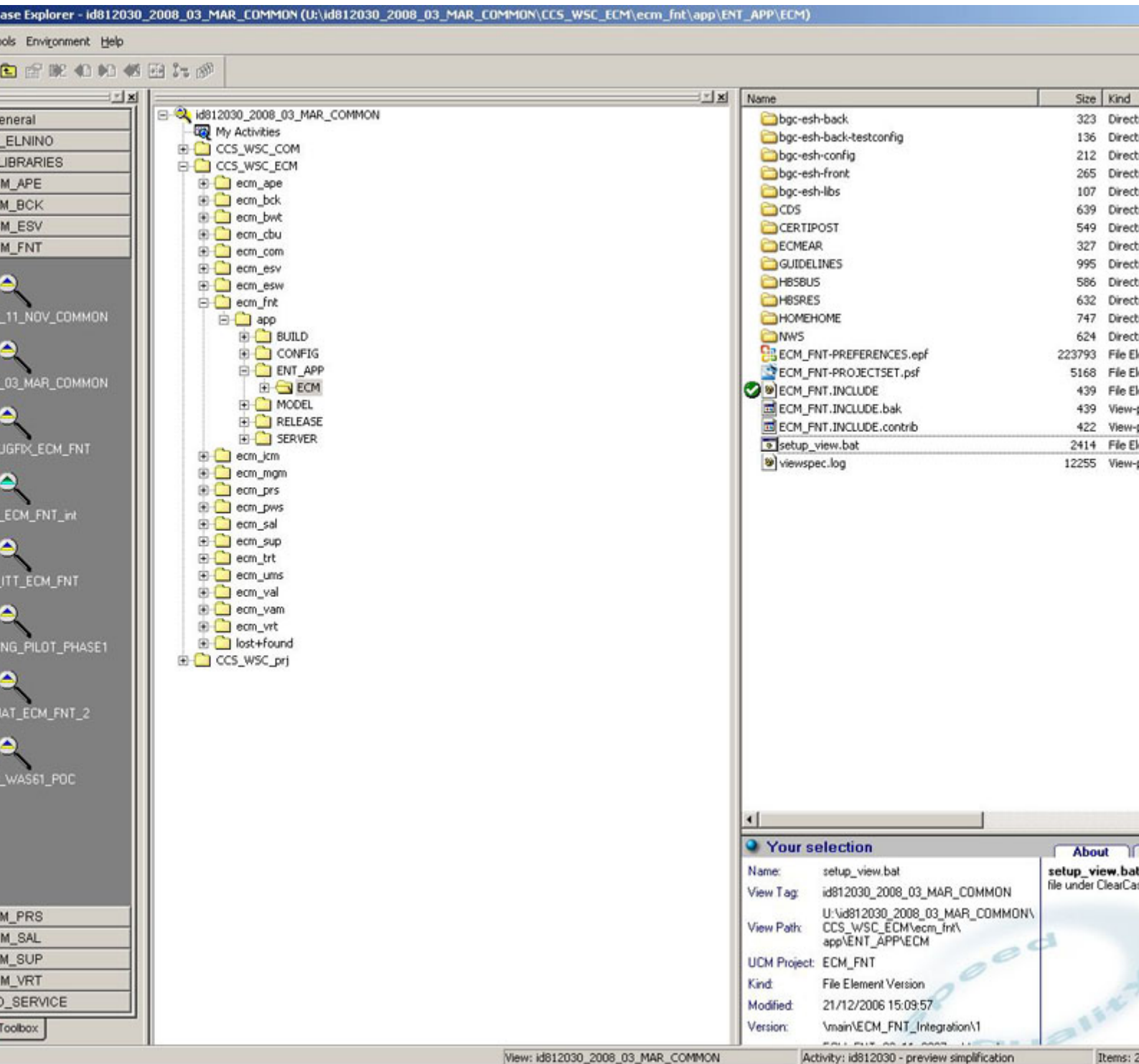
Bien que très éloigné de CVS et Subversion, on peut essayer de donner quelques points de repère:

- **Checkout** : notion existante dans CVS et Subversion mais rarement utilisée qui consiste à réserver un fichier pour le modifier (les autres ne pouvant plus alors le modifier), dans CVS on appelle cela un fichier **Watched**
- **Stream** : peut être remplacé par l'utilisation des branches
- **Baseline** : peut être remplacé par l'utilisation des tags
- **Activité** : notion inexistante dans CVS et Subversion (bien que le commentaire puisse remplacer une partie de l'activité, celle de déclarer le pourquoi des modifications)

### III - Les principaux outils

Clearcase est livré avec de nombreux outils. Deux sont nécessaires à son utilisation (les autres étant plutôt accès administration)

#### III-A - ClearCase Explorer



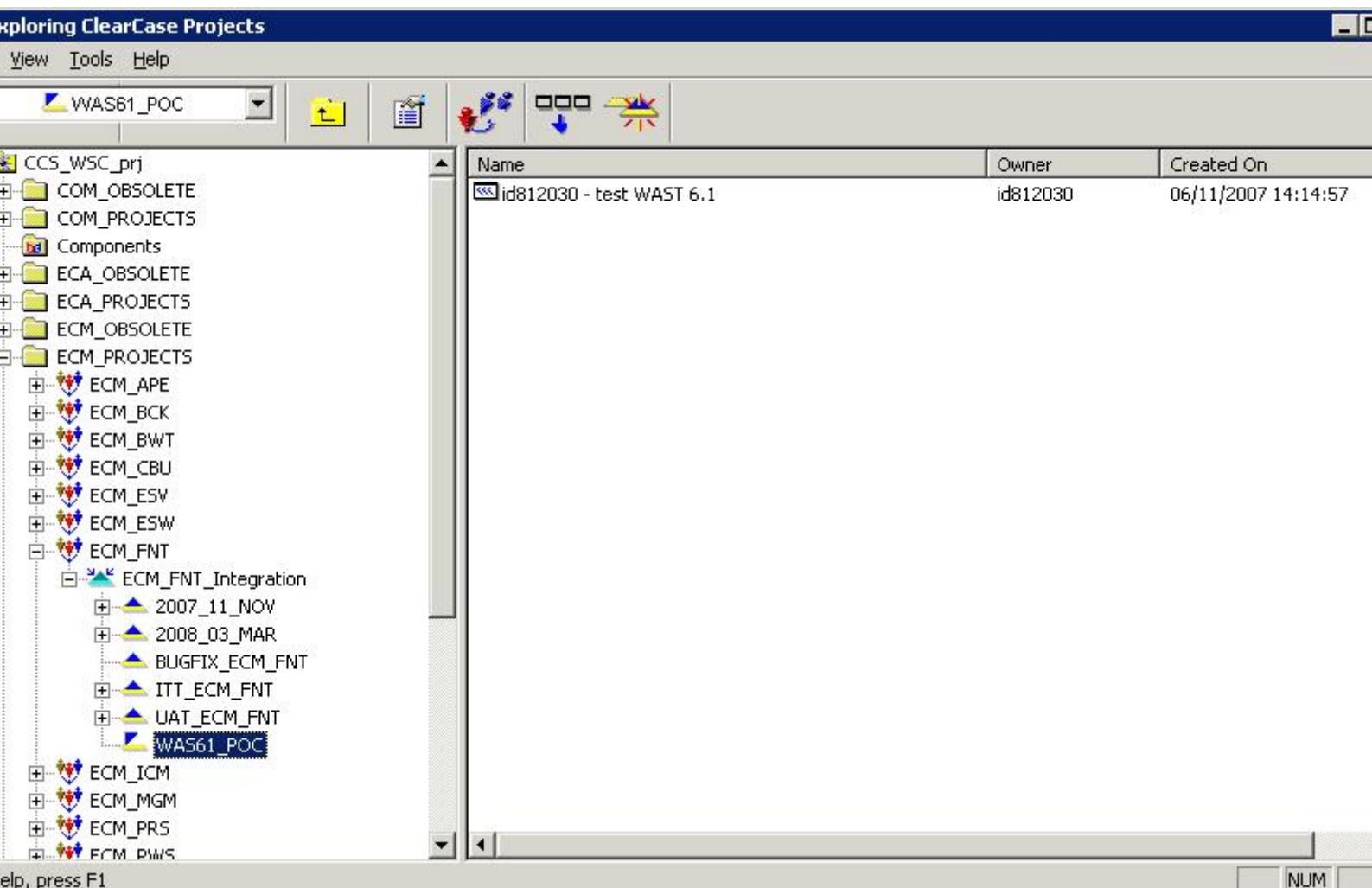
### ClearCase Explorer

Cet outil sera celui que vous utiliserez le plus si ClearCase est votre gestionnaire de code source. Il permet de réunir toutes les vues que l'on a sur tous les VOBs et Projets. Ces vues sont classées sur la gauche dans des onglets qui représentent les projets. Par le menu 'View' -> 'Refresh view shortcuts', ClearCase Explorer va directement aller vous chercher toutes les vues existantes définies sur votre ordinateur et créer un shortcut à gauche.

En cliquant sur un shortcut à gauche, on accède à l'arborescence de la vue. On peut alors parcourir cette arborescence et, sur chaque fichier exécuter les actions ClearCase voulues sur un fichier (Checkout, Checkin, Undo Checkout,...). ClearCase Explorer vous donne une vue 'directe' sur vos fichiers, leur statut est toujours correct contrairement aux plugins des IDE qui peuvent se désynchroniser et ne montrent pas toujours le bon statut d'un fichier.

Un menu déroulant par clic droit sur une vue permet les actions suivantes: Find Checkout, Rebase, Update, Deliver et 'Properties of view'. Ce dernier donne accès à un bouton 'Synchronize with Stream' qui peut être utile pour resynchroniser une vue après un rebase de la stream. En effet, toute action sur une stream, se fait via une vue, les autres personnes travaillant sur cette stream mais utilisant une vue différente, doivent alors resynchroniser leur stream pour être sûrs de voir ces changements.

### III-B - ClearCase Project Explorer



*ClearCase Project explorer*

Cet outil permet de naviguer entre les VOBs, les projets et les streams. Il est très pratique car donne accès directement à toutes les informations sur une stream et permet de lancer directement les opérations possibles sur une stream (rebase, création d'une baseline, deliver). On a aussi, sur une stream, un aperçu des activités faites par tous les utilisateurs.

Via Clearcase Project Explorer, on navigue au sein des VOB/projets/stream et on peut facilement, d'un clic droit, créer une vue sur une de ces streams. On a aussi accès aux propriétés des streams et on peut effectuer les actions suivantes : voir la dernière baseline recommandée, la changer, créer une baseline, locker une stream (plus personne à part vous ne pourra faire de modification), la délocker, la rendre obsolète (elle n'est alors plus visible), ...

### III-C - Cleartool

ClearCase fournit un outil en ligne de commande permettant de réaliser toutes les actions possibles dans ClearCase. Son utilisation est plus à destination des administrateurs et fait partie d'une utilisation avancée de Clearcase, je ne vais donc pas en parler ici. Cleartool peut être appelé depuis un shell DOS via la commande **cleartool.exe** ou est lui-même une sorte de shell qui est par défaut appelé via Clearcase Explorer lors d'un clic droit -> Command Prompt sur un répertoire.

## IV - Intégration

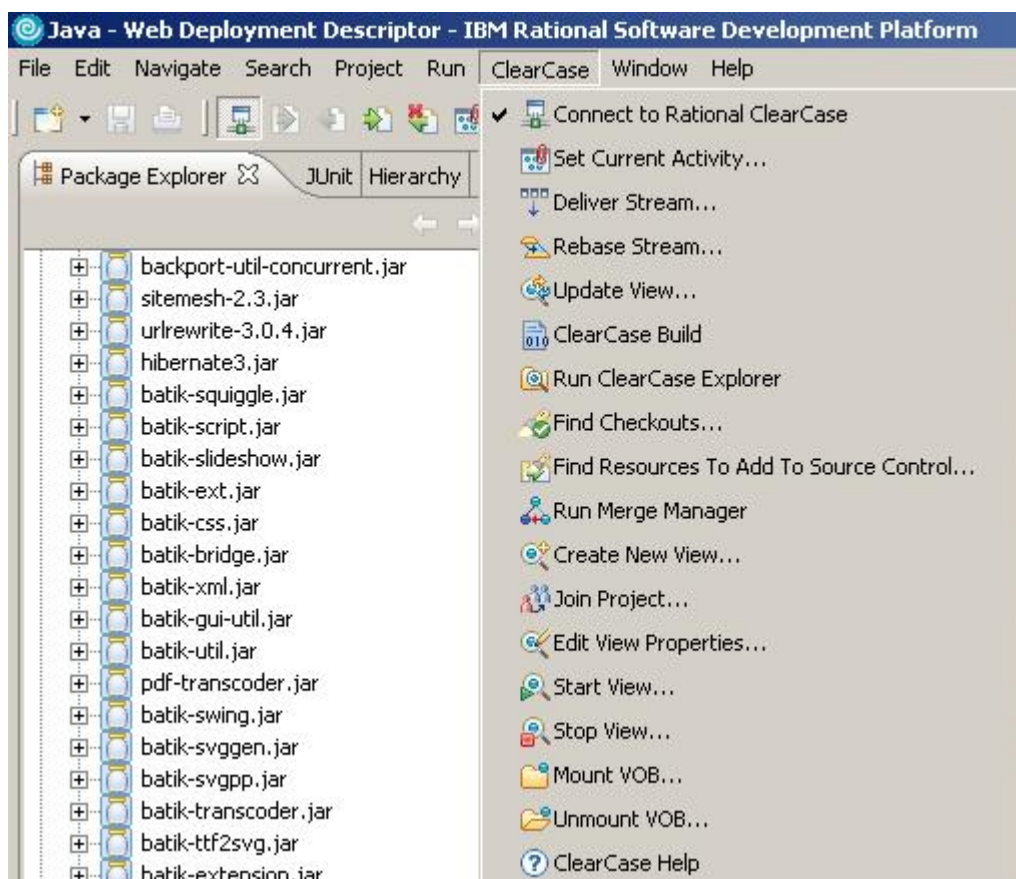
### IV-A - Dans eclipse

Un plugin ClearCase est disponible en téléchargement sur le site d'IBM. Après installation et activation (procédure standard décrite sur le site), on obtient la barre des tâches suivante:



*ClearCase - la barre des tâches du plugin Eclipse*

Un menu déroulant permet d'avoir accès aux mêmes options:



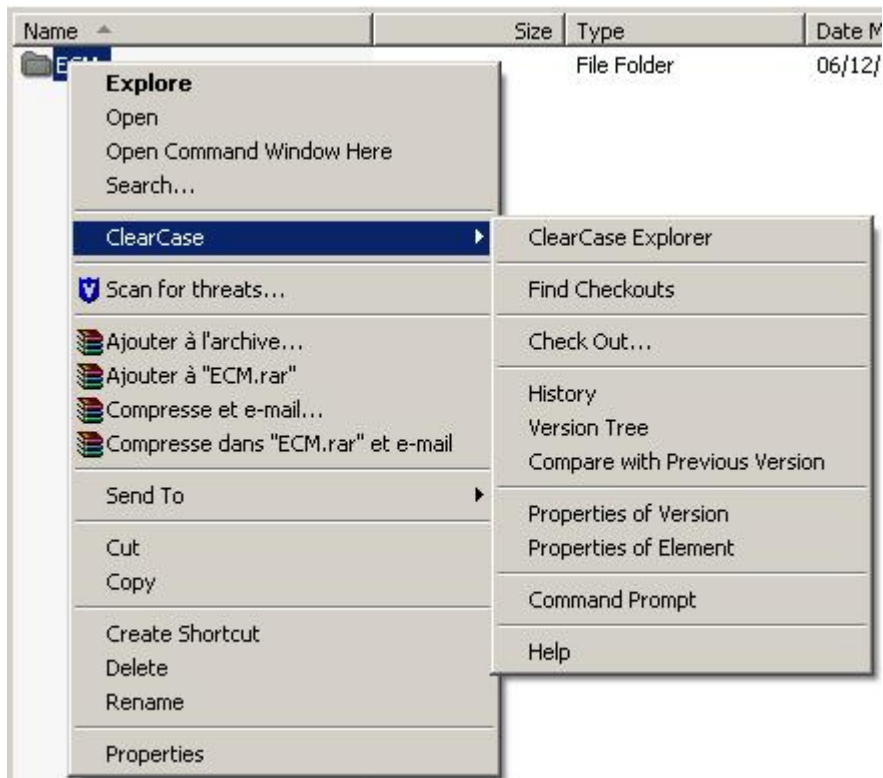
*ClearCase - le menu du plugin Eclipse*

### IV-B - Autres IDE

Un plugin ClearCase semblable à celui d'Eclipse existe pour la plupart des IDE Java connus (j'ai testé ceux de Netbean et IntelliJIDEA). Les IDE d'IBM (WASD, RAD, Websphere AST) sont bien sûr livrés avec le plugin ClearCase pré-installé, qui est le même que celui d'Eclipse, ces IDE étant basés sur Eclipse.

## IV-C - Dans l'explorateur Windows

Par clic droit sur un fichier, un menu déroulant apparaît donnant accès aux principales options/commandes de ClearCase sur un fichier ou un répertoire.



*Intégration de ClearCase dans l'explorateur windows*

## IV-D - Dans ANT

ANT fournit une suite de tâches permettant de réaliser les principales actions clearcase.

La documentation peut être trouvée ici:  <http://ant.apache.org/manual/OptionalTasks/clearcase.html>.

En voici la liste des plus importantes:

- **CCCheckin** : effectue un checkin d'un élément (fichier ou répertoire)
- **CCCheckout** : effectue un checkout d'un élément (fichier ou répertoire)
- **CCUnCheckout** : effectue un undo checkout d'un élément (fichier ou répertoire)
- **CCUpdate** : effectue un update d'une vue
- **CCLock** : permet de faire un lock sur une stream ce qui empêche tout autre personne de la modifier
- **CCUnlock** : délock une stream

Une autre manière de faire est d'appeler l'outil cleartool via la commande exec de ANT. Je vous en fournis ici un exemple, voir la documentation de l'outil cleartool pour plus de détails

```
<exec dir="" executable="cleartool.exe">  
  <arg line="ci -comment 'Mon commentaire' U:/MaStream/monfichier.txt" />  
</exec>
```

## IV-E - Dans MAVEN

Ne l'ayant pas utilisé avec MAVEN, je vais me contenter de vous donner le lien vers la page de documentation du plugin MAVEN de clearcase:



<http://maven.apache.org/scm/clearcase.html>

## IV-F - Travail depuis Eclipse avec ClearCase

Le refactoring depuis Eclipse avec ClearCase est long et plus compliqué qu'avec des outils tels que CVS ou Subversion. Les principaux problèmes sont dus au fait que pour modifier un fichier, contrairement à CVS ou Subversion, l'IDE doit faire un checkout du fichier avant. Apparemment, cela pose parfois un problème à Eclipse (dû peut-être à une mauvaise synchronisation avec le plugin) qui n'arrive pas à faire tous les checkouts. Un refactoring entraînant des suppressions/déplacements de fichier est encore pire.

Dans tout les cas, pour un refactoring, je vous conseille de faire un 'preview' avant, cela aide le plugin clearcase à se synchroniser, il fait alors les checkouts lors de la preview et modifie les fichiers ensuite.

Une dernière limitation au niveau du travail dans Eclipse, est l'impossibilité de faire une copie d'un fichier sur lui-même (pour mettre une nouvelle version par exemple), le plugin eclipse supprimant le fichier précédent et le remplaçant par le nouveau au lieu de créer une nouvelle version. Pour copier une nouvelle version d'un fichier sur lui-même, je conseille donc d'aller dans ClearCase Explorer, de faire un checkout du fichier, de copier la nouvelle version puis de faire un checkin.

En cas de problème sous Eclipse, ne pas oublier que seul ClearCase Explorer est toujours synchronisé avec ClearCase, il se peut que de temps en temps Eclipse se désynchronise, dans ce cas 'refresh project', puis 'ClearCase -> Synchronize Status'. Si le problème persiste, allez voir dans Clearcase Explorer.

## V - Aller plus loin : les références entre projets

Je vais ici aborder une partie fort complexe mais il faut l'avouer très puissante de ClearCase. La possibilité de faire des références entre projets ClearCase. Prenons par exemple, une application web qui veut utiliser un framework commun à de nombreuses autres applications, ce framework étant développé sur un projet différent (sur le même VOB ou un VOB différent). Vous voulez alors, facilement, avoir accès depuis votre projet aux classes de ce framework pour automatiser la gestion des versions de ce framework.

**Etape 1** : Créer un fichier INCLUDE qui définit quelles baselines lier à votre stream. On définit ici l'accès à une baseline de notre framework en précisant que le checkout n'est pas autorisé depuis cette stream. On peut mettre autant d'éléments que nécessaire pour lier vers plusieurs projets.

### STEAM\_SPEC.INCLUDE

```
element * BASELINE_FRAMEWORK_0.01 -nocheckout
```




**Etape 2** : Mettre à jour les spécifications de votre stream avec le fichier include précédemment créé. Pour ceci, on va utiliser l'outil cleartool

```
cleartool setcs -tag U:/ViewTag U:/ViewTag/STEAM_SPEC.INCLUDE
```

**Etape 3** : Si vous utilisez une vue de type snapshot, il faut faire un update de votre vue.

Vous avez maintenant, depuis votre vue, accès aux données de la stream de votre framework (sans pouvoir les modifier), ils seront tout simplement accessibles depuis la racine de votre vue (qui comprendra donc le répertoire racine de votre projet ainsi que celui du projet du framework). Les données du framework seront celles de la baseline que vous avez spécifiée.

## VI - Liens

- Informations produit sur le site d'IBM :  <http://www-306.ibm.com/software/awdtools/clearcase/>.
- Les tâches ClearCase de ANT :  <http://ant.apache.org/manual/OptionalTasks/clearcase.html>.
- Le plugin ClearCase de MAVEN :  <http://maven.apache.org/scm/clearcase.html>

## VII - Remerciements

Je remercie tous les gens qui m'ont aidés dans la rédaction de cet article et spécialement :

- denisC : pour sa relecture technique et ses nombreux conseils
- vbrabant : pour sa relecture technique
- Pi2 : pour sa relecture orthographique

