ClearCase Change Management Integration FAQ

Question

How can I configure ClearCase to integrate with ClearQuest, Rational Team Concert, or Atlassian JIRA?

Answer

I. Table of Contents

II. Overview

ClearCase supports integrations with Change Management systems such as Rational ClearQuest (CQ), Rational Team Concert (RTC), and Atlassian JIRA through the Change Management Integration (CMI). The integrations require some initial configuration; after which you can work with UCM activities or base ClearCase versions that are associated with the tasks.

The Change Management Integration allows for flexible configurations and a lightweight connection to one or more Change Management systems (CQ, RTC, or JIRA). It is highly customizable to fit numerous configuration requirements while not requiring extensive knowledge of the integration for end users.

III. CMI Configuration

Note: For the CMI integration to work properly with ClearQuest:

- ClearCase must be installed on the ClearQuest Web Server.
- The ClearCase installation on the client must point to the same ClearCase Registry as the ClearCase installation on the ClearQuest Web Server so that the VOBs can be found.

1. ClearCase and Change Management Prerequisites

There are a few prerequisites before CMI can be configured and used:

- **UCM VOB feature level:** UCM PVOBs must be at feature level 7 in order to use CMI.
- **CQ OSLC Links package:** When using CMI with ClearQuest, the CQ record types in use must have the OSLC Links package applied to them. This can be done in the same manner as applying any other package in CQ. The OSLC Links package version should not have any impact on CMI functionality. If adding the OSLC Links package to an existing CQ schema or user database, the ClearQuest CM Server needs to be restarted after that change.

2. Configuring ClearCase VOBs

Before using the integration, it must be configured at the VOB level and the stream (UCM) or branch type level (base CC). Configuring a VOB for CMI involves making one or more attribute types and running a mkcmprovider command on the VOB. These attribute types are required in order to use CMI.

Use the **-shared** option to create the type in replicated VOBs, if more than one replica will be using the integration.

If the attribute type is created in an admin VOB, a local copy of the attribute type must be created in all participating client VOB families. The local copy of the type must be created at the site which masters the global type. A local copy can be made using the *cleartool cptype* command, for example, for the CC_CMI_TASK attribute type:

```
cleartool cptype -nc attype:CC_CMI_TASK@\adminvob CC_CMI_TASK@\clientvob
```

- **Base CC**
  Three attribute types must be created for a base CC VOB:
  - CC_CMI_CONTEXT
  - CC_CMI_PROVIDERS
  - CC_CMI_TASK

  Examples:
  ```
  cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@<VOB tag>
  cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@<VOB tag>
  cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@<VOB tag>
  ```

- **UCM**
  Only one attribute type needs to be made for a UCM Project VOB:
    - CC_CMI_PROVIDERS

  Example:
  ```
  cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\<PVOB
  tag>
  ```

- **mkcmprovider**
  After creating the appropriate attribute types it is necessary to run a mkcmprovider command on the VOB. The arguments required are the same for all Change Management adapter types (CQ, RTC, and JIRA). The contents of each argument will differ.
    - **-vob**
      This specifies the VOB we are configuring for CMI. In a UCM environment this will be the Project VOB.
    - **-version**
      The CMI adapter version. There is currently only one version – **v1_0**.
    - **-description**
      This can be used to enter custom details about this particular configuration. It should be enclosed in quotation marks.
    - **-type**
      Specifies which Change Management adapter type this provider is – CQ (cmcq), RTC (cmrtc), or JIRA (cmjira).
    - **-connection**
      The base URL for the Change Management provider must be part of this argument as a key:value pair. It can also contain any custom key:value pairs that do no conflict with keys used by the integration.
      The base URL key is case sensitive. The key for CQ and JIRA providers is **baseurl**. The key for RTC providers is **baseUrl**.
    - **Provider name**
      Each provider must be given a unique name to match between the VOB and the branch type or stream. It is possible to configure multiple providers on each VOB and/or branch type or stream.

  Examples:
  ```
  cleartool mkcmprovider -vob <VOB tag> -version v1_0 -description "RTC
  Provider" -type cmrtc -connection baseUrl:http[s]://<RTC
  server>  rtc_provider

  cleartool mkcmprovider -vob <VOB tag> -version v1_0 -description "CQ
  Provider" -type cmcq -connection baseurl:http[s]://<CQ Web
  server>/cqweb/oslc cq_provider

  cleartool mkcmprovider -vob <VOB tag> -version v1_0 -description "JIRA
  Provider" -type cmjira -connection baseurl:http[s]://<JIRA server>
  JIRA_provider
  ```

3. Configuring CM Provider

After running the mkcmprovider on the desired VOB it must also be run on either a branch type (base CC) or a stream (UCM) with the same provider name.

- Base CC
  - Mkcmprovider
    - **Brtype**
      When configuring a Change Management provider for a base CC environment it must be done on the VOB and on an existing branch type. Specify the branch type with this argument.
- UCM
  - Mkcmprovider
    - **-stream**
      When configuring a Change Management provider for a UCM environment it must be done on the Project VOB and on an existing stream. Specify the stream with this argument. Streams will inherit provider configurations from their parent streams by default unless overridden.
    - **-enable**
      This option can be used to enable or disable a provider on a specific stream. When a provider is disabled no CMI associations or policies will apply but existing associations will not be removed.
    - **-options**
      Use this to specify policies and other preferences for this provider that are not specific to the adapter type (RTC, CQ, or JIRA). Each option must be specified as a key:value pair. Options include:
    - **validate**
      Whenever a CC operation is executed that would perform or check for a task association with a version (checkin) or activity (setactivity), CMI will check with the Change Management system (remote server) to see if the task exists. If the task does not exist, the CC operation will be blocked. This is mostly useful when using the CLI as users must manually enter the provider task values in that case. The default value is **false** for Base CC and **true** for UCM.

      An example use-case for **validate** being **false** would be if all of your users are on the CLI and your Change Management system goes down or is unavailable. Setting **validate** to **false** would allow those users to continue working by performing checkin or setact and using any task ID they wish. Note that setting **validate** back to **true** will not make any new associations on the Change Management system if they were not able to be created while the system was unavailable. Any failed task associations that could not be performed while the remote system was unavailable will be logged in the **cmi_repair_log** which can be used to retry the task associations (see the "Repairing failed CMI actions" section).

Example:
```
-options validate:true
```

- **reqProvTask**
  This option enforces or relaxes the policy that for the CM provider configured on a stream/brtype an activity/version must have a task association.This policy is enforced at checkin (base CC) and setactivity (UCM). When this policy is enabled it is **req**uired to have a **prov**ider **task** association at checkin/setactivity for a version/activity. The default value is **true**.

  An example use-case for **reqProvTask** is to have it **false** on an integration stream where scripted jobs that set activities occur so as to not hinder their progress but have it **true** on the child streams where normal development occurs.

  Example:
  ```
  -options reqProvTask:true
  ```

- **reqAnyTask**
  This option enforces or relaxes the policy that for **any** CM provider configured on a stream/brtype an activity/version must have a task association. When this policy is enabled it is **req**uired to have **any task** association from at least one enabled provider at checkin/setactivity for a version/activity. The **reqProvTask** policy overrides (takes precedence over) **reqAnyTask**. The default value is **false**.

  With a single provider on a stream/brtype this policy behaves identically to **reqProvTask**. With multiple providers on a stream/brtype this policy allows them to act as if they were one provider in terms of task selection requirements. Meaning a CMI setup with this policy on two providers prov1 and prov2 will require a user to select a task from either prov1 or prov2 (inclusive) in order to satisfy the policy requirements.

  An example use-case for **reqAnyTask** would be if you have two Change Management systems (remote servers) that are used interchangeably for development work assignments. By setting up two providers on a brtype/stream and setting **reqAnyTask** to **true** for both (and **reqProvTask** to **false** for both) developers will be allowed to choose a task from either Change Management system for their version/activity.

  Example:
  ```
  -options reqAnyTask:true
  ```

- Base CC specific policies
  There are two policies available only for base CC providers (on a brtype):
  - **ciVerifyProv**
    This option enforces the policy that checkin should be blocked if the Change Management system server (CQ, RTC, or JIRA) is unavailable. When this policy is enabled and a **checkin** operation is performed, CMI will **verify** that the **provider** has access to the remote Change Management system. The default value is **false**.

    Example:
    ```
    -options ciVerifyProv:true
    ```

  - **ciVerifyUser**
    This option enforces the policy that checkin should be blocked if user authentication fails. When this policy is enabled and a **checkin** operation is performed, CMI will **verify** that the **user** authentication is valid. The default value is **false**.

    Example:
    ```
    -options ciVerifyUser:true
    ```

- UCM specific options
  - **activityFormat**
    This option is used to specify a default activity ID when creating an activity with a CMI task. The following macros are supported:

    %task-id – inserts the task's ID into the activity ID
    %stream-name – inserts the stream's name into the activity ID

    When using the **activityFormat** option the activity's Headline will also be populated from the task's corresponding field.

    CQ – Headline (customizable)
    RTC – Title
    JIRA - Summary

    Examples:
    ```
    activityFormat:CQ%task-id
    ```
    - `activity:CQSAMPL00000040`

    ```
    activityFormat:RTC%stream-name_%task-id
    ```
    - `activity:RTCmy-dev-stream_1234`

- **-context**
  The -context argument is used to specify details for the distinct (CQ, RTC, or JIRA) Change Management adapter. It can also be used to specify custom key:value pairs as long as the keys do not conflict with any keys used by CMI. It must contain a list of comma-separated key:value pairs. The CMI-defined keys are case-sensitive. The value string for the -context option should be enclosed in quotation marks.

- CQ

  It is necessary to specify at least two key:value pairs when configuring a CQ provider.
  - **userdb**
    The user database from CQ. This key is always required.
  - **dbset**
    The database set (or connection name) from CQ. This key is always required.
  - **queryuri**
    Used for presenting available tasks (records) for association to users through the CLI or GUI. The queryuri is used when **reqProvTask** or **reqAnyTask** are true for a given stream/brtype to present the user with a list of tasks to choose for association with their activity/version. It should be URL-encoded (e.g. spaces converted to %20). This key is not required but is strongly recommended as users must manually add associations with the command line without it.

    The queryuri can either be the full URL for a corresponding CQ OSLC query or just the specific query URI starting with the "rcm.name" parameter.

    Example:
    ```
    queryuri:http(s)://cqserver1/cqweb/oslc/repo/MYDBSET/db/MYDB/query/?rcm.name=Public%20Queries/All%20Defects

    queryuri:rcm.name=Public%20Queries/All%20Defects
    ```

  - **customQueryField**
    This option is only necessary when using a field other than “Headline” in the CQ record type. It is used to specify an alternate field for CMI to use for displaying query results.

- In the default CQ out-of-the-box schema, the “Headline” field is mapped to the OSLC field dcterms:title. Both fields are required for CMI, but the “Headline” requirement can be customized with this key.
  - Choose the ClearQuest field in your record type to use instead of "Headline." This field is displayed when you run a query through CMI.
  - Add that field to the "Query Presentation" of the ClearQuest query that you plan to use with CMI (this means that the field must be displayed when you run the query).
  - Map the field to dcterms:title in oslc-mappings.xml on the ClearQuest Web Server. Refer to the ClearQuest Knowledge Center for instructions: http://www-01.ibm.com/support/knowledgecenter/SSSH5A_8.0.0/com.ibm.rational.clearquest.cli.doc/topics/r_oslc_mappings_xml.htm.
  - For the **-context** option of the mkcmprovider command (on a branch type or UCM stream) set the key **queryuri** to the query you modified earlier and set the key **customQueryField** to the chosen ClearQuest field.

  Example:
  For the record type Defect, this example substitutes the field "Summary" for the field "Headline."

  Edit oslc-mappings.xml:

  ```
  <oslcRecordConfig type="cq.record:Defect@CQDBSET/DB1">
  <oslcFieldMapping name="dcterms:title" field="Summary"/>
  </oslcRecordConfig>
  ```

- Run the **mkcmprovider** command for a UCM stream or base ClearCase branch type, as applicable:

  UCM:
  ```
  cleartool mkcmprovider -stream dev1@\pvob1 -context
  "queryuri:query,customQueryField:Summary" ucm_provider
  ```

  Base CC:
  ```
  cleartool mkcmprovider -brtype main@\vob1 -context
  "queryuri:query,customQueryField:Summary" base_provider
  ```

- **cmtrans**
  This is used to perform state transitions on ClearQuest records when specific ClearCase actions (UCM: set activity or deliver complete; Base: checkout) are performed. This is done using the default transition for a given record type defined in the ClearQuest schema being used with CMI. This key is optional.

  Specifying a state transition is done with a 4-tuple of key:value pairs. Note that these key:value pairs are separate by semicolons (;) instead of commas (,).

  For both Base and UCM CC the transition keys are:

```
cmtrans:
[vobOp:<operation>;
  recordType:<record type>;
  startState:<record state>;
  endState:<another record state>]
[...]
```

  Each **[…]** block represents one transition rule. The entire **cmtrans** block must be specified on a single line with no breaks. As many **[**…**]** blocks as are necessary can be specified within the **cmtrans** key. In order for a **[**…**]** block to be valid it must contain the 4 following key:value pairs. If one of the key:value pairs below does not exist for given transition rule (**[**…**]**), the rule will be ignored.

- **vobOp**
  This is the ClearCase VOB operation for which the transition will occur. Valid choices for UCM are: **set_activity** and **deliver_complete**. Valid choices for Base ClearCase are: **checkout**.

  - **recordType**
    This specifies the ClearQuest record type to transition for the given **vobOp**. If you'd like to have the same transition defined for different CQ record types they must each be specified individually.

    Examples:
    `Defect, BaseCMActivity, etc.`

  - **startState**
    This indicates the starting state that a ClearQuest record must be in for a CMI transition to occur. If the ClearQuest record's state does not match the **startState**, that particular transition rule will be ignored but that will not prevent CMI from looking for other matching **startStates**.

    Examples:
    `Submitted, Assigned, Opened, or Resolved`

- **endState**
  This is the destination state that CMI will attempt to reach by using each record's default transitions. It can involve as many actions as are necessary to complete the transition and get to the destination state.

  Examples:
  ```
  Assigned, Opened, Resolved, or Closed
  ```

  CMI will stop performing a defined transition rule on a record if any of the following occur:

  - There is no default action for the current state.
  - There are unpopulated fields that must be populated in order to perform the default action.
  - There is an error from ClearQuest when performing a transition.
  - The record ends up back in the startState.
  - More than one vobOp, recordType, and startState triplet with the same values is defined.

  Examples:
  ```
  -context "userdb:<CQ DB>,dbset:<CQ DBSET>,queryuri:http[s]://<CQ
  Web URL>/cqweb/oslc/repo/<CQ DBSET>/db/<CQ DB>
  /query/?rcm.name=Public%20Queries/All%20Defects"

  -context "userdb:<CQ DB>,dbset:<CQ DBSET>,
  cmtrans:
  [vobOp:set_activity;
    recordType:Defect;
    startState:Submitted;
    endState:Assigned]
  [vobOp:deliver_complete;
    recordType:Defect;
    startState:Assigned;
    endState:Complete],
  queryuri:http[s]://<CQ Web URL>/cqweb/oslc/repo/<CQ DBSET>/db/<CQ
  DB>/query/?rcm.name=Public%20Queries/All%20Defects"
  ```

  Note: Line breaks included in the above example are for clarification purposes only and must not be included when a mkcmprovider command is run.

- RTC

  The **queryUri** is not required but is strongly recommended as users must manually add associations with the command line without it.

  The **cmtrans** key is optional. The **actions** key is optional – it is only used to specify custom state transition actions to take in RTC with **cmtrans**.

  - **queryUri**
    Used for presenting available tasks (work items) for association to users through the CLI or GUI. The queryUri is used when **reqProvTask** or **reqAnyTask** are true for a given stream/brtype to present the user with a list of tasks to choose for association with their activity/version. It should be URL-encoded (e.g. spaces converted to %20). Note the difference between RTC (queryUri) and CQ (queryuri).
  - **disableRTCBridge**
    This is used to disable the functionality that is automatically provided when CTE and RTC are installed in the same Eclipse shell. This is to allow CMI to have full control of the end user experience in CTE. See the "Compatibility with CTE/RTC Eclipse integration" section for more details.
  - **cmtrans**
    This is used to perform state transitions on RTC work items when specific ClearCase actions (UCM: set activity or deliver complete; Base: checkout) are performed. This is done using a set of default transition actions defined by CMI for a given work item type or by specifying your own with the **actions** key. Defining **cmtrans** for RTC uses the same format as CQ (see above) except the **recordType** key is replaced by the **workItemType** key.

    Specifying a state transition is done with a 4-tuple of key:value pairs. Note that these key:value pairs are separate by semicolons (;) instead of commas (,).

    For both Base and UCM CC the transition keys are:

    ```
    cmtrans:
    [vobOp:<operation>;
      workItemType:<record type>;
      startState:<record state>;
      endState:<another record state>]
    [...]
    ```

    Each **[**…**]** block represents one transition rule. As many **[**…**]** blocks as are necessary can be specified within the **cmtrans** key. In order for a **[**…**]** block to be valid it must contain the 4 following key:value pairs. If one of the key:value pairs below does not exist for given transition rule (**[**…**]**), the rule will be ignored.

- **vobOp**
  This is the ClearCase VOB operation for which the transition will occur. Valid choices for UCM are: **set_activity** and **deliver_complete**. Valid choices for Base ClearCase are: **checkout**.
- **workItemType**
  This specifies the RTC work item type to transition for the given **vobOp**. If you'd like to have the same transition defined for different RTC work item types they must each be specified individually.

  Examples:
  `Defect, Task, Story, Epic, Adoption Item, etc.`

- **startState**
  This indicates the starting state that a RTC work item must be in for a CMI transition to occur. If the work item's state does not match the **startState**, that particular transition rule will be ignored but that will not prevent CMI from looking for other matching **startStates**.

  Examples:
  `New, In Progress, or Reopened`

- **endState**
  This is the destination state that CMI will attempt to reach by using each work item's defined transition actions. It can involve as many actions as are necessary to complete the transition and get to the destination state.

  Examples:
  `In Progress, Resolved, or Verified`

- **actions**
  CMI provides several default actions for transitions that will automatically be used by any defined rules in **cmtrans**. There are predefined actions for the following work item types in the format *startState : defaultAction (human-readable name)*:

  - **Defect**
    New : com.ibm.team.workitem.defectWorkflow.action.startWorking (*Start Working*)
    In Progress : com.ibm.team.workitem.defectWorkflow.action.resolve (*Resolve*)
    Resolved : com.ibm.team.workitem.defectWorkflow.action.verify (*Verify*)
  - **Task**
    New : com.ibm.team.workitem.taskWorkflow.action.startWorking (*Start Working*)
    In Progress : com.ibm.team.workitem.taskWorkflow.action.resolve (*Resolve*)
  - **Story**
    New : com.ibm.team.apt.story.define (*Start Working*)

    In Progress : com.ibm.team.apt.storyWorkflow.action.a2 (*Complete Development*)
    Implemented : com.ibm.team.apt.storyWorkflow.action.a7 (*Complete Testing*)

  - **Epic**
    New : com.ibm.team.apt.epic.workflow.action.a1 (*Start Working*)
    In Progress : com.ibm.team.apt.epic.workflow.action.a7 (*Complete*)
  - **Adoption Item**
    Proposed : com.ibm.team.rtc.workflow.adoption.action.a2 (*Approve*)
    Approved : com.ibm.team.rtc.workflow.adoption.action.a3 (*Complete*)

    The CMI-provided default actions require no extra configuration beyond specifying the desired state transitions in the configuration of a branch type or stream. They are included with CMI whether or not you have defined any default actions of your own. The CMI-provided default actions are always used unless they are overridden by your configuration. If an applicable user-defined default action is found in the context of the provider configuration (see below), the CMI-provided default action is ignored.

    You can define your own actions that override the predefined CMI default actions. Defining actions to be used for RTC state transitions is similar to defining the state transitions.

Example:

```
actions:
[workItemType:Defect;
startState:In Progress;
defaultAction:com.ibm.team.workitem.defectWorkflow.action.resolve
;
resolution:3;
resolutionComment:Works as Designed]
```

More examples are included in the Examples section later.

Note: The 4-tuple key:value pairs are separate by semicolons (;) instead of commas (,). The above example is a 5-tuple due to the addition of a comment key. This key is not part of the Change Management Integration nor is it a reserved key. It is allowed as a convenience to make the resolution human-readable.

For both Base and UCM CC, the action keys are identical:

```
actions:
[workItemType:<work item type>;
  startState<work item state>;
  defaultAction:<RTC action to perform>;
  resolution:<optional resolution choice>]
[…]
```

- **workItemType**
  This specifies the Rational Team Concert work item type to apply the action to.

  Examples:
  `Defect, Task, Story, Epic, Adoption Item, etc.`

  If you'd like to have the same action defined for both Defect and Task they must each be specified individually.

- **startState**
  This indicates the starting state that a Rational Team Concert work item must be in to perform the given action. If a work item is not in the state specified when CMI attempts to perform the action on the work item, then there will be no action taken for that work item.

  Examples:
  `New, In Progress, Resolved, etc.`

- **defaultAction**
  This is the default action that will be taken for all transitions that use the **workItemType** and **startState** defined above. This must come from the Process Configuration Source of the RTC Project Area. The Process Configuration Source cannot be accessed through the RTC Web Client and must be accessed through the Eclipse Client.

  To access the Process Configuration Source, open the RTC client and open the relevant Project Area. There is a tab in the Project Area named “Process Configuration Source” that contains XML for the Project Workflow.

  Example for Defect and the New state:

```
<state group="open"
icon="processattachment:/workflow/open.gif" id="s1" name="New"
showResolution="false">

<action
id="com.ibm.team.workitem.defectWorkflow.action.startWorking"/
>

<action
id="com.ibm.team.workitem.defectWorkflow.action.resolve"/>

</state>
```

  The action id is used to define the **defaultAction** in CMI. In order to perform the “Start Working” action above the **defaultAction** would be “com.ibm.team.workitem.defectWorkflow.action.startWorking”, the **startState** would be “New”, and the **workItemType** would be “Defect”.

- **resolution**
  When an action would result in a state that also has a Resolution (e.g. Defect's Resolved state), users can specify the Resolution RTC should use. If you do not specify a Resolution, RTC will use the first Resolution in the list. For the work item type Defect, the available resolutions for the action "Resolve" are:

```
<action icon="processattachment:/workflow/resolve.gif"
id="com.ibm.team.workitem.defectWorkflow.action.resolve"
name="Resolve" state="s3">

<resolution id="r1"/>

<resolution id="r2"/>

<resolution id="r3"/>

<resolution id="r4"/>

<resolution id="r5"/>

<resolution id="r8"/>

</action>
```

  The resolutions are designated with numbers (in the default workflow) when specifying them in the CMI configuration. So using resolution id "r1" would be "1" in the actions. The readable names for the resolutions are also contained in the Process Configuration Source. In the example below the Resolution "r1" is "Fixed" and Resolution "r4" is "Works for Me":

```
<resolution icon="processattachment:/workflow/reject.gif"
id="r5" name="Invalid"/>

<resolution icon="processattachment:/workflow/works.gif"
id="r4" name="Works for Me"/>

<resolution icon="processattachment:/workflow/wontdo.gif"
id="r3" name="Works as Designed"/>

<resolution icon="processattachment:/workflow/duplicate.gif"
id="r2" name="Duplicate"/>

<resolution icon="processattachment:/workflow/close.gif"
id="r1" name="Fixed"/>

<resolution icon="processattachment:/workflow/unresolve.gif"
id="r0" name="Unresolved"/>

<resolution icon="processattachment:/workflow/close.gif"
id="r8" name="Fixed Upstream"/>
```

- JIRA
  - **queryuri**
    Used for presenting available tasks (issues) for association to users through the CLI or GUI. The queryuri is used when **reqProvTask** or **reqAnyTask** are true for a given stream/brtype to present the user with a list of tasks to choose for association with their activity/version. It should be URL-encoded (e.g. spaces converted to %20).
  - **Provider name**
    The provider name specified on a stream/brtype must match what was used to setup the same provider on the VOB.
  - Using a data file (**-data**)
    It is possible to store some of the CMI mkcmprovider arguments in a text file and use that file with the command. Enclosing quotation marks are not required in the data file. This option cannot be used in conjunction with **-type**, **-version**, **-description**, **-options**, or **-context**. The following arguments can be specified in a data file:
    - **version**
    - **type**
    - **description**
    - **connection.baseurl or connection.baseUrl**
    - **queryuri or queryUri**
    - **userdb**
    - **dbset**
    - **disableRTCBridge**
    - **cmtrans**
    - **actions**
    - **cmi_options.activityFormat**
    - **cmi_options.validate**
    - **cmi_options.reqProvTask**
    - **cmi_options.reqAnyTask**
    - **cmi_options.ciVerifyProv**
    - **cmi_options.ciVerifyUser**

Any other key=value pairs specified that are not recognized will be put in the provider's **context**. This allows custom key=value pairs to be specified with the **-data** option. Any custom keys must not contain "cmi_options", "connection" or match any of the keys used by CMI above.

Example:

```
type=cmcq
version=v1_0
description=CQ Provider

connection.baseurl=http://hostname.com/cqweb/oslc

userdb=CQWAN
dbset=SAMPL
queryuri=rcm.name=Public%20Queries/All%20Defects

cmtrans=[vobOp:set_activity;recordType:Defect;startState:Ready;en
dState:Complete][vobOp:set_activity;recordType:BaseCMActivity;sta
rtState:Submitted;endState:Assigned]

cmi_options.ciVerifyProv=true
cmi_options.ciVerifyUser=true
```

IV. CMI Administration

- Listing provider details
    - **lsprovider**
      This command can be used to describe the currently configured provider on a VOB, stream, or brtype. It supports a default level of detail, less detail with -short, and more detail with -long.
        - **VOB**

          Use the **lsprovider** command with the -vob option to describe the currently configured provider at the VOB level for base or UCM CC.

          Usage: **`lsprovider [-short | -long] {[-vob vob-selector[,...]][-replica replica-selector[,...]]} [provider_name ...]`**

          Example:
          ```
          cleartool lsprovider -long -vob test_vob test_prov
          ```

        - **UCM**

          Use the **lsprovider** command with the -stream option to describe the currently configured CM provider at the stream level.

          Usage: **`lsprovider [-short | -long] {[-stream stream-selector[,...]]} [provider_name ...]`**

          Example:
          ```
          cleartool lsprovider -short -stream test_dev@\test_proj
          ```

        - **Base CC**

          Use the **lsprovider** command with the -brtype option to describe the currently configured CM provider at the stream level.

          Usage: **`lsprovider [-short | -long] {[[-pname pname[,...]][-brtype brtype_name[,...]]]} [provider_name ...]`**

          Example:
          ```
          cleartool lsprovider -brtype main@\test_vob test_prov
          ```

- Making changes to an existing CMI configuration

  Existing provider configurations on VOBs, brtypes, or streams can be modified with the **mkcmprovider** command by using the -replace flag.

- Changing the VOB provider

  Specify the options to be modified (along with the -replace flag) as with a normal **mkcmprovider** command. If the -connection is being modified its entire contents must be provided as individual key:value pairs are not parsed by mkcmprovider. This will usually only include the **baseurl** (or **baseUrl**) key and its value but could include any custom key:value pairs.

  Usage: **mkcmprovider {-vob vob-selector | -replica replica-selector} -replace {-data prov-info-file | -type <type> -version <version> -description <description> -connection <connection_info>} provider_name**

  Examples:

  ```
  cleartool mkcmprovider -vob test_vob -replace -description "New
  description" test_prov
  cleartool mkcmprovider -vob test_vob2 -replace -connection
  "baseurl:http[s]://<server URL>/cqweb/oslc" test_prov2
  ```

- Changing the Branch Type or Stream provider

  This process is the same as changing the VOB provider but if the -context is being modified its entire contents must be provided. For CQ this includes the **userdb**, **dbset**, and **queryuri**.

  Usage: **mkcmprovider {-brtype <brtype-name>} -replace [{-data context-info-file | -context <context-string> [-options <cmi_options>]}] provider_name**

  mkcmprovider {-stream <stream name>} -replace {[-options <cmi_options>] -context <context_options> | -data [info_file]} {-enable [true|false]} provider_name

  Examples:

  ```
  mkcmprovider -brtype main@\test_vob -replace -options reqProvTask:false
  test_prov
  mkcmprovider -stream test_dev@\test_pvob -replace -context
  "queryuri:http[s]://<CQ server
  URL>/cqweb/oslc,dbset:9.0.0,userdb:SAMPL" test_prov
  ```

- Removing an existing CMI configuration

  Use the rmprovider command on either a VOB, brtype, or stream with the provider name to remove the provider configuration. Removing a provider on a VOB will also automatically remove the same provider at the stream/brtype level. Removing a provider on a brtype or stream will not automatically remove the same provider on the corresponding VOB. Removing a provider from a VOB or a stream/brtype will not remove associations already created on versions, activities, or Change Management system tasks.

  Usage: **`rmprovider {-vob vob-selector | -replica replica-selector | -brtype brtype-selector | -stream stream-selector} provider_name`**

  Example:
  ```
  cleartool rmprovider -vob test_vob test_prov
  ```

- Restricting access to **mkcmprovider** and **rmprovider** commands

  System administrators may wish to restrict access to these commands to avoid inadvertent or malicious use. There are two ways to do this: locks and triggers.

  If a ClearCase VOB is locked, no changes of any kind are allowed by anyone except users in the "Excluded users" list. For example, an administrator might lock a VOB excluding the administrative account, during the configuration process, so that no other user can make changes until the configuration is complete. Similarly, a branch or stream may be locked to prevent modification by unauthorized users. The use of locks is a coarse approach, because while the object is locked, no operations of any kind can take place on the object. Finer control may be accomplished with triggers.

  Triggers may be used to detect and control the use of the mkcmprovider and rmprovider commands on branch types and streams. In a base-ClearCase environment a trigger can be attached to the "mkattr" or "rmattr" operation on a branch type. In this case, the ClearCase trigger environment variable CLEARCASE_ATTYPE will be set to "CC_CMI_CONTEXT" when the trigger fires. In a UCM environment, a trigger can be attached to the "chstream" operation. In this case, the ClearCase trigger environment variable CLEARCASE_CMI_OP will be set to either "mkcmprovider" or "rmprovider."

- Repairing failed CMI actions

  When one of the following CMI actions fails:

  - Associate
  - Disassociate
  - State transition

  The details necessary to perform the action at a later time are saved in the **cmi_repair_log** file, which can be found in the CC log directory for any CCLC client or on the CCRC WAN Server machine for CCRC users. Using clmutil, the actions saved in **cmi_repair_log** can be run again as the user running the command (instead of the original user) to correct any discordance (e.g. missing associations on Change Management tasks).

  You must be set to a view when you run clmutil cmi_repair, and (except when using **-preview**) you must be root (on UNIX or Linux) or a member of the Administrator group and the ClearCase group on Windows.

  If desired, you may copy the file **cmi_repair_log** to another location before attempting the repair. In this case, specify the **-repair_file** option and enter the path name of the copied file.

  Note: Without the **-repair_file** option, clmutil will attempt to rename the **cmi_repair_log** file to **cmi_repair_log.backup** so that the original file may continue to collect new errors as they occur. The backup file will be used as the source for repair operations.

  Credentials must be registered with the cmiregister command before running the repair tool (see the cmiregister section below).

  The **-verbose** option displays additional diagnostic information.

  The **-preview** option displays what would have been repaired but does not actually perform the repairs.

  Usage: **`clmutil cmi_repair [-verbose] [-preview] [-migrate] [-repair_file pathname]`**

  Example:

  ```
  clmutil cmi_repair -verbose -repair_file cmi_repair_log
  ```

- When the Change Management system is unavailable

  The most reliable way of temporarily disabling CMI is to:

- UCM

  Set the -enable argument to **false** for the configured providers corresponding to the unavailable Change Management provider. This can be done with a **mkcmprovider -replace** command.

- Base CC

  Remove the provider from the impacted brtypes with an **rmprovider** command or create another attribute type for this purpose and **chtype** the provider attribute (to re-enable use **chtype** to change the attribute back to the original type).

V. Use

- Credential Registering

  When using CCLC (dynamic or snapshot views) credentials can be stored in an encrypted format in a user's home directory with the cmiregister command. For CTE the credentials can be read from the encrypted file or users can be prompted for their credentials when necessary.

  - **cmiregister**

    Using cmiregister is necessary when using the CLI with CMI. Each provider being used must have credentials registered for it. Registering credentials requires:

    - **Provider type**
      This will either be -cq, -rtc, or -jira depending on the corresponding provider on the VOB and stream/brtype.
    - **Provider name**
      This must match the name of the provider on the VOB and stream/brtype.
    - **Username**
      Enter the login name used for the Change Management system.
    - **Password**
      Enter the password used for the Change Management system. If no password is necessary, this does not need to be specified.

      Usage: **`cmiregister add -<provider type> -name <provider-name> [-userdb <database-name>] [-dbset dbset-name>] -username <username> [-password <password>]`**

Examples:

```
cmiregister add -cq -name test_cq -userdb SAMPL -dbset 9.0.0 -
username cquser -password cqpass
cmiregister add -rtc -name test_rtc -username rtcuser -password
rtcpass
```

- CTE in both CCRC and CCLC
  - CCLC (dynamic / snapshot views)
    - When using CTE in CCLC, stored credentials in **cmiregister** will automatically be used if they already exist.
    - If the stored credentials in **cmiregister** are incorrect or out of date the user will be prompted to login to the Change Management system using the same login prompts CC and CQ use and the stored credentials will be updated.
    - Credentials entered through CTE in CCLC without stored credentials in **cmiregister** are never stored permanently. They can be stored in memory temporarily in the same manner as remote CC or CQ credentials by checking the "Store and reuse credentials" box.

  - CCRC (automatic / web views)
    - When using CTE in CCRC, stored credentials in **cmiregister** are never used.
    - The user will be prompted to login to the Change Management system using the same login prompts CC and CQ use.
    - Credentials entered through CTE in CCRC are never stored permanently. They can be stored in memory temporarily in the same manner as CC or CQ credentials by checking the "Store and reuse credentials" box.

  - CCLC Native UI

    It is necessary to store encrypted credentials with the **cmiregister** command in order to use CMI with any of the CCLC native UIs (e.g. xclearcase or ClearCase Explorer).

- General Behavior and CLI
  - UCM
    - **mkactivity**

      Running a default mkact command will make use of the **activityFormat** for the activity ID and pull the task's headline for the activity Headling if the activity being created is given a CMI task to associate with. A task can be specified by using the -task flag with the mkact command. The mkact command cannot be blocked by CMI – meaning that even if the setact that normally follows mkact fails the activity will still be created.

      Usage: **mkactivity [-c comment | -cfile pname | -cq | -cqe | -nc] [-headline headline] [-in stream-selector] [-nset] [-force] *[-tasks task-selector[,...]]* [activity-selector ...]**

      Example:
      ```
      cleartool mkactivity -task SAMPL00000045@test_cq
      ```

- **setactivity**

  The setact command (including the setact done by mkact) will check the **reqProvTask** and **reqAnyTask** policies. If either are enabled for a provider on the stream, then the user will be required to have at least one association to a Change Management task for the activity before setact can succeed.

  For each enabled provider on the stream with **reqProvTask** enabled, the user must associate at least one task from that Change Management system in order to set the activity. If there are three providers on a stream (prov1, prov2, and prov3) and two of them (prov 1 & prov2) have **reqProvTask** enabled (and none have **reqAnyTask** enabled), then the user will be required to associate any activity they're performing setact on with at least one task from each of those two Change Management systems (TASK1@prov1 & TASK2@prov2).

  If **any** enabled provider on the stream has **reqAnyTask** enabled, the user must associate at least one task from **any** Change Management system in order to set the activity. If there are three providers on a stream (prov1, prov2, and prov3) and **any one** of them (prov3) has **reqAnyTask** enabled (and none have **reqProvTask** enabled), then the user will be required to associate any activity they wish to set to a view with at least one task from **any** of those **three** Change Management systems (TASK1@prov1 or TASK2@prov2 or TASK3@prov3).

- **lsactivity -find**

  The lsactivity -find command allows a user to find available Change Management tasks to associate with activities. A **queryuri** or **queryUri** must be defined in a provider's **context** (see above) in order to run this command. It uses the defined **queryuri** or **queryUri** for the provider specified (or all providers on the stream if none are specified) to generate a list of available Change Management tasks.

  Usage: **`lsactivity -find`** ***`[-provider provider_name]`*** **`[-in stream-selector | -view view-tag | -cview | activity-selector]`**

  Example:
  ```
  cleartool lsactivity -find -provider test_prov -cview
  ```

- **rmactivity**

  Removing an activity with an association will remove that association on the Change Management system task.

- Base CC
  - **settask**

    The **cleartool settask** command associates a default task, or list of default tasks, with a view. When a ClearCase element is created, checked out or checked in, the version created is associated by default with a task, or tasks, that have been previously associated with the view by this command. The settask command is only available via the command line but the currently set task will be used by the CC GUIs.

    The **cleartool settask -find** command allows a user to find available Change Management tasks to use with **cleartool settask** to associate with versions. A **queryuri** or **queryUri** must be defined in a provider's **context** (see above) in order to run this command. It uses the defined **queryuri** or **queryUri** for the provider specified (or all providers on the brtype if none are specified) to generate a list of available Change Management tasks.

    Usage:
    settask -find **[-provider provider_name] {-pname pname | -brtype brtype_name}**
    settask **[-view view-tag] {task-selector | [[-add_task task-selector[...] [-remove_task [task-selector[...]]] | -none }**

    Examples:
    1. Set task1, which is located at provider CQPROV, for the current view:
       `cleartool settask task1@CQPROV`
    2. Replace task1 with task2 as the default task for the current view. (Note: The simple form from example 1 should be used unless working with multiple providers):
       `cleartool settask -add task2@CQPROV -rem task1@CQPROV`
    3. Clear the default task from the current view. A new version (created through **cleartool mkelem** or **checkout**) must have its task set explicitly with **cleartool chtask** before it can be checked in:
       `cleartool settask -none`
    4. Find available tasks for one of the above examples:
       `cleartool settask -find -brtype main@\my_vob`

  - **lstask**

    The **cleartool lstask** command lists the default task(s) associated with a view.

    Usage: **`lstask -view view-tag | -cview`**

    Example: `cleartool lstask -cview`

- **chtask**

  The **cleartool chtask** command changes the task, or tasks, associated with an existing ClearCase version.

  Usage: **`chtask [-view view-tag] [-remove_task task-selector,...] [-add_task task-selector,...] pname`** …

  Example: Replace task1 with task2 on the current version of test.c:
  `cleartool chtask -rem task1@CQPROV -add task2@CQPROV test.c`

- **checkout**

  When performing a checkout, a user can optionally associate a task with the checked out version by having a currently set task for the view (achieved with the **settask** command above). The **reqProvTask** and **reqAnyTask** policies do not apply to checkout.

- **checkin**

  When performing a checkin, a user can associate a task with the version being checked in. This is accomplished by either having an association already on the checked out version (for the command line) or choosing a required task from the task selection dialog in a CC GUI. When using the command line, the checked out version must already have an association for the checked in version to also have an association. If the checked out version does not already have an association, one can be added with the **chtask** command.

  If there is an association for a given task on the checked out version and the checked in version is being associated with the same task, then the checked out version association will be removed and a new association will be added for the checked in version. The Change Management system task will essentially be updated to have an association with the checked in version.

- **uncheckout**

  Performing an uncheckout on a version with an association will remove that association from the Change Management system task.

- **rmver**

  Removing a version with an association will remove that association from the Change Management system task.

- CCLC Native UI (ClearCase Explorer or xclearcase)
  - UCM
    - **setactivity**
      Performing a setactivity, when **reqProvTask** or **reqAnyTask** are enabled (requiring a task association) and the activity does not already have associated tasks that satisfy **reqProvTask** or **reqAnyTask**, will display a prompt showing the results of the relevant provider queries (**queryuri** or **queryUri**) for the user to select one or more tasks from.
  - Base CC
    - **checkin**

      Performing a checkin, when **reqProvTask** or **reqAnyTask** are enabled (requiring a task association) and the version being checked in does not already have associated tasks that satisfy **reqProvTask** or **reqAnyTask**, will display a prompt showing the results of the relevant provider queries (**queryuri** or **queryUri**) for the user to select one or more tasks from.

- CTE (RCP and Eclipse Extension Offering)
  - Local view credential prompting

    When using CTE with a CCLC view type (dynamic or snapshot) the user can either use **cmiregister** (see above) or be prompted by CTE to enter their Change Management system credentials. Out of date credentials stored with **cmiregister** will be updated by the CTE prompt if valid credentials are provided.

  - Remote view credential prompting

    When using CTE with a CCRC view type (web or automatic) the user will always be prompted by CTE to enter their Change Management system credentials. The **cmiregister** tool is not used for remote views.

  - UCM
    - **setactivity**

      Performing a setactivity, when **reqProvTask** or **reqAnyTask** are enabled (requiring a task association) and the activity does not already have associated tasks that satisfy **reqProvTask** or **reqAnyTask**, will display a prompt showing the results of the relevant provider queries (**queryuri** or **queryUri**) for the user to select one or more tasks from.

- Base CC
  - **checkin**

    Performing a checkin, when **reqProvTask** or **reqAnyTask** are enabled (requiring a task association) and the version being checked in does not already have associated tasks that satisfy **reqProvTask** or **reqAnyTask**, will display a prompt showing the results of the relevant provider queries (**queryuri** or **queryUri**) for the user to select one or more tasks from.

VI. Compatibility with CTE/RTC Eclipse integration

When CTE and RTC are installed into the same Eclipse Shell an integration (also known as the CTE/RTC Bridge) is automatically enabled with similar behavior to UCM CMI. It allows associations to be made between activities and work items. This integration is controlled entirely by the end user in their Eclipse preferences. This means that the end user decides whether or not to require task associations and which tasks are available for association.

When CMI is configured for UCM with RTC, the CTE/RTC Eclipse integration can pick up some settings from CMI to offer some control over the task requirements for the end user.

- The **reqProvTask** setting on the CMI configuration overrides any settings the user changes in their Eclipse preferences.
- The **activityFormat** specified in CMI is used by the CTE/RTC Eclipse integration.

It is possible to disable the functionality offered by the CTE/RTC Bridge Eclipse integration when using CMI with the disableRTCBridge keyword.

VII. Comparison of UCM/CQ Integration (SQUID) and CMI

Local and remote connections

- CMI
  With all available adapters (CQ, RTC, and JIRA), CMI uses a web connection [http(s)] to contact the Change Management server.
- SQUID
  Prior to 9.0, SQUID allowed both local and web connections to CQ. As of 9.0, only web connections are allowed.

Note: For either integration (CMI or web connections in SQUID) to work properly with ClearQuest:

- ClearCase must be installed on the ClearQuest Web Server.
- The ClearCase installation on the client must point to the same ClearCase Registry as the ClearCase installation on the ClearQuest Web Server so that the VOBs can be found.

Project and stream configuration

- CMI
  Each CMI provider is configured at a stream level with inheritance for child streams.
- SQUID
  There is only configuration at the UCM project level – meaning all streams under that UCM project must use the integration.

Base CC support

- CMI
  The same functionality for UCM is provided for Base CC through CMI.
- SQUID
  There is no Base CC functionality for this integration.

New record creation

- CMI
  If a new record is required, the user must create one in CQ before attempting a CC operation requiring an association.
- SQUID
  The web connection does not have this capability.

CTE support

- CMI
  The integration is fully supported in CTE but it does not directly utilize the CQ functionality provided by CTE.
- SQUID
  Local connections are no longer supported through CTE. Web connections are fully supported in CTE and include direct usage of CQ including WorkOn and running CQ queries.

StartWork from CQ

- CMI
  There is no direct connection from CQ back to CC with CMI so the StartWork option is not available.
- SQUID
  StartWork can be used to set a CQ record as the current activity from a CQ local client. This option is not available in CQ Web.

Displaying CQ records instead of activities

- CMI
  Activities are displayed as they normally would be without the integration.
- SQUID
  Activities cannot be displayed and CQ records are instead always used.

Linking activities to records

- CMI
  The integration creates links between the activity and the record (task).
- SQUID
  The IDs and Headlines are made to match between the activity and CQ record and the stream, VOB, and view fields are populated in the record.

CQ server connection to CC

- CMI
  There is no direct connection from the CQ server to CC.
- SQUID
  The UCM package involves calls back to CC from CQ for verification and activity update purposes.

UCM Package

- CMI
  The only CQ package required and used by CMI is the OSLC Links package.
- SQUID
  The UCM Package is required for this integration and enforces strict control over the related fields – including the UCM project, stream, view, and fetching the activity's change set.

Policy support

- CMI
  The policies available in CMI mostly govern task selection for a given activity or version and blocking setact/checkin if the requirements are not met.
- SQUID
  The policies here are for performing various actions in CQ via scripts either before or after specific CC operations.

Policy GUI

- CMI
  There is no GUI to modify the CMI policies.
- SQUID
  The CC Project Explorer GUI allows modification of the integration's policies. They can also be modified with local CQ clients (not web).

Querying for records

- CMI
  Records can be queried by running a **settask -find** command or perfoming an action in a CC GUI when a record (task) is required. This is only possible when a **queryuri** is defined in the stream or brtype configuration.
- SQUID
  Records are fetched using a pre-defined query.

Custom query support

- CMI
  The query is fully customizable – any valid query can be used.
- SQUID
  There is limited customization available – only through CTE can non-default queries be run for a web connection.

State transitions

- CMI
  All state transitions are optional and customizable.
- SQUID
  Specific state transitions are done automatically and some are available with policies.

Custom state transitions

- CMI
  Since there are no automatically enforced transitions with CMI they are all customizable.
- SQUID
  State transitions cannot be customized.

Trigger support

- CMI
  Configuring CMI can be restricted with triggers on **mkcmprovider** and **rmprovider**.
- SQUID
  Configuring SQUID can be restricted with triggers on **chproj** or **mkproj**.

Configuration inheritance

- CMI
  Configurations (providers) from parent streams are automatically inherited by child streams. An explicit configuration on a child stream will disable the inheritance
- SQUID
  Since the integration is configured on a per-project basis there is no inheritance.

Multiple CM server connections

- CMI
  Multiple Change Management providers can be configured on each stream/brtype.
- SQUID
  Only one CQ database can be linked per project.

Non-CQ CM support

- CMI
  In addition to CQ adapter support, CMI also supports RTC and JIRA adapters.
- SQUID
  There is no support beyond CQ.

Smart card authentication

- CMI
  Smart card authentication is supported for Change Management systems through (embedded) GSKit.
- SQUID
  There is no smart card authentication support for CQ through SQUID.

Discordance repair

- CMI
  The cmi_repair_log keeps track of failed actions (associations, disassociations, and transitions) that can be repaired later with clmutil.
- SQUID
  A scan can be run to correct out of sync activities/records with ucmutil.

Atomic checkin

- CMI
  When performing an atomic checkin from a GUI, CMI provides an “apply to all” option for task selection.
- SQUID
  There is no atomic checkin support.

Migration from Perl V2 Base CC integration

- CMI
  All existing links from the Perl V2 Base CC integration can be migrated to the CMI format.
- SQUID
  There is no migration from the Base CC integration to SQUID.

VIII. Migrating from the V2 Perl Base ClearCase Integration to CMI

CMI supports a two-step process for migrating from the V2 Perl integration to CMI.

- In the first step, a new utility, cmi_migrate.pl, is used to generate a file listing all versions that require task associations.
- In the second step, the clmutil utility is run in migrate mode to parse the generated file and create the task associations on the versions and to update the links tab on the ClearQuest records.

The migration does not remove the CrmRequest hyperlinks that were created by the V2 integration, but it is recommended that the V2 integration be disabled after the migration process has been completed.

Apply the OSLC Links package to the database used in the integration. This package is required for CMI to be configured with ClearQuest.

Running cmi_migrate.pl.

- We recommend that the VOB be locked -nusers for the User running the utility, so that other users do not accidentally generate new hyperlinks that will then have to be parsed. You must run the utility in a view, using ratlperl. Note that you may need to modify your PATH to include /opt/rational/common/bin.

Run **ratlperl cmi_migrate.pl**.

- The cmi_migrate utility will request that you enter the desired mode; enter “base” to migrate from the V2 perl integration. The cmi_migrate utility will then request that you enter the vobtag of the VOB you wish to migrate, the name of an existing temp directory, and names for two files it will generate (default names are suggested). It will then parse the V2 integration's config.pl file, and store extracted information in a file with the (default) name cmi_migration.txt, in the specified temp directory.
- At this point you can choose to enter CMI configuration information for all of the brtypes that require migration, or you can quit the utility and modify the cmi_migration.txt file manually. If you choose to quit, you can run the script again to resume the migration process. If you choose to continue, you will be prompted for a description for the CMI provider, a query uri, and a name for the provider. Once the configuration is complete, cmi_migrate will create attribute types required by CMI, will run the mkcmprovider command to configure the VOB and the participating brtypes, and will generate a migration version information file with the (default) name cmi_migration_versions.txt, that will contain entries for all VOB element and directory versions that require task associations in CMI.

Running clmutil in migrate mode.

- Once the migration version information file has been generated, you can use the clmutil utility to generate the actual task associations. This utility must be run as root on UNIX or as Privileged User on Windows. It is installed in <ClearCase-home>/bin
- If your VOB has ACLs enabled, you must ensure that User has permissions to modify elements. For example, you may use the cleartool chpolicy command to add permissions for root to the DefaultPolicy:
  ```
  > cleartool chpolicy -nc -kind element -add User:root -permission Full
  DefaultPolicy
  ```
- You must be set to a view when you run clmutil in migrate mode. Any user can preview its actions by running:
  ```
  > clmutil cmi_repair -verbose -preview -migrate -repair_file
  <migrate_versions_file_pname>
  ```
- The migrate_versions_file_pname must be the full path to the version information file generated by cmi_migrate.pl. In preview mode clmutil will list all of the CMI actions that would be attempted.
- Before creating the task associations, you must use the cmiregister command to store the necessary credentials for CMI. As root or Privileged User run
  ```
  > cmiregister add -cq -name <provider-name> -userdb <database-name> -
  dbset <dbset-name> -username <username> [-password <password>]
  ```
- Supply a password only if one is needed by your ClearQuest installation.
- To perform the actual work, run the utility as root or Privileged User, in a view, without the -preview option:
  ```
  > clmutil cmi_repair -verbose -migrate -repair_file
  <migrate_versions_file_pname>
  ```
- You will see output for each association that is created. After the utility completes, you can describe a version on an instance of a participating brtype to view the associated tasks, and you can check the Links tab on the ClearQuest record to view the version information. The migration process does not remove the hyperlinks that used as artifacts for the V2 Perl integration.

IX. Migrating from the UCM/CQ Integration (SQUID) to CMI

The Change Management Integration (CMI) supports a two-step process for migrating from SQUID to CMI.

- In the first step, the utility cmi_migrate.pl is used to generate a file listing all activities in a user-specified project VOB that require task associations, based upon a user-specified list of CQ-enabled projects to migrate. It also performs cleartool operations to configure the project VOB and the streams associated with the specified projects for CMI, based upon user input.
- In the second step, the clmutil utility is run in migrate mode to parse the generated file and create the task associations on the activities and to update the Links tab on the ClearQuest records.

Apply the OSLC Links package to the database used in the integration. This package is required for CMI to be configured with ClearQuest.

General considerations when migrating from SQUID to CMI:

- The migration does not remove the artifacts that were created by the SQUID integration, but it is recommended that the SQUID integration be placed in suspend mode or disabled after the migration process has been completed. Note that disabling the SQUID integration is irreversible and will remove all artifacts it has created in the VOB. If you do not yet want to disable the SQUID integration, you can instead suspend it temporarily. This can be done with the following command:
```
>cleartool chproject -crmenable user_dbname [-force] -suspend
project_selector
```
- Suspending the SQUID integration will keep all of the existing artifacts in place but will not create any new ones. Note that resuming the SQUID integration will prompt you to sync any unlinked activities to new CQ records. A suspended SQUID integration can be resumed with the following command:
```
>cleartool chproject -crmenable user_dbname [-force] -resume
project_selector
```
- A view context is required to resume a suspended SQUID integration. The -force option skips both the confirmation to suspend or resume the integration as well as the confirmation to sync each unlinked activity to a new CQ record (when resuming). If the -force option is not specified, then ClearCase will prompt the user to confirm each activity individually.

Running the cmi_migrate.pl.

- We recommend that the project VOB be locked -nusers for the User running the utility, so that other users do not accidentally generate new SQUID artifacts that will then have to be parsed. You must run the utility in a view context, using ratlperl. Note that you may need to modify your PATH, for example, on UNIX to include /opt/rational/common/bin.

```
> ratlperl cmi_migrate.pl
```

- The cmi_migrate utility will request that you enter the desired mode; enter “ucm” to migrate from the SQUID integration. The utility will then request you to enter the VOB tag of the VOB you wish to migrate, the name of an existing temp directory, names for two files it will generate (default names are suggested), and a list of projects you wish to migrate. The utility will display a list of all streams in those projects. In the following UNIX example, User responses appear in italics.

```
>Enter mode: 'base' for base-ClearCase, 'ucm' for UCM, or return to
quit: ucm
>Enter the VOB tag to migrate or return to quit: /test309_pvob
>VOB tag validated.
>Enter a temp directory for use with the migration or return to quit:
/var/tmp
>Enter a file name to store migration configuration information or
return to use the default (cmi_migration.txt): <CR>
>Enter a file name to store migration activity information or return to
use the default (cmi_migration_activities.txt):<CR>
>Enter a comma-separated list of projects to migrate (e.g.,
"proj1,proj2") or return to quit:
>test309
>Validate project test309@/test309_pvob is clearquest enabled for user
database SAMPL.
>Extracting information from crmregister ...
>Found the user database
SAMPL.
>Finding streams in project test309@/test309_pvob ...
test309_int@/test309_pvob
test309_dev@/test309_pvob
```

- At this point you can choose to enter CMI configuration information for all of the streams that require migration, or you can quit the utility and modify the cmi_migration.txt file manually. If you choose to quit, you can run the script again to resume the migration process. If you choose to continue, for each stream found in the specified projects, you will be asked if you wish to configure it now, configure it manually later, or skip configuration for it. You would choose to skip configuring a child stream, for instance, if you wanted it to inherit CMI configuration from its parent stream. (If you explicitly configure a child stream in CMI, it will not inherit any settings from its parent stream.) If you choose to configure the stream now, you will be prompted for a description for the CMI provider, a Query URI, and a name for the provider.

```
>Do you wish to enter CMI configuration for streams now? [y]
>Enter 'n' to enter the information manually later.
> y
>Note that child streams inherit CMI configuration from their parent
streams unless you explicitly configure them.
>Do you want to configure stream test309_int@/test309_pvob?
>Enter 'y' to configure it now, 'n' to enter the information manually
later, or 's' to skip configuring it.
>y
>Please enter the remaining information for:
Stream:test309_int@/test309_pvob
UserDB:SAMPL
DBSet:9.0.0
Server URL:http(s)://some-host-name:12080/cqweb/oslc
>Enter the description (e.g. "CQ Web Server 1"):"New CQ Provider"
>Enter the query URI (e.g. "rcm.name=Public%20Queries/All%20Defects")
or return to enter no query URI:
rcm.name=Public%20Queries/All%20Defects
>Enter the Provider Name (e.g. "CQPROV1"): CQPROV1
>Do you want to configure stream test309_dev@/test309_pvob?
>Enter 'y' to configure it now, 'n' to enter the information manually
later, or 's' to skip configuring it.
>s
```

- Once the configuration is complete, cmi_migrate will create the attribute type required by CMI, will run the mkcmprovider command to configure the VOB and the participating streams, and will generate a migration activities information file with the (default) name cmi_migration_activities.txt, that will contain entries for all VOB activities that require task associations in CMI.

Manually completing entries in the cmi_migration.txt file.

- If you choose to manually enter configuration details for a stream, cmi_migrate will create a partially populated entry in the cmi_migration.txt file. You must complete the entry with a Description of the CQ Provider, a Query URI, and a provider Name. The Query URI is optional, and this field may be left blank. The completed entries should be separated by a blank line:

```
Stream:test309_int@\test309_pvob
UserDB:SAMPL
DBSet:9.0.0
Server URL:http(s)://some-host-name:12080/cqweb/oslc
Version:v1_0
Description:"New CQ Provider"
Query:rcm.name=Public%20Queries/All%20Defects
Name:CQPROV1

Stream:test309_dev@\test309_pvob
UserDB:SAMPL
DBSet:9.0.0
Server URL:http(s)://some-host-name:12080/cqweb/oslc
Version:v1_0
Description:”New CQ Provider”
Query:
Name:CQPROV1
```

Running clmutil in migrate mode.

- Once the migration activities information file has been generated, you can use the clmutil utility to generate the actual task associations. This utility must be run as root on UNIX or as Privileged User on Windows. It is installed in <ClearCase-home>/bin.
- You must be in a view context when you run clmutil in migrate mode. Any user can preview its actions by running:

```
> clmutil cmi_repair -verbose -preview -migrate -repair_file
<migrate_activities_file_pname>
```

- The migrate_activities_file_pname must be the full path to the activities information file generated by cmi_migrate.pl. In preview mode clmutil will list all of the CMI actions that would be attempted.
- Before creating the task associations, you must use the cmiregister command to store the necessary credentials for CMI. As root or Privileged User run:

```
> cmiregister add -cq -name <provider-name> -userdb <database-name> -
dbset <dbset-name> -username <username> [-password <password>]
```

- Supply a password only if one is needed by your ClearQuest installation.
- To perform the actual work, run the utility as root or Privileged User, in a view context, without the -preview option:

```
> clmutil cmi_repair -verbose -migrate -repair_file
<migrate_activities_file_pname>
```

- You will see output for each association that is created. After the utility completes, you can describe an activity in a participating stream to view the associated tasks, and you can check the Links tab on the ClearQuest record to view the activity information. The migration process does not remove the artifacts that were created on the activities by the SQUID integration.

X. Examples

Base CC with CQ

- Pre-existing VOB base_vob_1

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_1

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_1

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_1

cleartool mkcmprovider -vob vob:\base_vob_1 -version v1_0 -description
"CQ Web Server 1" -type cmcq -connection
baseurl:https://cqweb1:12443/cqweb/oslc cq_test_1

cleartool mkcmprovider -brtype main@\base_vob_1 -options
reqProvTask:true -context
"queryuri:https://cqweb1:12443/cqweb/oslc/repo/9.0.0/db/SAMPL/query/?rcm.name=Public%20Queries/All%20Defects,dbset:9.0.0,userdb:SAMPL" cq_test_1

cmiregister add -cq -name cq_test_1 -userdb SAMPL -dbset 9.0.0 -
username test_1 -password pass_1
```

This example sets up one CQ provider, cq_test_1, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin.

Other branch types are not impacted and won't be able to perform CMI associations without their own CMI configuration. Additional branch types can reuse the VOB level configuration, e.g:

```
cleartool mkcmprovider -brtype mine@\base_vob_1 -options
reqProvTask:true -context
"queryuri:https://cqweb1:12443/cqweb/oslc/repo/9.0.0/db/SAMPL/query/?rcm.name=Public%20Queries/All%20Defects,dbset:9.0.0,userdb:SAMPL" cq_test_1
```

Base CC with CQ (using -data for mkcmprovider commands)

- Pre-existing VOB base_vob_1

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_1

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_1

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_1

cleartool mkcmprovider -vob vob:\base_vob_1 -data
cmi_base_cq_data.txt  cq_test_1

cleartool mkcmprovider -brtype main@\base_vob_1 -data
cmi_base_cq_data.txt cq_test_1

cmiregister add -cq -name cq_test_1 -userdb SAMPL -dbset 9.0.0 -
username test_1 -password pass_1
```

- cmi_base_cq_data.txt contents:

```
type=cmcq
version=v1_0
description= CQ Web Server 1

connection.baseurl=https://cqweb1:12443/cqweb/oslc
queryuri=https://cqweb1:12443/cqweb/oslc/repo/9.0.0/db/SAMPL/query/?rcm
.name=Public%20Queries/All%20Defects

dbset=9.0.0
userdb= SAMPL
cmi_options.reqProvTask=true
```

This example sets up one CQ provider, cq_test_1, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin.

Base CC with CQ State Transitions

- Pre-existing VOB base_vob_2

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_2

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_2

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_2

cleartool mkcmprovider -vob vob:\base_vob_2 -version v1_0 -description
"CQ Web Server 2" -type cmcq -connection
baseurl:https://cqweb2:12443/cqweb/oslc cq_test_2

cleartool mkcmprovider -brtype main@\base_vob_2 -options validate:true
-context "
cmtrans:[vobOp:checkout;recordType:Defect;startState:Submitted;endState
:Assigned],queryuri:https://cqweb2:12443/cqweb/oslc/repo/TEST/db/SAMPL/query
/?rcm.name=Personal%20Queries/test%20CMI,dbset:TEST,userdb:SAMPL" cq_test_2

cmiregister add -cq -name cq_test_2 -userdb SAMPL -dbset TEST -username
test_2 -password pass_2
```

This example sets up one CQ provider, cq_test_2, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin since the default value for **reqProvTask** is **true**. It will also **validate** that any associated tasks exist at the Change Management server during checkin.

With the given **cmtrans** configuration (for state transitions), any time a **checkout** is performed with an associated CQ **Defect** in the **Submitted** state the record will be transitioned to the **Assigned** state.

Base CC with RTC

- Pre-existing VOB base_vob_3

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_3

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_3

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_3

cleartool mkcmprovider -vob vob:\base_vob_3 -version v1_0 -description
"RTC Server 3" -type cmrtc -connection baseUrl:https://rtcserver3:9443/ccm
rtc_test_3

cleartool mkcmprovider -brtype main@\base_vob_3 -context "queryUri:
web/projects/cmi%20project%20(Change%20Management)#action=com.ibm.team.
workitem.runSavedQuery&id=_rwGz4FeIEeOjR5YNkvoFJg&refresh=true"
rtc_test_3

cmiregister add -rtc -name rtc_test_3 -username test_3 -password pass_3
```

This example sets up one RTC provider, rtc_test_3, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin since the default value for **reqProvTask** is **true**.

Base CC with RTC State Transitions

- Pre-existing VOB base_vob_4

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_4

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_4

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_4

cleartool mkcmprovider -vob vob:\base_vob_4 -version v1_0 -description
"RTC Server 4" -type cmrtc -connection baseUrl:https://rtcserver4:9443/ccm
rtc_test_4

cleartool mkcmprovider -brtype main@\base_vob_4 -context "queryUri:
web/projects/cmi%20project%20(Change%20Management)#action=com.ibm.team.
workitem.runSavedQuery&id=_rwGz4FeIEeOjR5YNkvoFJg&refresh=true,
cmtrans:[vobOp:checkout;workItemType:Task;startState:New;endState:In
Progress][vobOp:checkout;workItemType:Defect;startState:New;endState:In
Progress]" rtc_test_4

cmiregister add -rtc -name rtc_test_4 -username test_4 -password pass_4
```

This example sets up one RTC provider, rtc_test_4, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin since the default value for **reqProvTask** is **true**.

It will also attempt to perform state transitions for **Tasks** and **Defects** on checkouts.

If an associated **Task** is in the **New** state when a checkout is performed, then CMI will attempt to transition it to the **In Progress** state using the CMI-defined default actions.

If an associated **Defect** is in the **New** state when a checkout is performed, then CMI will attempt to transition it to the **In Progress** state using the CMI-defined default actions.

Note that even though the state transitions for **Task** and **Defect** appear to be the same they must each be defined separately.

Base CC with JIRA

- Pre-existing VOB base_vob_5

```
cleartool mkattype -shared -nc -vtype string CC_CMI_CONTEXT@\base_vob_5

cleartool mkattype -shared -nc -vtype string CC_CMI_TASK@\base_vob_5

cleartool mkattype -shared -nc -vtype string
CC_CMI_PROVIDERS@\base_vob_5

cleartool mkcmprovider -vob vob:\base_vob_5 -version v1_0 -description
"JIRA Server 5" -type cmjira -connection
baseurl:http://jiraserver5:8080 jira_test_5

cleartool mkcmprovider -brtype main@\base_vob_5 -context
"queryuri:http://jiraserver5:8080/rest/api/latest/search?jql=project+%3D+IP+ORDER+
BY+summary+ASC%2C+updatedDate+DESC" jira_test_5

cmiregister add -jira -name jira_test_5 -username test_5 -password
pass_5
```

This example sets up one JIRA provider, jira_test_5, on the branch type main. The provider will require task associations with versions on that branch type when performing a checkin since the default value for **reqProvTask** is **true**.

UCM with CQ

- Pre-existing PVOB pvob_6, pre-existing integration stream int_6 and child development stream dev_6

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_6

cleartool mkcmprovider -vob \pvob_6 -version v1_0 -description "CQ Web
Server 6" -type cmcq -connection baseurl:https://cqweb6:12443/cqweb/oslc
cq_test_6

cleartool mkcmprovider -stream int_6@\pvob_6 -options
validate:true,activityFormat:CQ%task-id -context
"userdb:DB1,dbset:CQDB,queryuri:https://cqweb6:12443/cqweb/oslc/repo/CQDB/d
b/DB1/query/?rcm.name=Public%20Queries/All%20Defects" -enable "true"
cq_test_6

cmiregister add -cq -name cq_test_6 -userdb DB1 -dbset CQDB -username
test_6 -password pass_6
```

This example sets up on CQ provider, cq_test_6, on the integration stream int_6. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**. It will also **validate** that any associated tasks exist at the Change Management server during setact.

When creating a new activity with the -tasks option, a user can use the automatically generated ID to create an activity with an ID of CQ<record ID> and the record's Headline as the activity headline.

Since the integration stream (int_6) has a CMI configuration all of its child streams will inherit that configuration unless they are explicitly overridden. This means that the dev stream (dev_6) will have the same behavior and will show the inherited provider with an **lsprovider** command.

UCM with CQ (using -data for mkcmprovider commands)

- Pre-existing PVOB pvob_6, pre-existing integration stream int_6 and child development stream dev_6

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_6

cleartool mkcmprovider -vob \pvob_6 -data cmi_ucm_cq_data.txt cq_test_6

cleartool mkcmprovider -stream int_6@\pvob_6 -data ucm_cq_data.txt -
enable "true" cq_test_6

cmiregister add -cq -name cq_test_6 -userdb DB1 -dbset CQDB -username
test_6 -password pass_6
```

- cmi_ucm_cq_data.txt contents:

```
type=cmcq
version=v1_0
description=CQ Web Server 6

connection.baseurl=https://cqweb6:12443/cqweb/oslc

cmi_options.validate=true
cmi_options.activityFormat=CQ%task-id

userdb=DB1
dbset=CQDB
queryuri=https://cqweb6:12443/cqweb/oslc/repo/CQDB/db/DB1/query/?rcm.na
me=Public%20Queries/All%20Defects
```

This example sets up on CQ provider, cq_test_6, on the integration stream int_6. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**. It will also **validate** that any associated tasks exist at the Change Management server during setact.

When creating a new activity with the -tasks option, a user can use the automatically generated ID to create an activity with an ID of CQ<record ID> and the record's Headline as the activity headline.

Since the integration stream (int_6) has a CMI configuration all of its child streams will inherit that configuration unless they are explicitly overridden. This means that the dev stream (dev_6) will have the same behavior and will show the inherited provider with an **lsprovider** command.

UCM with CQ State Transitions

- Pre-existing PVOB pvob_7, pre-existing integration stream int_7 and child development stream dev_7

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_7

cleartool mkcmprovider -vob \pvob_7 -version v1_0 -description "CQ Web
Server 7" -type cmcq -connection
baseurl:https://cqweb7:12443/cqweb/oslc cq_test_7

cleartool mkcmprovider -stream int_7@\pvob_7 -options
activityFormat:CQ%task-id -context "userdb:DB1,dbset:CQDB,
cmtrans:[vobOp:set_activity;recordType:Defect;startState:Submitted;endS
tate:Assigned][vobOp:deliver_complete;recordType:Defect;startState:Assi
gned;endState:Complete],queryuri:https://cqweb7:12443/cqweb/oslc/repo/CQDB/
db/DB1/query/?rcm.name=Public%20Queries/All%20Defects" -enable "true"
cq_test_7

cmiregister add -cq -name cq_test_7 -userdb DB1 -dbset CQDB -username
test_7 -password pass_7
```

This example sets up one CQ provider, cq_test_7, on the integration stream int_7. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

When creating a new activity with the -tasks option, a user can use the automatically generated ID to create an activity with an ID of CQ<record ID> and the record's Headline as the activity headline.

If an associated **Defect** is in the **Submitted** state when a setactivity is performed on the associated activity, then CMI will attempt to transition the **Defect** to the **Assigned** state using the CQ-defined default actions.

If an associated **Defect** is in the **Assigned** state when a deliver complete is performed on the associated activity, then CMI will attempt to transition the **Defect** to the **Complete** state using the CQ-defined default actions.

UCM with RTC

- Pre-existing PVOB pvob_8, pre-existing integration stream int_8 and child development stream dev_8

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_8

cleartool mkcmprovider -vob \pvob_8 -version v1_0 -description "RTC
Server 8" -type cmrtc -connection
baseUrl:https://rtcserver8:9443/ccm rtc_test_8

cleartool mkcmprovider -stream int_8@\pvob_8 -context "queryUri:
web/projects/CMI%20Project%206.0%20(Change%20Management)#action=com.ibm
.team.workitem.runSavedQuery&id=_boaAwIe_EeW0Co056aprIQ&refresh=true" -
enable "true" rtc_test_8
```

cmiregister add -rtc -name rtc_test_8 -username test_8 -password pass_8

This example sets up one RTC provider, rtc_test_8, on the integration stream int_8. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

UCM with RTC State Transitions

- Pre-existing PVOB pvob_9, pre-existing integration stream int_9 and child development stream dev_9

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_9

cleartool mkcmprovider -vob \pvob_9 -version v1_0 -description "RTC
Server 9" -type cmrtc -connection
baseUrl:https://rtcserver9:9443/ccm rtc_test_9

cleartool mkcmprovider -stream int_9@\pvob_9 -context "queryUri:
web/projects/CMI%20Project%206.0%20(Change%20Management)#action=com.ibm
.team.workitem.runSavedQuery&id=_boaAwIe_EeW0Co056aprIQ&refresh=true,cm
trans:[vobOp:set_activity;workItemType:Defect;startState:New;endState:I
n Progress][vobOp:deliver_complete;workItemType:Defect;startState:In
Progress;endState:Resolved][vobOp:set_activity;workItemType:Defect;star
tState:Reopened;endState:In
Progress],actions:[workItemType:Defect;startState:In
Progress;defaultAction:com.ibm.team.workitem.defectWorkflow.action.reso
lve;resolution:3;resolutionComment:Works as
Designed][workItemType:Defect;startState:Reopened;defaultAction:com.ibm
.team.workitem.defectWorkflow.action.startWorking]" -enable "true"
rtc_test_9

cmiregister add -rtc -name rtc_test_9 -username test_9 -password pass_9
```

This example sets up one RTC provider, rtc_test_9, on the integration stream int_9. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

It will also attempt to perform state transitions for **Defects** on setactivity and deliver -complete.

If an associated **Defect** is in the **New** state when a setactivity is performed, then CMI will attempt to transition it to the **In Progress** state using the CMI-defined default actions.

If an associated **Defect** is in the **In Progress** state when a deliver -complete is performed, then CMI will attempt to transition it to the **Resolved** state using the user-defined action.

Note that the user-defined action here also has a comment under the resolutionComment key to indicate what “resolution:3” refers to. The resolutionComment key is not required and is not part of the CMI reserved keywords used for setup.

If an associated **Defect** is in the **Reopened** state when a setactivity is performed, then CMI will attempt to transition it to the **In Progress** state using the user-defined action.

UCM with RTC State Transitions (using -data for mkcmprovider commands)

- Pre-existing PVOB pvob_9, pre-existing integration stream int_9 and child development stream dev_9

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_9

cleartool mkcmprovider -vob \pvob_9 -data cmi_ucm_rtc_data.txt
rtc_test_9

cleartool mkcmprovider -stream int_9@\pvob_9 -data cmi_ucm_rtc_data.txt
-enable "true" rtc_test_9

cmiregister add -rtc -name rtc_test_9 -username test_9 -password pass_9
```

- cmi_ucm_rtc_data.txt contents:

```
type=cmrtc
version=v1_0
description=RTC Server 9

connection.baseUrl=https://rtcserver9:9443/ccm

queryuri=web/projects/CMI%20Project%206.0%20(Change%20Management)#actio
n=com.ibm.team.workitem.runSavedQuery&id=_boaAwIe_EeW0Co056aprIQ&refres
h=true

cmtrans=[vobOp:set_activity;workItemType:Defect;startState:New;endState
:In Progress][vobOp:deliver_complete;workItemType:Defect;startState:In
Progress;endState:Resolved][vobOp:set_activity;workItemType:Defect;star
tState:Reopened;endState:In Progress]

actions=[workItemType:Defect;startState:In
Progress;defaultAction:com.ibm.team.workitem.defectWorkflow.action.reso
lve;resolution:3;resolutionComment:Works as
Designed][workItemType:Defect;startState:Reopened;defaultAction:com.ibm
.team.workitem.defectWorkflow.action.startWorking]
```

This example sets up one RTC provider, rtc_test_9, on the integration stream int_9. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

It will also attempt to perform state transitions for **Defects** on setactivity and deliver -complete. See the example preceding this one for more details on the state transitions CMI will attempt.

UCM with JIRA

- Pre-existing PVOB pvob_10, pre-existing integration stream int_10 and child development stream dev_10

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_10

cleartool mkcmprovider -vob \pvob_10 -version v1_0 -description "JIRA
Server 10" -type cmjira -connection baseurl:http://jiraserver10:8080
jira_test_10

cleartool mkcmprovider -stream dev_10@\pvob_10 -context
"queryuri:http://jiraserver10:8080/rest/api/latest/search?jql=project+%3D+IP+ORDER
+BY+summary+ASC%2C+updatedDate+DESC" -enable "true" jira_test_10

cmiregister add -jira -name jira_test_10 -username test_10 -password
pass_10
```

This example sets up one JIRA provider, jira_test_10, on the development stream dev_10. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

UCM with JIRA (using -data for mkcmprovider commands)

- Pre-existing PVOB pvob_10, pre-existing integration stream int_10 and child development stream dev_10

```
cleartool mkattype -shared -nc -vtype string CC_CMI_PROVIDERS@\pvob_10

cleartool mkcmprovider -vob \pvob_10 -data cmi_ucm_jira_data.txt
jira_test_10

cleartool mkcmprovider -stream dev_10@\pvob_10 -data
cmi_ucm_jira_data.txt -enable "true" jira_test_10

cmiregister add -jira -name jira_test_10 -username test_10 -password
pass_10
```

- cmi_ucm_jira_data.txt contents:

```
type=cmjira
version=v1_0
description=JIRA Server 10

connection.baseurl=http://jiraserver10:8080

queryuri=http://jiraserver10:8080/rest/api/latest/search?jql=project+%3
D+IP+ORDER+BY+summary+ASC%2C+updatedDate+DESC
```

my_custom_key=my_custom_value

This example sets up one JIRA provider, jira_test_10, on the development stream dev_10. This provider will require task associations with activities when they are set to a view since the default value for **reqProvTask** is **true**.

This example also contains a custom key:value pair in the data file that will be stored in the **context** and will appear with an **lsprovider** command:

```
>cleartool lsprovider -long -stream dev_10@\pvob_10
stream  "dev_10"
change management provider  "jira_test_10"
enabled: true
type  "cmjira"
version  "v1_0"
description : JIRA Server 10
baseurl : http://jiraserver10:8080
my_custom_key : my_value
queryuri :
http://jiraserver10:8080/rest/api/latest/search?jql=project+%3D+IP+ORDER+BY+summ
ary+ASC%2C+updatedDate+DESC reqAnyTask : false reqProvTask : true validate : true
```

XI. Troubleshooting

CMI log

CMI messages are logged in <CC_VAR>\log\cmi_log. Where <CC_VAR> = <ClearCase Home>\var (Windows).

Any errors logged may indicate configuration and/or Change Management operation errors.

As some errors may lead to potential discordance between ClearCase and the Change Management server (CQ, RTC, or JIRA) with respect to version association, periodic review of the log file for information usable for off-line resolution is recommended.

ClearCase CMI tracing

The CMI libraries that directly interact with the rest of ClearCase have trace settings that are enabled and used in the same manner as other CC subsystems. The subsystem here is CMI. The TRACE_VERBOSITY can be up to 5.

Example (Windows):

```
set TRACE_VERBOSITY=4
set TRACE_SUBSYS=CMI
```

ClearCase CMI tracing with CTE

The CMI CC tracing can be enabled when using CTE by putting the trace settings in the server.conf file located in: <ClearCase Home>\config\ccrc\server.conf. This file can be created if it does not exist. These trace settings work with remote (on the CCRC WAN Server) and local (on the client machine) view types.

Add the following settings to enable CMI tracing:

```
ccrcTraceLevel=5
ccrcTraceSubsystem=CMI
```

The traces will go in the <CC_VAR>\log\trace\<username> directory.

CM Adapter tracing

The CMI libraries that contact the Change Management server (CQ, RTC, or JIRA) have their own tracing settings that can be enabled with mkcmprovider on a VOB as part of the -connection argument.

The tracelevel can be up to 5. The tracesubsystems are: CMOSLC, CMCQ, CMRTC, and CMJIRA. The CMOSLC system applies to CQ, RTC, and JIRA. The remaining three subsystems apply to CQ, RTC, and JIRA (respectively). Each subsystem being traced is separated by a colon (:).

The traces will be written to the <CC_VAR>\log\cm_trace\<username> directory. The user being traced will need to have write permission for the cm_trace directory.

Example:

```
cleartool mkcmprovider -vob <VOB_TAG> -version v1_0 -description "CQ trace" -
type cmcq -connection
"baseurl:http://cqtraceserver:12080/cqweb/oslc,tracelevel:4,tracesubsystem:CM
OSL…" CQ_PROV
```

SSL certificates

Please refer to the following technote for SSL specific issues:
https://www.ibm.com/support/pages/node/541765