Understanding ClearCase Client, Administrative VOB Interactions and Metadata in a UCM Environment

William Frontiero (wfronti@us.ibm.com)

November 5, 2008

INTRODUCTION 3

OVERVIEW OF ADMINISTRATIVE VOB HIERARCHY IN UCM 4

BUSINESS CASE FOR UNDERSTANDING THE ADMINISTRATIVE VOB HIERARCHY IN UCM ENVIRONMENTS FOR OPTIMAL DEVELOPMENT: 4
THINGS TO CONSIDER FROM THE START 4
HOW UCM ENABLES MULTIPLE COMPONENT VOBS TO SHARE METADATA TO ACHIEVE A COMMON GOAL 5
CORE ADMINISTRATIVE VOB METADATA UNDERSTANDING 5
WHAT METADATA IS SPECIFIC TO COMPONENT VOBS / WHAT METADATA EXISTS SOLELY IN THE PROJECT VOB 5

SUMMARY / CONCLUSION 17

REFERENCES 18

Introduction

IBM® Rational® ClearCase® UCM (Unified Change Management) makes extensive use of client and administrative VOB relationships. This white paper provides details about those relationships and the underlying actions that result from various operations performed in a UCM environment. It does not contain all possible interactions that may take place on objects in a UCM environment.
This document is designed to be read by ClearCase UCM Administrators who are responsible for their organizations UCM configuration.
Before proceeding you should have an understanding of the general UCM concepts covered in the Understanding UCM section of *IBM Rational ClearCase Managing Software Projects*.

Overview of Administrative VOB Hierarchy in UCM

UCM Development implies parallel development amongst multiple component VOBs. These component VOBs are administered by a common administrative VOB (project VOB). Each component VOB can be developed independently, while adhering to metadata common to all. Understanding the Administrative VOB structures, along with inter-relationships between client and Administrative VOBs, is key to the successful design and administration of an optimal UCM environment.

Business Case for Understanding the Administrative VOB Hierarchy in UCM Environments for Optimal Development:

UCM environments are based on the core understanding and functionality of the administrative VOB Structure. Administration and design of a UCM environment must take into consideration the underlying administrative VOB structure. Things to consider include; backup and restore, MultiSite replication, total size of all associated VOBs, ownership of metadata and hyperlinks shared by the VOBs.

Without a solid understanding of administrative VOB structures and how UCM relies on them, making design decisions, troubleshooting issues and general development can become overwhelming. For instance UCM development will not operate as desired if metadata is unavailable from required locations. UCM operations such as make component, make baseline, make stream, checkout/in and deliver require the use of shared metadata. Understanding where UCM specific metadata resides is required when troubleshooting, configuring and operating a UCM environment

Things to Consider from the Start

When Creating a UCM environment, you should consider building for scalability and growth. The creation of a requirements document will help in achieving these desired goals. Before implementing the environment, you should consider the following while also considering future project requirements:

1.) Total number of component VOBs.
2.) Will the component VOBs and project VOB exist on the same VOB server?
3.) How large will the component VOBs become over time.
4.) What dependencies will exist between the component VOBs.
5.) Will the component VOBs need to be shared with new project VOBs in the future
6.) Will there be a need for multiple project VOBs upon implementation.

How UCM enables Multiple Component VOBs to Share Metadata to achieve a common goal

Linking multiple components to a common administrative VOB (PVOB) provides the structure necessary for sharing component data and metadata amongst various inter-related development projects. The sections below will provide further details about the infrastructure UCM supplies to make this all possible.

Core Administrative VOB Metadata Understanding

UCM baselines (identified by the use of UCM metadata) are vital to the success of each UCM component's development. If there is no means of testing the baselines integrity, then a successful build becomes the only means of validation.

Many operations performed in a UCM environment trigger a chain of events that include interactions between a client VOB containing one or more UCM components and it's associated administrative project VOB (PVOB). There is key metadata which is generated during those interactions.

The information below outlines the key metadata generated during standard UCM operations. The examples assume a project VOB "bfPvob" and a component VOB "bfCvob". We will outline and discuss what, when and where metadata is generated during typical UCM development.

What Metadata is specific to Component VOBs / What Metadata exists solely in the Project VOB

The following sections identify the metadata generated in the component VOB and project VOB when performing traditional UCM operations.

1.) Component Creation: Overview of metadata generated in the VOBs during component creation:

Project VOB "bfPvob"

- Component object
- ComponentRootDir hyperlink (pointing from the component toward the root directory Element)
- Component INITIAL_BASELINE label type

Component VOB "bfCvob"

- Root Director Element (Vob root which is marked by a CRDE "Component root directory Element" Number)
- AdminVOB hlink (pointing from the component VOB toward the project VOB)

2.) Stream Creation: Overview of metadata generated in the VOBs during stream creation:

Project VOB “bfPvob”
- Global branch type associated with the UCM stream UCM Stream object
- IndependentGuard hyperlink associating the stream with the global branch type
- UseBaseline hyperlink linking the stream to the baseline being chosen

Component VOB “bfCvob”
- No NEW metadata generated on stream creation

3.) Checkout/Checkin: Overview of metadata generated in the VOBs during Checkout / Checkin:

Project VOB “bfPvob”
- UCM Activity object
- Changeset hyperlink pointing from the activity to the version

Component VOB “bfCvob”
- Local copy of the global branch type
- GlobalDefinition hyperlink pointing from the branch local copy to the Global Copy
- New Directory version

4.) Make Baseline: Overview of metadata generated in the VOBs during the make baseline Operation:

Project VOB “bfPvob”
- UCM baseline object
- BaselineLbtype hyperlink (pointing from the baseline to the lbtype in the component VOB)

Component VOB “bfCvob”
- Ordinary label type associated with the baseline object
- IncrementalLbtype hyperlink pointing to the previous baseline associated label type

The following operations are discussed in the sections below:

- Make component Operation (cleartool mkcomp) figure 1.0
- Make Stream Operation (cleartool mkstream) figures 1.1 and 1.2

- Checkout / Checkin versions on the UCM stream (cleartool co / cleartool ci)
- Make baseline Operation (cleartool mkbl)

1.) Operation: Make component (cleartool mkcomp –root .\bfCvob component:bfCvob@\bfPvob)

The creation of a UCM component will require two major hyperlinks to be created; a ComponentRootDir hyperlink and an AdminVOB hyperlink. The ComponentRootDir hyperlink will be created in the project VOB. The ComponentRootDir hyperlink is unidirectional. It will point from the component object in the PVOB, toward the root directory object (either a VOB root, or subdirectory of the root). The AdminVOB hyperlink will be generated in the respective component VOB. The AdminVOB hyperlink is also unidirectional. It will point from the component VOB toward the project VOB. Any failure to create these two hyperlinks can result in component creation failure.

Examples:

- cleartool describe –l ComponentRootDir@111@\bfPvob

  Here is what a long describe of the ComponentRootDir hyperlink looks like:

  ComponentRootDir@111@\bfPvob component:bfCvob@\bfPvob -> M:\betaBase\bfCvob\.@@

- cleartool describe –l –ahlink –all component:bfCvob@\bfPvob

  Here is what the hyperlink looks like when running a long describe of the component object in a view context:

  ComponentRootDir@111@\bfPvob -> M:\betaBase\bfCvob\.@@

- cleartool describe –l –ahlink –all .@@

  Here is what the hyperlink looks like when running a long describe of the root directory object in a view context:

  ComponentRootDir@111@\bfPvob <- component:bfCvob@\bfPvob

- cleartool describe –l AdminVOB@46@\bfCvob

  Here is what a long describe of the AdminVOB hyperlink looks like:

  AdminVOB@46@\bfCvob vob:\bfCvob -> vob:\bfPvob

- cleartool describe –l –ahlink –all vob:\bfCvob

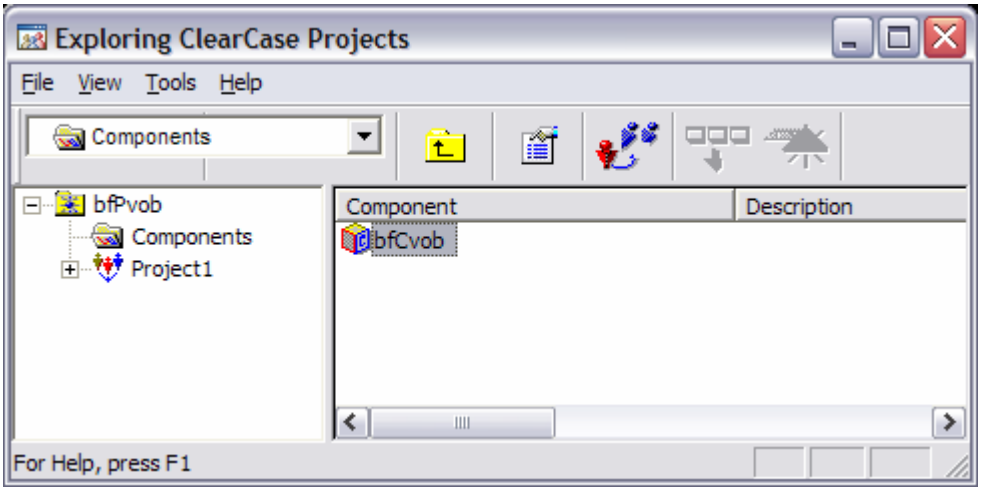
  Here is what a long describe of the component VOB looks like:

  AdminVOB@46@\bfCvob -> vob:\bfPvob

- cleartool describe –l –ahlink –all vob:\bfPvob

  Here is what a long describe of the project VOB looks like:
  AdminVOB@46@\bfCvob <- vob:\bfCvob

  Figure 1.0 (This screen shot shows the component “bfCvob” created and associated with project VOB “bfPvob”)



2.) Operation: Make Stream (cleartool mkstream -integration -in project:Project1@\bfPvob -nc -baseline bfCvob_INITIAL@\bfPvob Project1_Integration_Stream@\bfPvob)

Each stream is linked to a global branch type in the project VOB. This ensures that any component VOB performing a checkout will receive a local copy of the shared global branch type. This is the core of how UCM is able to track versions from multiple component VOBs using a single stream. A stream becomes a central point of administration for any component added to its configuration. Any version generated from the streams associated global branch type can be tracked and managed centrally in UCM. Although UCM baselines are specific to components, they are also created in a stream context. This means a stream enables users to run a single operation to generate new baselines across multiple components. This also means the

stream can enforce policies that will be applied to all components configured in the stream. The shared branching structure enabled by the administrative VOB structure becomes the Core of UCM development.

The creation of a UCM stream will generate a corresponding global branch type in the project VOB. The stream will be linked to the branch type by an IndependentGuard hyperlink. Independent Guard hyperlinks are created in the project VOB and are unidirectional. They point from the stream object toward the corresponding branch type. When creating a stream, you can choose the UCM baselines you would like the stream Configured with. A UseBaseline hyperlink will be created in the project VOB associating the stream with the baseline chosen. The UseBaseline hyperlink is also a unidirectional hyperlink. It will point from the stream object toward the baseline object being selected. There will be a UseBaseline hyperlink for each foundation baseline in the streams configuration.

Examples:

- Here is the name of the stream:
  Stream:Project1_Integration_Stream@\bfPvob

- Here is the name of the global branch type:
  brtype:Project1_Integration_Stream@\bfPvob

- Here is the name of the IndependentGuard hyperlink generated:
  IndependentGuard@146@\bfPvob

  IndependentGuard hyperlinks point from the stream object, to the global branch type

- cleartool describe –l hlink:IndependentGuard@146@\bfPvob

  Here is what a long describe of the hyperlink looks like:
  IndependentGuard@146@\bfPvob
  stream:demo_Proj_Integration@\bfPvob ->
  brtype:demo_Proj_Integration@\bfPvob

- cleartool describe –l –ahlink –all stream:
  Project1_Integration_Stream@\bfPvob

  Here is what the hyperlink looks like from a long describe of the stream

IndependentGuard@146@\bfPvob ->
brtype:Project1_Integration_Stream@\bfPvob

- cleartool describe –l –ahlink –all brtype:
  Project1_Integration_Stream@\bfPvob

  Here is what the hyperlink looks like from a long describe of the Associated global branch type:

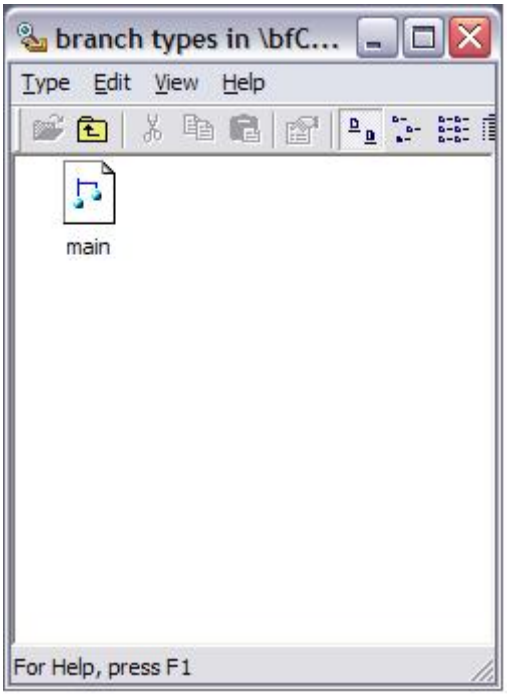
  IndependentGuard@146@\bfPvob <-
  stream:Project1_Integration_Stream@\bfPvob

- cleartool describe –l hlink:UseBaseline@154@\bfPvob

  Here is an example of the UseBaseline hyperlink that is created from the stream to the UCM baseline:

  UseBaseline@154@\bfPvob
  stream:Project1_Integration_Stream@\bfPvob ->
  baseline:bfCvob_INITIAL@\bfPvob

  NOTE: At this point, component VOBs do NOT contain a copy of the global branch type.



NOTE: At this point the global branch types exist in the project VOB:

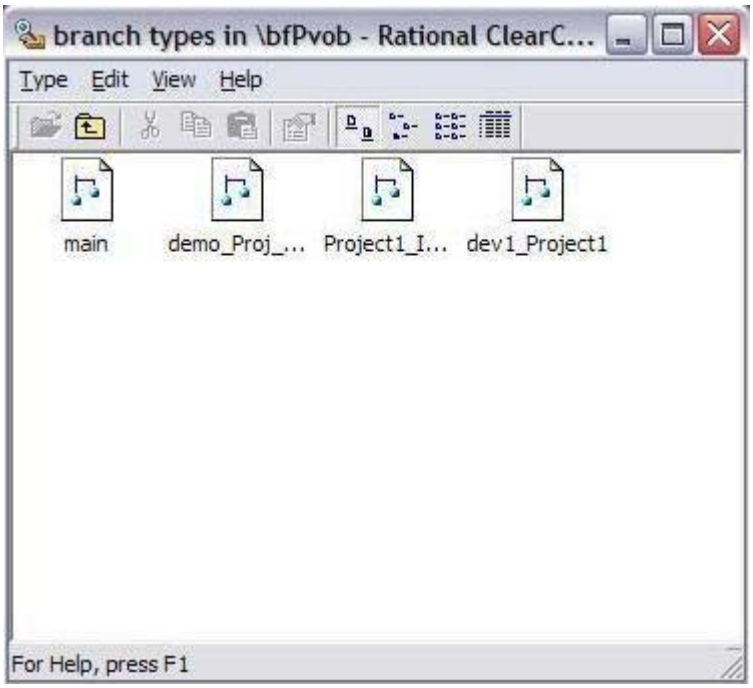


Figure 1.1 (Screen Shot of the stream created in the bfPvob)
In the figure below you can see the stream was created under a new project in the project VOB "vob:\bfPvob"

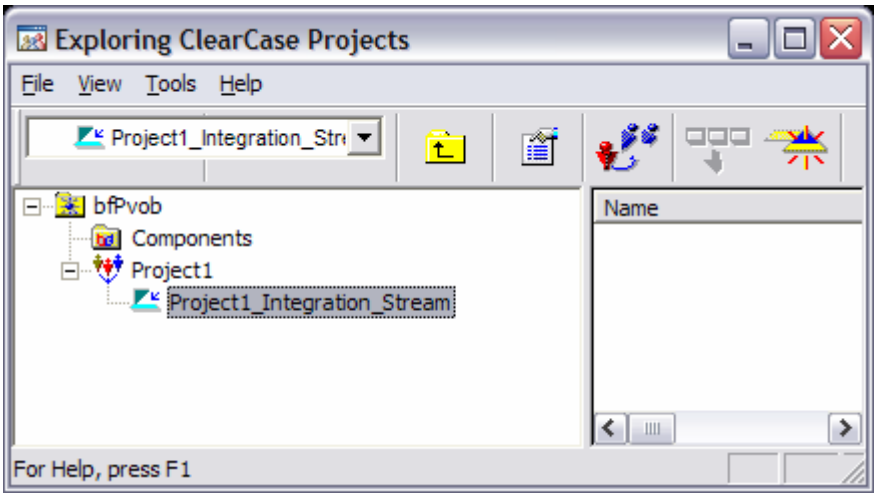
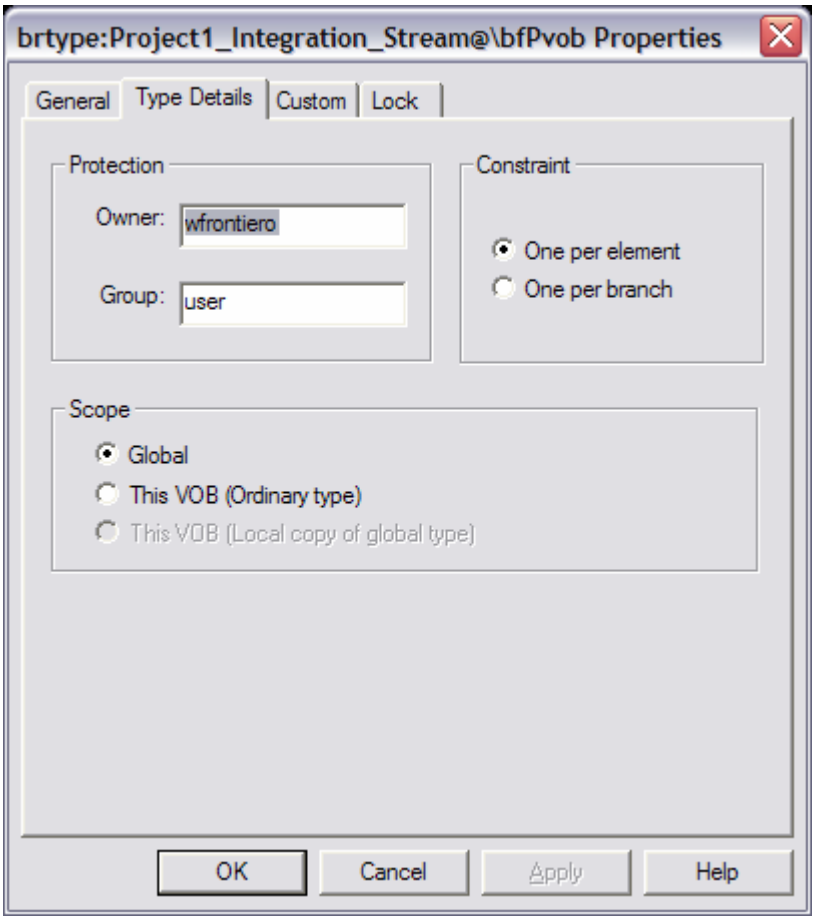


Figure 1.2 (Screen Shot of the project VOBs branch type properties in the type explorer)
In the screen shot below you can see the branch type is a "Global" branch type. This means any component VOBs associated with the "bfPvob" can generate local copies of this global type



3.) Operation: Checkout / Checkin versions on the UCM stream (cleartool co / cleartool ci)

Checking out an element on a UCM stream will require the usage of a UCM Activity. Once you have created and set to a desired activity you can then perform a checkout. When checking out an element on a UCM stream, the component VOB containing the element will generate a local copy of the streams corresponding global branch type. In the information above, we demonstrate that the stream is linked to the global branch type in the project VOB. This ensures that any component VOB performing a checkout will receive a local copy of the shared global branch type. This is how UCM is able to track versions from multiple component VOBs using a single stream. Any version generated from a local instance of the global branch type can be tracked by the guarding stream. Any checkin on a UCM stream will require that an activity is also associated at the time of checkin. The activity is

usually the same activity used for checkout. During the checkin, a changeset hyperlink is created from the activity to the version in the component VOB. (NOTE: When you checkout on a UCM stream, a change hyperlink is associated with the checkout reference until the change is committed)

Before checking out you will need to set to a UCM Activity. You can use cleartool setact <Existing_Act> or you can make a new activity and set to it in one step.

In the example below we are using mkactivity: cleartool mkact bfDevActivity

Here is the output of running mkact from a UCM View:

Created activity "bfDevActivity".
Set activity "bfDevActivity" in view "Project1_int".

Here is an example of a directory element checkout on a UCM stream: (cleartool checkout –nc .)

Note at this point that the global branch type is being used to generate a local copy in the component VOB.

Automatically created branch type "Project1_Integration_Stream" from global definition in VOB "\bfPvob".
Created branch "Project1_Integration_Stream" from "." version "\main\0".
Checked out "." from version "\main\Project1_Integration_Stream\0".
  Attached activity:
    activity:bfDevActivity@\bfPvob "bfDevActivity"

Notice that the activity we created is being attached to the checked out version of the directory element on the UCM stream.

At this point, let's take a look at the hyperlink created that associates the local copy of the branch type in the component VOB to the global type in the project VOB. The hyperlink is of type GlobalDefinition. The hyperlink is created in the component VOB. It is a unidirectional hyperlink. It points from the local copy of the branch type in the component VOB toward the global branch type in the project VOB.

- Here is an example of a long describe of the global definition hyperlink created:
  cleartool describe –l GlobalDefinition@48@\bfCvob
  GlobalDefinition@48@\bfCvob

brtype:Project1_Integration_Stream@\bfCvob ->
brtype:Project1_Integration_Stream@\bfPvob

Notice that it points from the brtype in the bfCvob toward the brtype in the bfPvob.

At this point the directory checkout is going to be checked in. The example below demonstrates the directory version being associated with the UCM activity.

- Here is a long describe of the activity before the checkin:
  Cleartool describe –l –ahlink –all activity:bfDevActivity@\bfPvob
  Change@164@\bfPvob ->
  M:\Project1_int\bfCvob\.@@\main\Project1_Integration_Stream\CHECKEDOUT.51

- Here is a long describe of the activity after the directory checkin.:
  cleartool describe –l –ahlink –all activity: :bfDevActivity@\bfPvob
  Change@164@\bfPvob ->
  M:\Project1_int\bfCvob\.@@\main\Project1_Integration_Stream\1

- Here is a long describe of the change hyperlink associating the version with the activity:
  cleartool describe –l hlink:Change@164@\bfPvob
  Change@164@\bfPvob
  activity:bfDevActivity@\bfPvob ->
  M:\Project1_int\bfCvob\.@@\main\Project1_Integration_Stream\1

  Notice that the change hyperlink points from the UCM activity to the version created in the component VOB. It is also helpful to notice that the change hyperlink exists in the project VOB.

4.) Operation: Make Baseline (cleartool mkbl BF2_Demo)

Making a UCM baseline will require a component object and a stream context. The baseline creation procedure will determine which component it will be created for, and which stream it will be associated with. A baseline creation procedure will generate a baseline object in the UCM project VOB. It will also create a label type in the component VOB. A BaselineLbtype hyperlink will be generated in the project VOB linking the baseline to the label type. The BaselineLbtype hyperlink is unidirectional pointing from the baseline object toward the label type. When creating a UCM baseline you can choose to create a full baseline or an incremental baseline. Deciding to create a full baseline will not alter the actual baseline object. It will generate a label type applied recursively across all LATEST versions being selected by the streams config spec. When choosing to make an incremental baseline, it will only apply the label to the versions changed or created since the last fully labeled baseline created. The make baseline operation is able to determine the last

fully labeled baseline through the use of IncrementalLbtype hyperlinks. These hyperlinks connect incremental label types back to the last fully labeled lbtype. A fully labeled baseline is considered a backstop for successive baselines created until another full.

Examples:

- Here is an example of the make baseline operation: (cleartool mkbl BF2_Demo)
  Created baseline "BF2_Demo" in component "bfCvob".
  Begin incrementally labeling baseline "BF2_Demo".
  Done incrementally labeling baseline "BF2_Demo".
  Notice that the baseline is being incrementally labeled. This means the baseline is an incremental baseline. The make baseline command will create an incremental baseline unless Specified with the "-full" flag. The GUI has a drop down option to choose.

- Here is a long describe of the baseline to show any hyperlinks associated:
  cleartool describe –l –ahlink –all baseline:BF2_Demo@\bfPvob
  BaselineLbtype@172@\bfPvob -> lbtype:BF2_Demo@\bfCvob

  Notice that the baseline object does not contain any information about the labeling structure. The label is responsible for keeping track of the label status.

- Here is a long describe of the label type associated with the baseline.:
  cleartool describe –l –ahlink –all lbtype:BL1_Demo@\bfCvob

  The first hyperlink we see is of type IncrementalLbtype. This link will keep track of previous labels. These types of hyperlinks work together to trace back to the last fully generated label type.
  IncrementalLbtype@55@\bfCvob -> lbtype:BL1_Demo@\bfCvob

  The second hyperlink we see is of type BaselineLbtype. This link will associate the baseline object with the label type.
  BaselineLbtype@172@\bfPvob <- baseline:BF2_Demo@\bfPvob (This hlink associates the label type with the baseline)

- Here is the long describe of the IncrementalLbtype hyperlink:
  cleartool describe –l hlink:IncrementalLbtype@55@\bfCvob
  IncrementalLbtype@55@\bfCvob lbtype:BF2_Demo@\bfCvob ->
  lbtype:BL1_Demo@\bfCvob

- Here is the long describe of the BaselineLbtype hlink:
  cleartool describe –l hlink:BaselineLbtype@172@\bfPvob
  BaselineLbtype@172@\bfPvob baseline:BF2_Demo@\bfPvob ->
  lbtype:BF2_Demo@\bfCvob

Summary / Conclusion

As discussed above, a core understanding of the administrative VOB structure in a UCM environment is required for optimal development. If the administrative VOB structure is damaged or administered improperly, UCM functions will suffer greatly. Understanding how to protect and manage the VOB structure will enable administrators and developers to better perform day to day operations. This means less downtime and higher productivity leading to overall project success.

References

The following were used in references or as other sources of information:

- http://www.ibm.com
- Managing Software Project: http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/index.jsp?topic=/com.ibm.rational.clearcase.cc_proj.doc/cc_proj_manage.htm
- Developing Software http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/index.jsp?topic=/com.ibm.rational.clearcase.dev.doc/topics/cc_dev/c_basic_cc_concepts.htm
- Command Reference http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/index.jsp?topic=/com.ibm.rational.clearcase.cc_ref.doc/topics/refintro.htm