QAD Adaptive Applications

User Guide

QAD Customer Version Control

70-3382-2.8
QAD Customer Version Control 2.8
October 2020

CVC_UG_v028.pdf/sl8

QAD Inc.
100 Innovation Place
Santa Barbara, California 93108
Phone (805) 566-6000
https://www.qad.com

Contents

QAD CVC User Guide Change Summary

Product Name Changes

Starting in September 2019, the new name for QAD's complete portfolio of products is QAD Adaptive Applications. Additionally, QAD Adaptive ERP is the new name for QAD's flagship ERP solution. QAD Adaptive ERP includes the functionality previously associated with QAD Cloud ERP and QAD Enterprise Applications - Enterprise Edition, plus the QAD Enterprise Platform and Adaptive UX which resulted from the Channel Islands program. Going forward, the terms QAD Enterprise Applications, QAD Cloud ERP, and Channel Islands will be deprecated but will remain in previous documentation and training materials. QAD's intention is to—as soon as possible—eliminate the use of the deprecated terms going forward.

Change Summary

The following table summarizes significant differences between this document and previous versions.

Date/Version	Description	Reference
October 2020/2.8	Updated for CVC 2.8.	--
	Updated the typical customization process.	page 10
	Added information about the XREF index analysis process.	page 32
	Updated information for the `cvc download` command.	page 49
	Updated information for the `cvc report` command.	page 64
	Added information for the `cvc resolve-all` command.	page 72
April 2019/2.7.0.60	Updated for CVC 2.7.0.60.	--
	Added information about the overlap detecting process.	page 20, page 71
	Added information about the file mapping management process.	page 24, page 50
December 2018/2.6.1.0	Rebranded for CVC 2.6.1.0.	--
November 2018/2.6.0.1	Initial version.	--

Chapter 1

QAD CVC Overview

This chapter provides a functional and architectural overview of QAD Customer Version Control (CVC).

Introduction to Customer Version Control

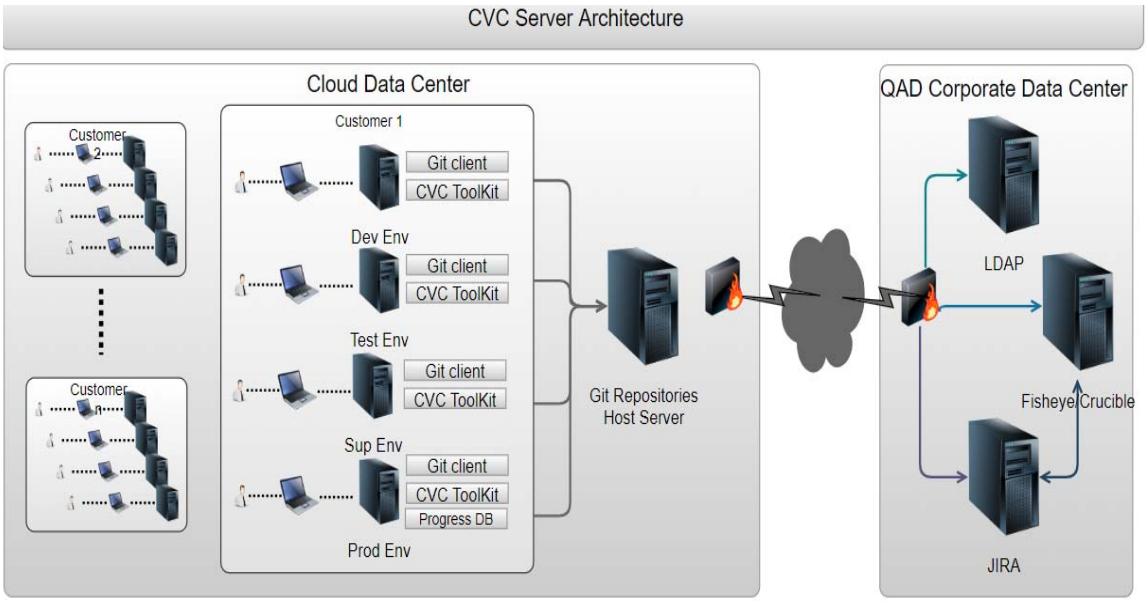
QAD Customer Version Control, also known as QAD CVC, is a version control system (VCS) for software delivery. It provides a controlled, traceable, and secure process for the maintenance, update, and delivery of software within customer environments.

QAD CVC is an integrated tool chain providing version control for QAD service software development life cycle (SSDLC). It is specifically tailored for the QAD customization development process.

Architecture

QAD CVC is a command line toolkit and is based on Git. It requires a Progress database to store all the metadata. The CVC toolkit should be installed in customer environments including development, test, support, and production. CVC pushes changes to a remote Git repository hosted by QAD, and this repository is connected to QAD's JIRA and Fisheye systems.

Fig. 1.1
Architecture of QAD CVC

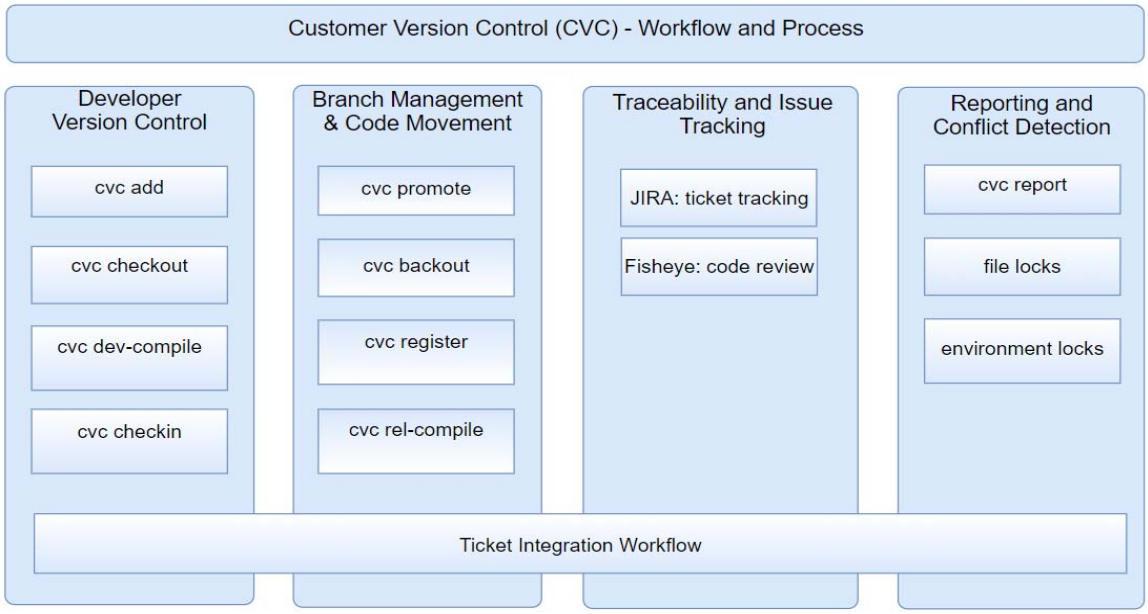


Key Features

QAD CVC provides the following key features:

- Developer version control
- Branch management and code movement
- Traceability and issue tracking
- Reporting and conflict detection
- QAD compile tools integration

Fig. 1.2
Key Features of QAD CVC



Developer Version Control

The developer version control process is the foundation layer of the CVC system. The CVC toolkit records changes to a file or a set of files over time so that you can recall specific versions later. It helps developers increase productivity and enhance code quality while improving communication between different development teams.

QAD CVC also allows environment-breaking changes to be reverted, quickly restoring functionality. In addition, developer version control enables users to set up a standardized development process, making sure that all file changes are linked to valid tickets. This also ensures code integrity, making sure that code cannot be accidentally overwritten by others in the future.

Branch Management and Code Movement

QAD CVC has embedded a special branching and merging strategy built atop Git for source control, with specific attention paid toward the goals of quality, integrity, and ease of development. It allows you to pick file changes by ticket and apply them easily in the test and production environments. It also checks dependencies between tickets so you can make sure not to promote partial ticket changes if multiple tickets have changed the same file.

Traceability and Issue Tracking

QAD CVC keeps a full history of content changes. It tracks every change made in the environments detailed with the modifier information, modified date, and the changed content itself.

QAD CVC provides granular access control to CVC commands. This is regulated by user roles and environments, thus ensuring that only the right people can make changes to the code. QAD CVC also stores all activity history in the database, allowing users to view full command history through provided reports.

QAD CVC is integrated with a suite of commonly used tools for issue tracking, namely, QAD JIRA and Fisheye systems. JIRA automatically updates issues and transitions work when code is committed in CVC. Developers can directly create new code reviews from JIRA tickets and use the QAD Fisheye system to review the file changes online.

Reporting and Conflict Detection

QAD CVC provides various reports based on the commit history and metadata stored in the database. The reports can be exported to files in several different formats and can be sent to email addresses as attachments directly.

QAD CVC can help developers to analyze the impact of a version upgrade or new installation of a standard product or patch release. It can generate a report listing all custom files that may be affected by applying the incoming update. The listed files should then be reviewed and revised to ensure that they work with the new updated files.

For more details about available reports, see “Report Process” on page 18.

QAD Compile Tools Integration

QAD CVC is integrated with all major QAD compile tools, including YAB, QDT, and MFGUTIL. The compilation process is triggered after CVC commits file changes in the environment. The compile propath is predefined in the metadata for each file, so users can make sure the files are always compiled with the same compile propath in all customer environments. If the compilation fails, CVC immediately rolls back the file changes to the previous version to keep the environment in a clean and usable status.

CVC Basics

User Roles

QAD CVC has a well-organized permission control system based on user roles. Users can only run the set of commands relevant to their roles. Administrator users can entitle each role to different sets of CVC commands in different environments. One user can be assigned with multiple roles.

The following roles are predefined in CVC:

- Developer

  Developer users can access the commands related to their development work—for example, `cvc checkout`, `cvc checkin`, and `cvc report`.

- Release manager

  Release manager users can use commands like `cvc promote` and `cvc backout` to make changes to the non-development environments.

- Reporter

Reporter users can only run reporting-related commands, and they do not have access to make code changes to the environments.

- Administrator

  Administrator users manage all the CVC metadata information, such as users, roles, modules, environments, and so on.

Environment Model

Generally, for a CVC customer, there are three environments that need to be maintained with one code base. The environments are named `devl` (for development), `test` (for testing), and `prod` (for production). Those environments are predefined in the CVC database. CVC allows administrator users to define more environments for multi-site customers, but this document only describes the simple environment model.

New customizations can only be added in the development environment, which should always be the first entry in the environment list. The other environments only accept code changes through the `cvc promote` command. This command can help users pick the changes by ticket and apply them easily to the target environment.

Code changes can only be promoted from one specific environment to one target environment. The former environment is defined as the target environment's pre-environment in the system metadata. Users cannot change the pre-environment once it is set. For example, the `test` environment's pre-environment is `devl`, so every change in the `test` environment should be from the `devl` environment.

The typical workflow is as follows:

1. Developers check in new changes in the `devl` environment.
2. Then, after code review and approval, release managers can promote the code changes from `devl` to `test` for testing purpose.
3. Finally, if the changes work as expected, release managers can promote them from `test` to `prod` to put them online.

Module Definition

The concept module in CVC includes the detailed information of a file object, including parent directory and all compilation-related information. All the files in the version control system should be linked to a module. Users need to specify the module when they try to add new files into version control. This makes sure the files are placed in the correct directory and are always compiled with the same settings across the environments.

Module fields vary with environment types. For example, the `yab-process` and `code-instance` fields are available only in YAB environments. You can check the list of all the modules in the current environment with the `cvc module-show` command. Administrator users can add more modules and update existing modules with the `cvc module` command.

Table lists all module fields and their descriptions.

Table 1.1
Module Fields

Field	Description	Example
name	Module name	mfgpro_cust
type	Module type (standard or custom)	custom
root-dir	Root directory to store source code	mfgpro/xxsrc
src-dir (optional)	Unencrypted source directory (relative to `root-dir`)	src
xrc-dir (optional)	Encrypted source directory (relative to `root-dir`)	xrc
work-dir (optional)	Work directory to put rcode (relative to the environment directory)	mfgpro/xxsrc/work
yab-process (optional)	YAB process to compile	code-mfg-update
code-instance (optional)	Name of the YAB code instance	mfg
rcode-dir (optional)	Directory to put rcode files (relative to the environment directory)	mfgpro
compile-propath (optional)	Propath for compile (separated by commas)	mfgpro/xxsrc/src, mfgpro/xrc, mfgpro/src
compile-pf (optional)	Compile pf file (relative to deploy tool's directory)	scripts/batchCompile.pf
languages (optional)	Languages code to compile module	us
rcode-layout (optional)	rcode layout (three options available: `staggered`, `flat`, or `source`)	staggered
source-layout (optional)	Source layout (two options available: `twoletter` or `flat`)	twoletter
before-compile (optional)	Scripts to run before compilation	
after-compile (optional)	Scripts to run after compilation	

Getting Help

If you need help while using QAD CVC, there are two ways to get the comprehensive manual page (manpage) help for any CVC command:

```
$ cvc <verb> --detailed-help
$ man cvc-<verb>
```

For example, you can get the manpage help for the `cvc checkout` command by running the following command:

```
$ cvc checkout --detailed-help
```

You can access the commands anywhere, even offline. If the manual page or this document fails to help solve your issues and you need in-person help, you can try sending an email to the CVC forum (forum_customer_version_control@qad.com).

In addition, if you do not need the full-blown manual page help, but just need a quick refresher on the available options for a certain CVC command, you can ask for the more concise “help” output with the `-h` or `--help` options, as in:

```
$ cvc checkout -h
Customer Version Control -- checkout tool (2.6.0.0)
usage: cvc checkout <files>... -t <arg> [-l <arg>] [-f] [-c] [-n <arg>] [-e <arg>] [-h] [-
dh] [-y]
```

`-l,--list-file <arg>`	List containing the files to be checked out.
`-f,--force`	Forcibly check out files despite any lock.
`-c,--cancel`	Cancel the file checkout operation.
`-n,--note <arg>`	Note to check out files.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Chapter 2

QAD CVC Usage

This chapter provides detailed instructions on using QAD Customer Version Control.

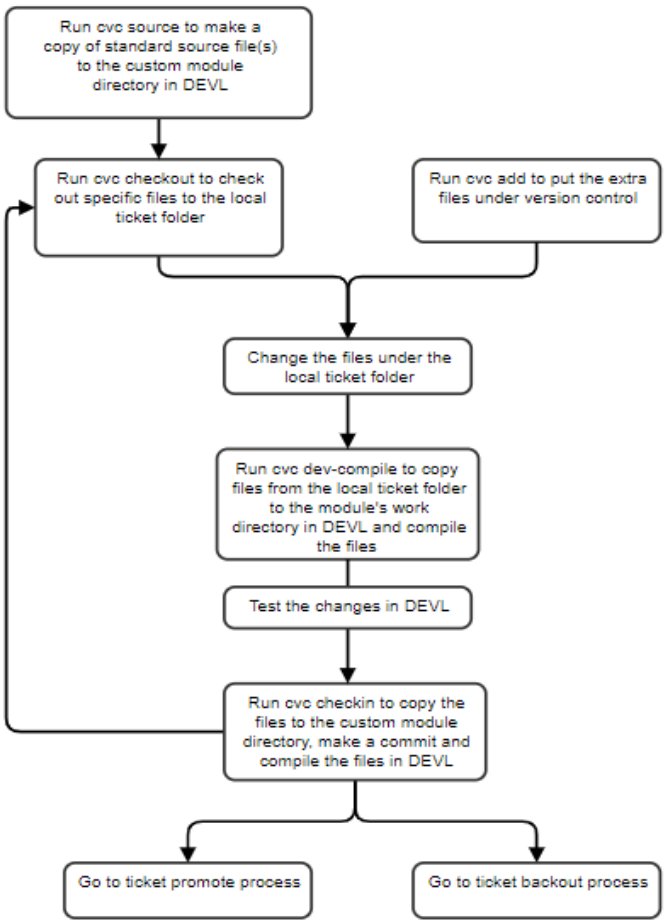
Typical Customization Process

This section describes how to perform a typical customization in the customer environment through CVC.

Process Diagram

Figure 2.1 illustrates the main customization process.

Fig. 2.1
Typical Customization Process Diagram



Process Description

A typical customization involves the following operations:

- Sourcing a standard file

  Users can run the `cvc source` command to copy one or more standard files to the customized module directory.

  The `cvc source` command requires users to specify the source filename and a custom module name.

Users can either add a custom prefix to the new file or use the origin filename. For example, `us/so/sosorp.p` becomes `us/xx/xxsosorp.p` after users add a custom prefix `xx`.

If the source files are from the ticket directory, CVC validates them with the standard source files in the custom module's propath. If the files are valid, CVC copies them to the custom module's `src` directory and commits them.

CVC also creates the file mapping relationship between the custom file and its source file. The file mapping information is then used for the overlap detecting process. (For details, see "Overlap Detecting Process" on page 20.)

Users can specify the `--auto` option to let CVC find the correct source file based on the custom module's propath. Otherwise, users should manually prepare the correct source file under the ticket folder.

Users can specify the `--copy-local` option to let CVC find the correct source file based on the custom module's propath and then copy the file to the local ticket folder.

- Checking out an existing file

  Users can run the `cvc checkout` command to check out the files under the ticket folder.

  CVC locks all the files that have been checked out to prevent other users from checking them out.

  Users can specify the `--force` option to use forcible checkout. This means the users acquire a file lock.

  If user A checks out a file and user B then forcibly checks out the same file, user A needs to perform a checkout again. If there are conflicts, CVC shows a warning message and waits for a manual merge.

  Users can specify the `--cancel` option to cancel the checkout operation and release the file lock.

- Adding a new file

  Users can run the `cvc add` command to add custom files to a custom module, but only brand-new files can be added by using this command.

  The `cvc add` command requires users to specify a module for the new files. The module determines the target folder to hold those files in the system as well as all compilation-related information of the files. Users can view all module information by running the `cvc module-show` command.

  CVC validates and reports an error if there are existing files with the same module name and filename in the system.

  CVC prompts for the source filename to create the file mapping relationship. The file mapping information is then used for the overlap detecting process. (For details, see "Overlap Detecting Process" on page 20.)

- Performing quick compilation

  Users can run the `cvc dev-compile` command to copy the files to the module's work directory and quickly compile the files.

  Users need to run the `cvc add` or `cvc checkout` command before running the `cvc dev-compile` command.

  In YAB environments, CVC runs YAB commands with the module's compilation process and YAB handles the code compilation and rcode deployment.

In QDT or MFGUTIL environments, CVC prepends the module's work directory in the module propath, triggers compilation, and deploys the rcode files.

- Checking in changes

  Users can run the `cvc checkin` command to check in changes.

  The `cvc checkin` command requires a commit message and a ticket number. The ticket is then used in future promotion or backout.

  Users need to run the `cvc add` or `cvc checkout` command before running the `cvc checkin` command.

  If the file is checked out by using the `cvc checkout` command, CVC checks whether the user has a file lock.

  If the file is added by using the `cvc add` command, CVC checks whether the file already exists in the system.

  The `cvc checkin` command copies the files to the module directory, makes a commit, and compiles the files. The corresponding existing files in the module's work directory are deleted before the compilation.

- Completing development

  If development completes and tests pass, developers can ask the customer to review and approve the ticket to be promoted to the test environment. When receiving the approval, the Cloud SDLC team becomes responsible for the ticket promotion process. (For details, see “Ticket Promotion Process” on page 13.)

  If the code changes in the ticket need to be reverted, developers can go through the ticket backout process. (For details, see “Ticket Backout Process” on page 15.)

Example

To source a custom file `us/xx/xxsosorp.p`, which is generated after users add a custom prefix `xx` to the `us/so/sosorp.p` file, to the module `CUSTOM-MFG`, run:

```
$ cvc source --ticket CVC-100 --module CUSTOM-MFG --auto us/so/sosorp.p
```

Note Before running the `CVC source` command, the user needs to create the ticket folder `CVC-100` under the user's home directory.

To check out the custom file `us/xx/xxsosorp.p` with ticket CVC-100, run:

```
$ cvc checkout --ticket CVC-100 us/xx/xxsosorp.p
```

After making some changes, to add a new file `us/xx/xxnewfile.p` to the module `CUSTOM-MFG`, run:

```
$ cvc add --ticket CVC-100 --module CUSTOM-MFG us/xx/xxnewfile.p
```

Note Before running the `CVC add` command, the user needs to create the ticket folder `CVC-100` under the user's home directory and prepare the file `us/xx/xxnewfile.p` in the ticket folder `CVC-100`.

To compile local changes within the work directory, run:

```
$ cvc dev-compile
```

To check in the changes with a note, run:

```
$ cvc checkin --ticket CVC-100 --note "Add xxnewfile.p and change xxsosorp.p"
```

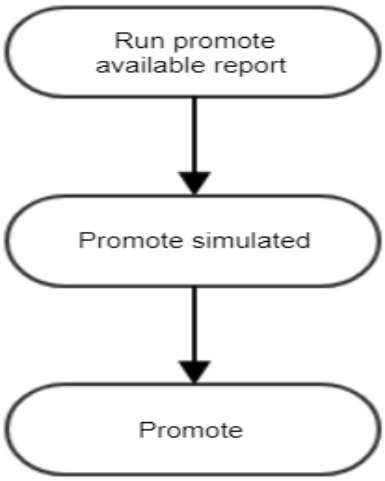
Ticket Promotion Process

This section explains the basic process of promotion and also discusses some special cases and errors that users may encounter when using the `cvc promote` command.

Process Diagram

Figure 2.1 illustrates the ticket promotion process.

Fig. 2.2
Ticket Promotion Process Diagram



Process Description

A typical ticket promotion involves the following operations:

- Obtaining the list of tickets available for promotion

  Users can run the `cvc promote --available` command to obtain the list of tickets available for promotion.

- Performing a simulated promotion

  Users can specify the `--simulate` option to perform a simulated promotion.

  The `cvc promote` command requires a ticket list and a promotion target environment, and uses tickets as promotion units.

  The `cvc promote` command checks the environment settings to determine the source environment. The pre-environment of the target environment is treated as the source environment.

  Users can specify multiple tickets within one `cvc promote` command to promote the tickets together.

- Performing promotion

  QAD recommends that users run a simulated promotion before the real one. In this way, users can check the file change list through the simulation first.

  The `cvc promote` command applies the ticket-related changes from the source environment to the target environment.

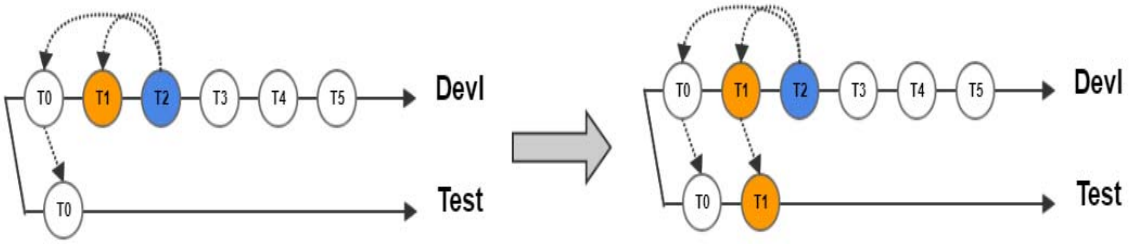
The `cvc promote` command verifies ticket dependencies before conducting promotion.

The `cvc promote` command compiles the file changes during promotion.

The normal promotion process is completed if no error occurs. The following discusses some special cases and errors in promotion.

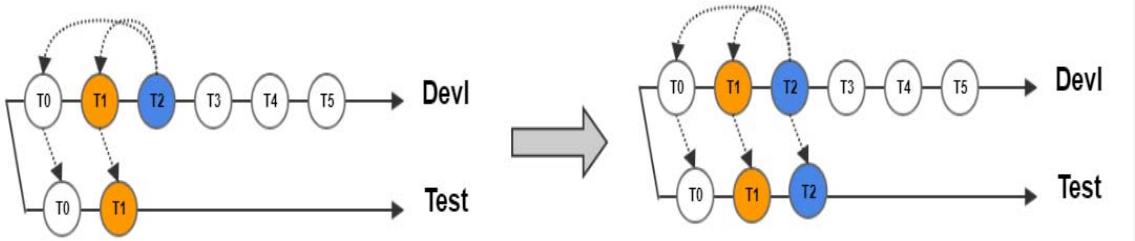
Example **Promoting dependent tickets**

Fig. 2.3
Promoting Dependent Ticket T1



Assume that T2 has dependencies with T1 and T0, and now you want to promote T2 to the `test` branch. There are two feasible ways to implement the promotion. You can either promote all T0, T1, and T2 in one command, or promote T0 and T1 before T2. Note that you cannot promote T2 without promoting T0 and T1. If you only promote T2, an error occurs. In this case, you need to promote it separately. Since T0 is already in the `test` branch, you only need to promote T1 before promoting T2.

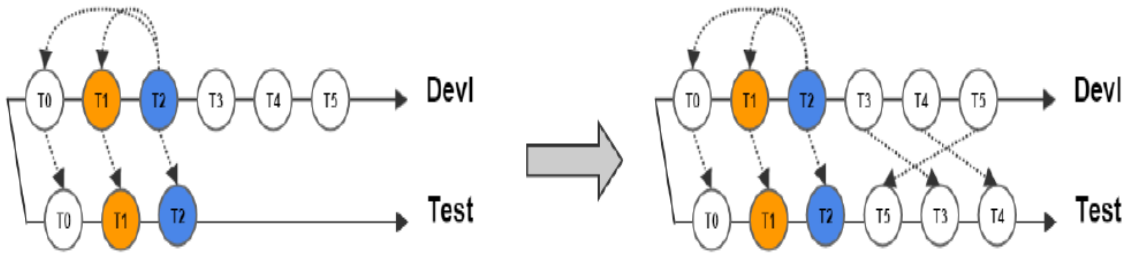
Fig. 2.4
Promoting Dependent Ticket T2



After promoting T1 into the `test` branch, you can promote T2 to the `test` branch because all dependent tickets (T0 and T1) have been already promoted to the `test` branch.

Example **Promoting independent tickets**

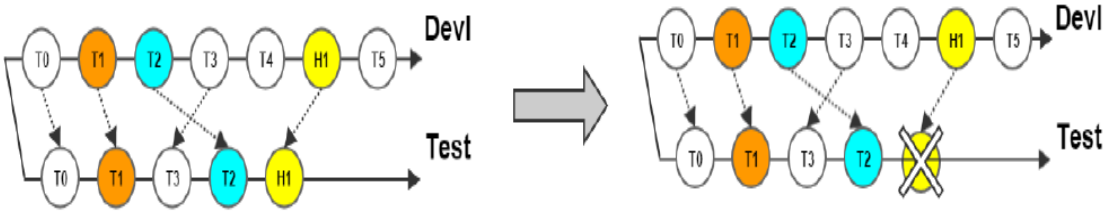
Fig. 2.5
Promoting Independent Tickets T3, T4, and T5



When promoting independent tickets, T3, T4, and T5, you could first promote T5, then T3, and finally T4 to the `test` branch. The sequence in the `test` branch is T5, T3, and T4 as the diagram above shows.

Example **Handling errors in promotion**

Fig. 2.6
Handling Errors When Promoting Ticket H1



If a promotion fails, CVC rolls back to exactly what the environment was before the promotion execution. As shown in Figure 2.6, the user promotes H1 from `devl` to `test`, but receives an error message. The system invokes the rollback function and removes all related data generated from this promotion in the `test` environment.

Example

To list all the tickets that could be promoted in the `test` environment, run:

```
$ cvc promote --available --env test
```

To simulate the process of promoting ticket CVC-101 to the `test` environment, run:

```
$ cvc promote CVC-101 --env test --simulate
```

To promote ticket CVC-101 from the `devl` environment to `test`, run:

```
$ cvc promote CVC-101 --env test
```

To promote ticket CVC-101 from the `test` environment to `prod`, run:

```
$ cvc promote CVC-101 --env prod
```

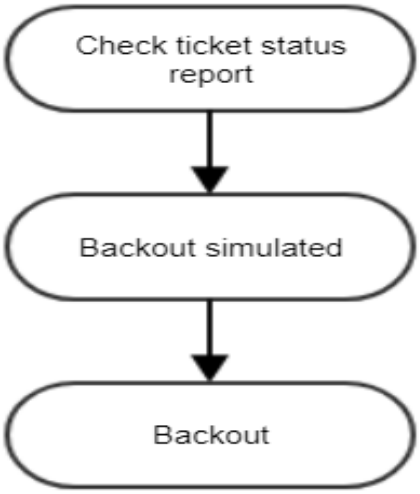
Ticket Backout Process

This section explains the basic process of ticket backout and also discusses some special cases and errors that users may encounter when using the `cvc backout` command.

Process Diagram

Figure 2.7 illustrates the ticket backout process.

Fig. 2.7
Backout Process Diagram



Process Description

A typical ticket backout involves the following operations:

- Obtaining the ticket status report

  Users can run the `cvc report --ticket` command to obtain ticket commit information.

  In the target environment, users can only back out the tickets that have not been promoted to the next environment.

- Performing a simulated backout

  Users can specify the `--simulate` option to perform a simulated backout.

  The `cvc backout` command requires a list of tickets and a target environment, and uses tickets as backout units.

  The `cvc backout` command checks the environment settings in the database. Users need to define the target environment.

  Users can specify multiple tickets in one `cvc backout` command to back out the tickets together.

- Performing a backout

  QAD recommends that users run a simulated backout before the real one. In this way, users can check the file change list through the simulation first.

  The `cvc backout` command reverts the ticket related changes in the target environment.

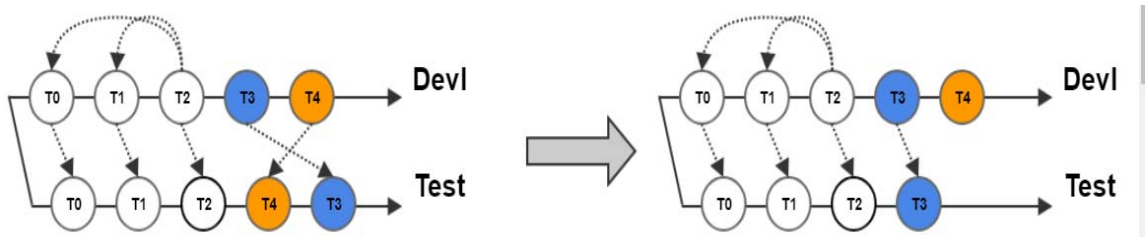
  The `cvc backout` command verifies ticket dependencies before conducting backout.

  If the target environment is the first environment (usually named as `devl`), the environment is treated as a development environment. If users perform backout in a development environment, CVC will back up those file changes in their local directory. As the changes in the development environment have not been promoted to any environment yet, users are not able to get back the changes easily after backout.

The normal backout process is completed if no error occurs. The following discusses some special cases and errors in backout.

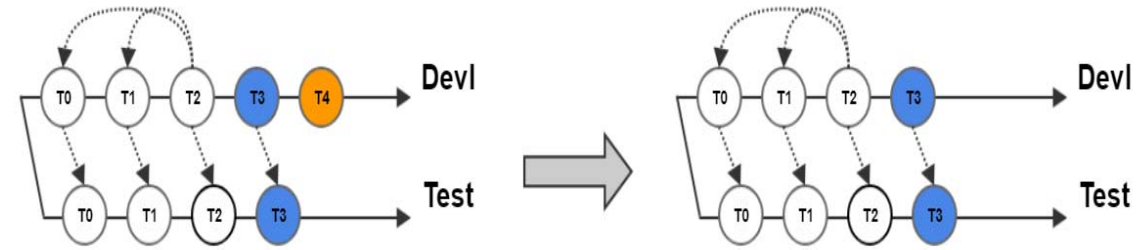
Example **Backing out independent tickets**

Fig. 2.8
Backing Out Independent Ticket T4 in test



Assume that you have promoted a ticket named T4 from the `devl` environment to the `test` environment and it has no dependency with other tickets. If you find T4 useless and want to remove it from `test`, you should run the `cvc backout` command in the `test` environment.

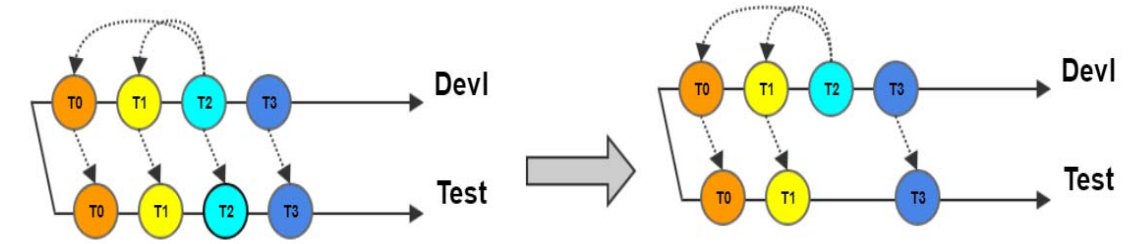
Fig. 2.9
Backing Out Independent Ticket T4 in devl



If you want to remove T4 from the `devl` environment, specify the `devl` environment. Then T4 is removed completely from `devl`.

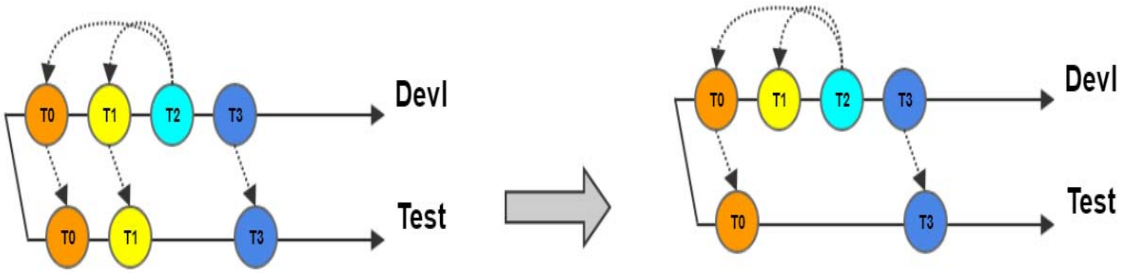
Example **Backing out dependent tickets**

Fig. 2.10
Backing Out Dependent Ticket T2 in test



Assume that you have a ticket named T2 and it has dependencies with two tickets, T1 and T0. T1 and T0 are independent to each other. If you want to back out T1, you have to back out T2 first.

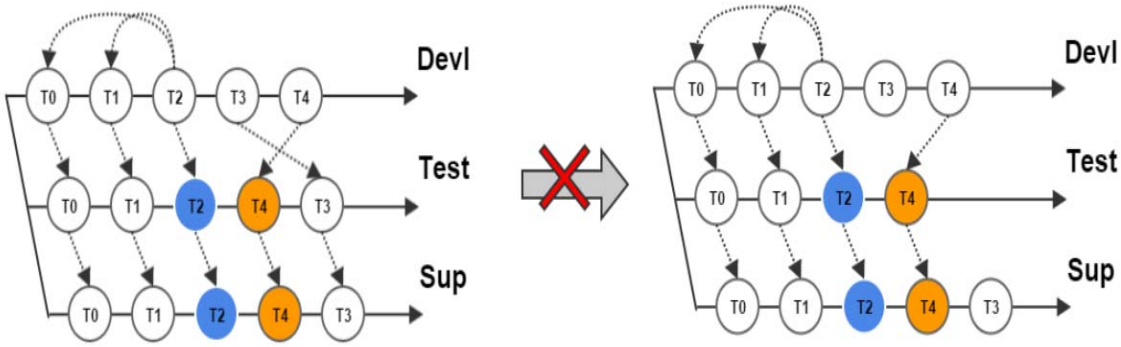
Fig. 2.11
Backing Out Dependent Ticket T1 in test



After backing out T2 from `test`, now you are able to back out T1 from `test`.

Example **Handling errors in ticket backout**

Fig. 2.12
Handling Errors When Backing Out T3 in test



Assume that you want to back out a ticket named T3 from `test`. The ticket has already promoted to `prod`, which means it is available in all three environments. It is illegal to back out a ticket when it still exists in a higher environment. In this case, you could not back out T3 before removing it from `prod`. In other words, you can back out a ticket in the current environment only after you have backed it out in all higher environments.

Example

To list all ticket commits in all environments, run:

```
$ cvc report --ticket CVC-101
```

To simulate the process of backing out ticket CVC-101 in the `test` environment, run:

```
$ cvc backout CVC-101 --env test –-simulate
```

To back out ticket CVC-101 in the `test` environment, run:

```
$ cvc backout CVC-101 --env test
```

To back out ticket CVC-101 in the `devl` environment, run:

```
$ cvc backout CVC-101 --env devl
```

Report Process

This section introduces some report features of QAD CVC.

CVC supports the following reports:

- Activity log report

- File detail report
- Module detail report
- Environment commit report
- Ticket commit report
- Overlap report
- File history report
- Promote available tickets report
- Environment difference report
- Recent file report
- Online user report
- JIRA ticket status report
- File mapping report

Users can run the `cvc report --help` command to obtain all the options in the help message.

Users can specify the `--report-type` option to obtain reports in different formats, such as default (tabular), csv, xml, json or raw report.

Users can specify the `--mail-to` and `--mail-subject` options to send a report file to a specified email address.

Example

To view command usage, run:
```
$ cvc report --help
```

To list activity logs, run:
```
$ cvc report --activity-log
```

To show the details of files ending with `somt.p`, run:
```
$ cvc report --file *somt.p
```

To show the details about module `mfgpro_cust`, run:
```
$ cvc report --module mfgpro_cust
```

To list the environment commits in the `devl` environment, run:
```
$ cvc report --env devl
```

To list the commits of ticket CVC-101, run:
```
$ cvc report --ticket CVC-101
```

To list the file history of file `us/xx/sosorp.p` during the last 1000 days, run:
```
$ cvc report –-file-history us/so/sosorp.p --days 1000
```

To list the tickets available for promotion, run:
```
$ cvc report --promote-avail --env test
```

To list the different commits in the environments `devl` and `test`, run:
```
$ cvc report --env-diff devl test
```

To list the recent file changes during the last 100 days, run:
```
$ cvc report --recent-files --days 100
```

To list the online users, run:
```
$ cvc report --online-users
```

To list the file overlaps in the `devl` environment, run:

```
$ cvc report --overlap --env devl
```

To list the file mappings of files named `us/xx/xxlsomt.p`, run:

```
$ cvc report --file-mappings --file "us/xx/xx1somt.p"
```

To list the JIRA ticket status of ticket CVC-101, run:

```
$ cvc report --jira-status CVC-101
```

Overlap Detecting Process

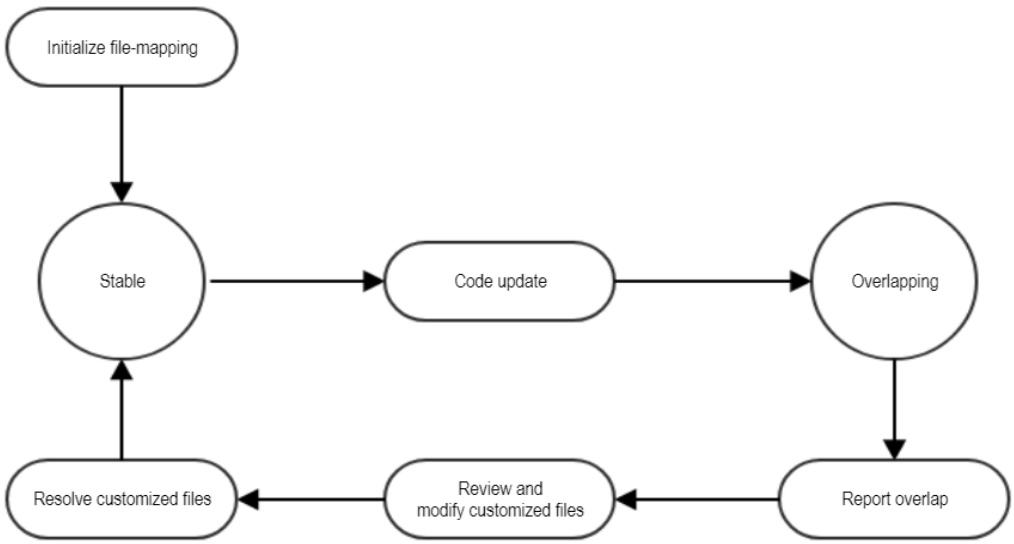
This section describes the rules for identifying file overlaps and how to use CVC to detect overlaps and perform impact analysis.

The most important requirement for the overlap detecting process is to check whether a product upgrade or patch installation would impact existing customization programs. The situation happens when a customer wants to upgrade some existing products to a new version or to install a new patch for the existing product. If the customer already has the product customized in the environment, developers need to verify that the customization is compatible with the new product version. To help the developers figure out all the custom files that need to be reviewed and revised, CVC provides the overlap detecting feature to generate a report of custom files affected by the patch during the product upgrade. CVC requires the mappings between standard source files and custom files before it can find all the custom files related to those updated standard files. CVC also needs the mappings between include files and custom files so that it can report the affected files if any include files are upgraded. In order to keep such mapping information in CVC, file-mapping is introduced. For details about how to manage file mappings in CVC, please refer to “File Mapping Management Process” on page 24.

Process Diagram

Figure 2.13 illustrates the overlap detecting process.

Fig. 2.13
Overlap Detecting Process Diagram



Process Description

If custom files map to standard source files precisely, you can use the mappings to report exact overlap information. When the effective standard file is changed (that is, sha1sum is different), the CVC overlap report lists custom files—those are mapped to the modified source files. You need to review and revise those overlap custom files based on the version of the new effective standard source file.

If standard `.i` files are changed during product upgrade, all custom `.p` files including these `.i` files are listed in the file overlap report as well.

A typical overlap detecting process consists of the following procedures:

1 Set up the initial file mappings for the existing custom files.

Administrator users need to use the `cvc file-mapping --initialize` command to scan all the version-controlled custom files and try to identify the possible source file path based on the filename. The detailed rules are described in “File Mapping Management Process” on page 24. After setting up the initial data, administrator users can review and update the source file mapping data manually. In addition to source file mappings, administrator users need to use the `cvc analyze-rcode` command to initialize the file mappings of the `include` type so that you can obtain the relationships between custom procedure files and standard include files. You also need to generate file mappings of the `included_in` type to obtain the relationships between the custom include files and standard procedure files.

After you set up the file mappings, all file mappings in the system are classified into three types:

Custom File	File Mapping Type	Source File	Description
Custom file	source	Standard file	The custom file is sourced from the standard file. Or the standard file is detected as the potential source for the custom file based on the naming convention and compile propath. The file mapping may be from the CVC file-mapping initialization process or be manually specified by users.
Custom procedure file	include	Standard include file	The custom procedure file includes the standard include file. The file mapping may be from the CVC analyze-rcode process or be generated from checkin or registration.
Custom include file	included_in	Standard procedure file	The custom include file is included in the standard procedure file, which means the standard procedure file includes the custom include file. The file mapping may be from the CVC analyze-rcode process or be generated from checkin or registration. Generally the custom include file has the same name of a standard include file that is included in the standard procedure file. You need to modify the `compile.lst` file to include the standard procedure file into the compilation process and generate the relationships between standard procedure files and custom include files.

2 When adding a new custom file for version control, the `cvc add` command tries to find the potential source of the new file and allows you to select the correct source for creating the file mapping relationship.

If the custom file is sourced from a standard file, developers can also use the `cvc source` command to select and copy the corresponding standard source file to the customization folder. CVC records the source and custom file relationship in the database. The source file sha1sum should be saved. CVC also encrypts the source file if it is unencrypted and saves the sha1sum of the encrypted version. That is because you need to consider that some files only have the encrypted version in the environment and you need to treat the source file identically if new unencrypted source takes effect and the sha1sum of both files is the same after encryption.

3 The customer upgrades the standard product in the normal process.

The source files with the same names may be added to the module's propath so that they can take effect and replace the old files.

4 When the customer upgrades the standard product, developers need to run the `cvc report --overlap` command to check file overlap.

CVC first scans all the version-controlled custom files and searches for their file mappings in the database. Then CVC goes through each file mapping record. For each mapped source file, CVC finds the first effective standard source in the propath and checks if their file checksum is different from the record in the database. If the effective source file is changed, the corresponding custom file is listed in the overlap report for review. All the custom files mapping to those modified source files are considered as file overlap and are listed in the file overlap report.

Updated Source File Flag	Description
R origin_source_path -> new_source_path	The valid source path found in the propath is not equal to the original source path. Developers need to review the overlap files based on the source file content of `new_source_path`.
M source_path	The valid source path is equal to the original source path, but the source file content is changed. Developers need to review the overlap files based on the source file content of `source_path`.
D source_path	The original source file is deleted and CVC cannot find any valid source file with the source filename in the propath. Developers need to review the overlap files and determine whether it is obsolete and can be deleted.

5 Developers need to review those overlap custom files manually.

They can use the `cvc checkin` command to commit the revised custom files if needed.

6 After revising the custom files, developers need to use the `cvc resolve` command to mark the file overlap as resolved and map the new custom files to the new source files.

The custom file mapping records in the database are updated with the new source file sha1sum. Next time developers run the `cvc report --overlap` command, those resolved overlaps are not listed in the report any longer.

Example

- File mapping initialization

  Administrator users need to set up file mappings for all custom files using the `cvc file-mapping --initialize` and `cvc analyze-rcode` commands. These commands could also be used to update file mappings of specific files or all files from a specific module.

```
$ cvc file-mapping --initialize
Initializing file mappings.....[ok]
These file mappings are initialized successfully:
--- ------- ---------------------------------- ------ ---------------- ---------------------------- ------------------- ----------------------------------------
SEQ ID      File Path                          Type   Source File      Source Path                  Last Modified       Original Checksum for Encrypted Source
--- ------- ---------------------------------- ------ ---------------- ---------------------------- ------------------- ----------------------------------------
001 1000178 mfgpro/xxsrc/src/us/so/so05a01.i   source us/so/so05a01.i  mfgpro/xrc/us/so/so05a01.i  2016-08-06 14:03:48 0983edefe3d5ade58952d0abff480ebfafa4363c
002 1000040 mfgpro/xxsrc/src/us/so/so05a02.i   source us/so/so05a02.i  mfgpro/xrc/us/so/so05a02.i  2016-08-06 14:03:48 9519f9b7753aa587bccc6fa95b3abc27dc1dad4b
003 1000398 mfgpro/xxsrc/src/us/so/so05b01.i   source us/so/so05b01.i  mfgpro/xrc/us/so/so05b01.i  2016-08-06 14:03:48 b474240e74021ed87f7075286e7b2e53ed71d987
004 1000322 mfgpro/xxsrc/src/us/so/so05c01.i   source us/so/so05c01.i  mfgpro/xrc/us/so/so05c01.i  2016-08-06 14:03:48 001dc6ec2fedd7b802ad80721fc41c846f9104e2
......
```

```
$ cvc analyze-rcode
Analyzing rcode.....[ok]
These file mappings are updated successfully:
Preview first [500] lines of total [16125] lines:
----- ------- ---------------------- ------- ---------------------- ---------------------------- ------------------- ----------------------------------------
SEQ   ID      File Path              Type    Source File            Source Path                  Last Modified       Original Checksum for Encrypted Source
----- ------- ---------------------- ------- ---------------------- ---------------------------- ------------------- ----------------------------------------
00001 1003873 mfgpro/xrc/mf1.p       include us/bbi/eudeclre.i      mfgpro/xrc/us/bbi/eudeclre.i 2016-08-06 14:03:46 d21dc62c6e0c833391607c49a19c3de8dbbfe022
00002 1003875 mfgpro/xrc/mf1.p       include us/bbi/gpreturn.i      mfgpro/xrc/us/bbi/gpreturn.i 2016-08-06 14:03:46 b7eeeae1803dde1c25f338b9ed9b7e1bd855feb9
00003 1003860 mfgpro/xrc/mf1.p       include us/bbi/gprun.i         mfgpro/xrc/us/bbi/gprun.i    2016-08-06 14:03:46 3f92148e0e56ee486602f182ca1d13f80513e371
00004 1003865 mfgpro/xrc/mf1.p       include us/bbi/gprun1.i        mfgpro/xrc/us/bbi/gprun1.i   2016-08-06 14:03:46 a2fd656077a6b216478f16e10b5e47d47690a75b
00005 1003866 mfgpro/xrc/mf1.p       include us/bbi/icmf.i          mfgpro/xrc/us/bbi/icmf.i     2016-08-06 14:03:46 68f36d0d378139e95bd676b46485868caf1a6982
......
```

- Check the custom file state (**Stable** or **Overlapping**) using the `cvc report --overlap` command

```
$ cvc report --overlap
Generating report.....
 - Detecting files impacted by code update.....[ok]
.....[ok]
No impacted files found.
```

After code update, the state changes from **Stable** to **Overlapping**.

```
$ cvc report --overlap
Generating report.....................................................................
 - Detecting files impacted by code update.....................................[ok]
.....................................................................................[ok]
These are the impacted files details:
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
SEQ File       File Path                      Type   Updated Source Files                 Original Checksum for Encrypted Source
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
001 us/xx/xxsosorp.p mfgpro/xxsrc/src/us/xx/xxsosorp.p source M /devl/apps/mfgpro/xrc/us/so/sosorp.p 84b2d6a14b5756695436e7f584f2650f69437e03
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
You can check the whole report in [overlap_20190128171103704.crpt]
```

- (Optional) Review and modify custom files when necessary

  Release managers or developers can review the custom files listed by the `cvc report --overlap` command to make sure that they are compatible with the update of standard code.
- Resolve file overlaps using the `cvc resolve` command.

```
$ cvc resolve us/xx/xxsosorp.p
Detecting files impacted by code update.....................................[ok]
These are the impacted files details:
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
SEQ File       File Path                      Type   Updated Source Files                 Original Checksum for Encrypted Source
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
001 us/xx/xxsosorp.p mfgpro/xxsrc/src/us/xx/xxsosorp.p source M /devl/apps/mfgpro/xrc/us/so/sosorp.p 84b2d6a14b5756695436e7f584f2650f69437e03
--- ---------- ------------------------------ ------ ------------------------------------ ----------------------------------------
Do you want to resolve the overlaps?
  [Y]es/[N]o:y
Before resolving the overlaps, please make sure you have compiled the files using CVC successfully or run [cvc analyze-rcode] command after manual compilation.
Do you want to continue?
  [Y]es/[N]o:y
Updating file mappings.....................................................[ok]
These file mappings are updated:
--- ------- ------------------------------ ------ ------------- ------------------------ ------------------- ----------------------------------------
SEQ ID      File Path                      Type   Source File   Source Path              Last Modified       Original Checksum for Encrypted Source
--- ------- ------------------------------ ------ ------------- ------------------------ ------------------- ----------------------------------------
001 1000006 mfgpro/xxsrc/src/us/xx/xxsosorp.p source us/so/sosorp.p mfgpro/xrc/us/so/sosorp.p 2019-01-28 17:10:12 8d407fd4cc402768329cd3445331631e425993bf
--- ------- ------------------------------ ------ ------------- ------------------------ ------------------- ----------------------------------------
```

File Mapping Management Process

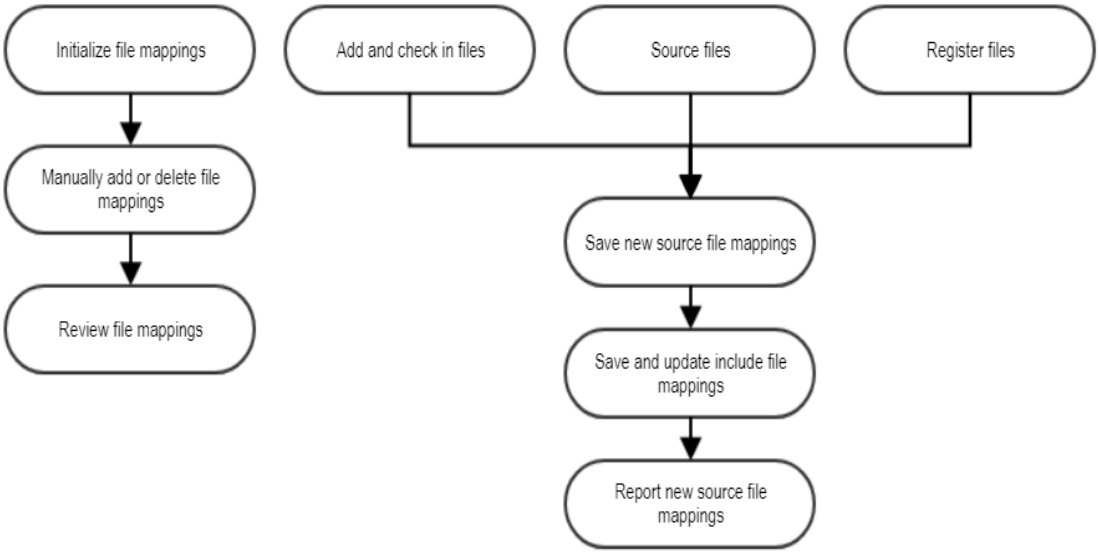
This section describes the rules for detecting potential sources and managing different types of file mappings.

In order to help developers figure out all the custom files that need to be reviewed and revised, CVC provides the overlap detecting feature to generate the report of custom files affected by the patch during the product upgrade. In order to do that, CVC needs to obtain the mappings between standard source files and custom files before it can find all the custom files related to those updated standard files. CVC also requires the mappings between include files and custom files so that it can report the affected files if any include files are upgraded.

Process Diagram

Figure 2.14 illustrates the file mapping management process.

Fig. 2.14
File Mapping Management Process Diagram



Process Description

What Is File Mapping?

A file mapping indicates a mapping relationship between a custom file and a standard file or another custom file.

There are three types of file mappings:

- **Source file mapping** maps a custom file to its corresponding standard source file.

  Note that a standard file may be customized as many custom files. This indicates a source file mapping is a multi-to-one mapping.
- **Include file mapping** maps a custom procedure (`*.p`) file to its include (`*.i`) files.

  A custom procedure file could include many standard files, and a standard include file could be included by many custom procedure files, so that a include file mapping is a multi-to-multi mapping.
- **Included_in file mapping** maps a custom include file to the standard procedure file that includes it.

  This is also a multi-to-multi mapping as a custom include file could be included in multiple standard procedure files, at the same time a standard procedure file may include more than one custom include file.

A file mapping record mainly contains the following information:

- Custom file path
- File mapping type

- Source file name and path
- Last modified time and checksum

Example

Below is an example of file mapping:

```
--- ------- ------------------------- ------- ----------- ------------------------- ------------------ ----------------------------------------
SEQ ID      File Path                 Type    Source File Source Path               Last Modified      Original Checksum for Encrypted Source
--- ------- ------------------------- ------- ----------- ------------------------- ------------------ ----------------------------------------
001 1000019 mfgpro/xxsrc/src/include.p include include.i   mfgpro/xxsrc/src/include.i 2019-01-15 14:36:37 5a8ca84c7d4d9b055f05c55b1f707f223979d387
002 1000014 mfgpro/xxsrc/src/source.p  source  source.p    mfgpro/src/source.p       2019-01-07 11:08:45 657459e82bfa13206677d59bca010ecd39d78e2a
--- ------- ------------------------- ------- ----------- ------------------------- ------------------ ----------------------------------------
```

How to Generate File Mappings?

This section describes the rules for generating different types of file mappings.

Source File Mapping

The following explains how to generate the source file mappings for all existing custom files based on the propath and file naming conventions.

Custom file naming conventions include:

- If the standard code is used by several procedures or includes within QAD Enterprise Applications, and developers do not want all of the procedures and includes to use the customized version of the code, they must give the customized code a filename, which is different from the file name of the standard QAD Enterprise Applications code that they have modified. This way the standard code remains with its original file name.

  Custom source file names = custom prefix (xx) + module + function

Note The custom source file name must have the same last 4 characters with its source file name. In addition, if the custom source file name consists of 8 or more characters, at least 50% of the characters in its source file name must be included in the custom source file name.

  Examples:

  `sosorp.p --> xxsorp.p / xxsosorp.p`

  `sosomt.p --> xxsomt.p / xxsosomt.p`

- If developers have customized standard code and want all standard procedures to use the customized code, they do not need to rename the standard file. QAD recommends that developers save a copy of the original code in its original state in the customization folder for future reference.

If CVC finds multiple standard source files with different names matching the naming convention for one custom file, it needs to save multiple records to the database. However, for each unique standard file name, only the first file in the propath can be saved.

For example, for the custom file `xxsorp.p`, CVC finds the following files in the database propath that matches the overlap detecting rules:

- `folder_A/us/ab/absorp.p`
- `folder_A/us/cd/cdsorp.p`
- `folder_B/us/ab/absorp.p`

Note that the first and third files have the same file name `us/ab/absorp.p`.

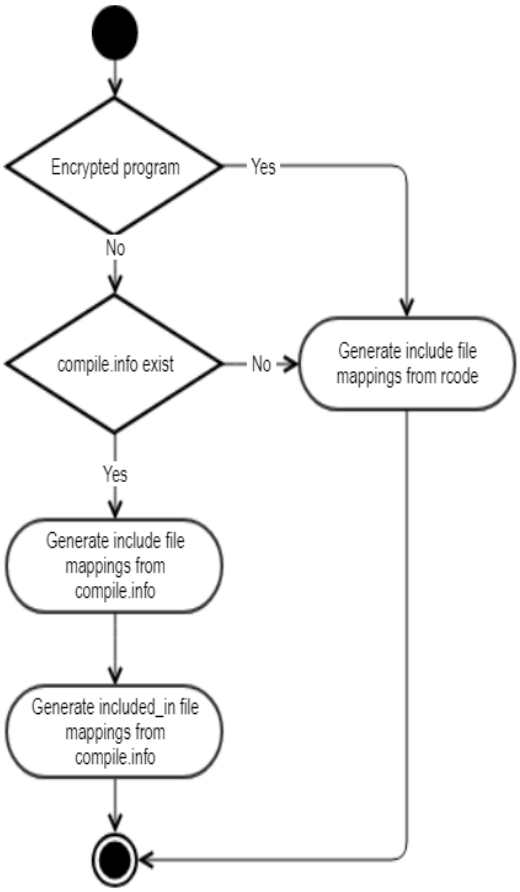
CVC should only save the first and second files as the source file mappings.

Include and Included_in File Mappings by Program Reference Analysis

File mappings of `include` and `included_in` types are generated through program reference analysis. The reference include file list of a program file can be stored in the `rcode` or `compile.info` file:

- The `compile.info` file is generated after compilation. It includes the reference list for all compiled files of the module. For encrypted files, however, the reference list is empty.
- Both encrypted program files and unencrypted files can read the reference list from the rcode string results. However, such information may be inaccurate for `.cls` files.
- The program reference should be parsed from the `compile.info` file first, since this file is more efficient and reliable.

Based on these premises, `include` and `included_in` file mappings are generated according to the process below:



CVC Commands Related to File Mappings

The following CVC commands are related to file mappings:

- cvc add

The `cvc add` command detects potential file sources for newly added files and allows users to select and map each new file to its correct source files.

- cvc checkin

The `cvc checkin` command creates new source file mappings for newly added files and updates file mappings of `include` and `included_in` types.

- cvc source

The `cvc source` command creates source file mappings for new source files.

- cvc register

The `cvc register` command creates or updates the file mappings for the registered custom files.

- cvc file-mapping

The `cvc file-mapping` command provides a direct approach for manual management of source file mappings, allowing administrator users and release managers to initialize, add, delete and show file mappings in the system.

- cvc analyze-rcode

The `cvc analyze-rcode` command performs program reference analysis and generates the file mappings of `include` and `included_in` types.

- cvc report

The `cvc report` command provides the file mapping report with filters. It also provides the overlap report that lists the custom files affected by the standard product code change. The overlap report is generated by comparing the current standard files status with the file mappings data in the database.

Example

- Initialize file mappings

  Administrator users need to set up file mappings for all custom files using the `cvc file-mapping --initialize` and `cvc analyze-rcode` commands. These two commands could also be used to update file mappings of specific files or all files from a specific module.

```
$ cvc file-mapping --initialize
Initializing file mappings.....[ok]
These file mappings are initialized successfully:
--- ------- ------------------------------ ------ ---------------- ---------------------------- ------------------ ----------------------------------------
SEQ ID      File Path                      Type   Source File      Source Path                  Last Modified      Original Checksum for Encrypted Source
--- ------- ------------------------------ ------ ---------------- ---------------------------- ------------------ ----------------------------------------
001 1000178 mfgpro/xxsrc/src/us/so/so05a01.i source us/so/so05a01.i mfgpro/xrc/us/so/so05a01.i 2016-08-06 14:03:48 0983edefe3d5ade58952d0abff480ebfafa4363c
002 1000040 mfgpro/xxsrc/src/us/so/so05a02.i source us/so/so05a02.i mfgpro/xrc/us/so/so05a02.i 2016-08-06 14:03:48 9519f9b7753aa587bcccc6fa95b3abc27dc1dad4b
003 1000398 mfgpro/xxsrc/src/us/so/so05b01.i source us/so/so05b01.i mfgpro/xrc/us/so/so05b01.i 2016-08-06 14:03:48 b474240e74021ed87f7075286e7b2e53ed71d987
004 1000322 mfgpro/xxsrc/src/us/so/so05c01.i source us/so/so05c01.i mfgpro/xrc/us/so/so05c01.i 2016-08-06 14:03:48 001dc6ec2fedd7b802ad80721fc41c846f9104e2
.....
```

```
$ cvc analyze-rcode
Analyzing rcode.....[ok]
These file mappings are updated successfully:
Preview first [50] lines of total [7534] lines:
```

SEQ	ID	File Path	Type	Source File	Source Path	Last Modified	Original Checksum for Encrypted Source
00001	1003873	mfgpro/xrc/mf1.p	include	us/bbi/eudeclre.i	mfgpro/xrc/us/bbi/eudeclre.i	2016-08-06 14:03:46	d21dc62c6e0c833391607c49a19c3de8dbbfe022
00002	1003875	mfgpro/xrc/mf1.p	include	us/bbi/gpreturn.i	mfgpro/xrc/us/bbi/gpreturn.i	2016-08-06 14:03:46	b7eeeae1803dde1c25f338b9ed9b7e1bd855feb9
00003	1003860	mfgpro/xrc/mf1.p	include	us/bbi/gprun.i	mfgpro/xrc/us/bbi/gprun.i	2016-08-06 14:03:46	3f92148e0e56ee486602f182ca1d13f80513e371
00004	1003865	mfgpro/xrc/mf1.p	include	us/bbi/gprun1.i	mfgpro/xrc/us/bbi/gprun1.i	2016-08-06 14:03:46	a2fd656077a6b216478f16e10b5e47d47690a75b
00005	1003866	mfgpro/xrc/mf1.p	include	us/bbi/icmf.i	mfgpro/xrc/us/bbi/icmf.i	2016-08-06 14:03:46	68f36d0d378139e95bd676b46485868caf1a6982

.....

- Review file mappings

```
$ cvc report --file-mappings
Generating report.....[ok]
Preview first [50] lines of total [3012] lines:
```

SEQ	ID	File Path	Type	Source File	Source Path	Last Modified	Original Checksum for Encrypted Source
0001	1006835	mfgpro/xrc/mf1.p	include	us/bbi/eudeclre.i	mfgpro/xrc/us/bbi/eudeclre.i	2016-08-06 14:03:46	d21dc62c6e0c833391607c49a19c3de8dbbfe022
0002	1006837	mfgpro/xrc/mf1.p	include	us/bbi/gpreturn.i	mfgpro/xrc/us/bbi/gpreturn.i	2016-08-06 14:03:46	b7eeeae1803dde1c25f338b9ed9b7e1bd855feb9
0003	1006822	mfgpro/xrc/mf1.p	include	us/bbi/gprun.i	mfgpro/xrc/us/bbi/gprun.i	2016-08-06 14:03:46	3f92148e0e56ee486602f182ca1d13f80513e371
0004	1006827	mfgpro/xrc/mf1.p	include	us/bbi/gprun1.i	mfgpro/xrc/us/bbi/gprun1.i	2016-08-06 14:03:46	a2fd656077a6b216478f16e10b5e47d47690a75b
0005	1006828	mfgpro/xrc/mf1.p	include	us/bbi/icmf.i	mfgpro/xrc/us/bbi/icmf.i	2016-08-06 14:03:46	68f36d0d378139e95bd676b46485868caf1a6982

.....

```
; cvc file-mapping --show
'hese are the file mappings in the system:
'review first [50] lines of total [0012] lines:
... ....... ..... ....... ..... ..... ..... .....
EQ  ID      File Path                  Type    Source File                  Source Path                  Last Modified       Original Checksum for Encrypted Source
... ....... ..... ....... ..... ..... ..... .....
001 1006835 mfgpro/xrc/mf1.p           include us/bbi/eudeclre.i            mfgpro/xrc/us/bbi/eudeclre.i 2016-08-06 14:03:46 d21dc62c6e0c833391607c49a19c3de8dbbfe022
002 1006837 mfgpro/xrc/mf1.p           include us/bbi/gpreturn.i            mfgpro/xrc/us/bbi/gpreturn.i 2016-08-06 14:03:46 b7eeeae1803dde1c25f338b9ed9b7e1bd855feb9
003 1006822 mfgpro/xrc/mf1.p           include us/bbi/gprun.i               mfgpro/xrc/us/bbi/gprun.i    2016-08-06 14:03:46 3f92148e0e56ee486602f182ca1d13f80513e371
004 1006827 mfgpro/xrc/mf1.p           include us/bbi/gprun1.i              mfgpro/xrc/us/bbi/gprun1.i   2016-08-06 14:03:46 a2fd656077a6b216478f16e10b5e47d47690a75b
005 1006828 mfgpro/xrc/mf1.p           include us/bbi/icmf.i                mfgpro/xrc/us/bbi/icmf.i     2016-08-06 14:03:46 68f36d0d378139e95bd676b46485868caf1a6982
'''''
```

- Add and check in files with file mappings

```
$ cvc add us/xx/xxsosomt.p -t CVC-200 -m mfgpro_cust
Validating ticket.....................................................................[ok]
Validating files......................................................................[ok]
Detecting file potential overlaps.....................................................[ok]
There appears to be some potential sources of the file [us/xx/xxsosomt.p]:
--- ----------------- ----------------------------
SEQ Source File       Source Path
--- ----------------- ----------------------------
001 us/so/sosomt.p    mfgpro/xrc/us/so/sosomt.p
002 us/mg/mgursomt.p mfgpro/xrc/us/mg/mgursomt.p
003 us/rc/rcsomt.p    mfgpro/xrc/us/rc/rcsomt.p
--- ----------------- ----------------------------
Do you want to map the file [us/xx/xxsosomt.p] to its potential source?
  [Y]es/[N]o:y
Please enter the sequence number:1
These files are going to be added:
--- ------------------------------------------------------------------------------
SEQ Path
--- ------------------------------------------------------------------------------
001 /devl/apps/mfgpro/xxsrc/src/us/xx/xxsosomt.p
--- ------------------------------------------------------------------------------
Do you want to add these files?
  [Y]es/[N]o:y
These files are added successfully:
--- ------------------------------------------------------------------------------
SEQ File
--- ------------------------------------------------------------------------------
001 us/xx/xxsosomt.p
--- ------------------------------------------------------------------------------
```

```
You are going to check in these files:
--- --- ----------- ------------ ----------------------------------------------
SEQ CHG Module      Source Format File
--- --- ----------- ------------ ----------------------------------------------
001  A  mfgpro_cust  unencrypted  us/xx/xxsosomt.p
--- --- ----------- ------------ ----------------------------------------------
Do you want to check in these files?
  [Y]es/[N]o:y
You are going to map these files to their source files:
--- ---------------- ------ -------------- -------------------------
SEQ File             Type   Source File    Source Path
--- ---------------- ------ -------------- -------------------------
001 us/xx/xxsosomt.p source us/so/sosomt.p mfgpro/xrc/us/so/sosomt.p
--- ---------------- ------ -------------- -------------------------
Do you want to add these file mappings?
  [Y]es/[N]o:y
Backing up files.....................................................[ok]
Committing files to central repository...............................[ok]
Detecting files impacted by code update..............................[ok]
Compiling............................................................[ok]
Releasing file locks.................................................[ok]
Updating files info in the database..................................[ok]
Uploading changes to central server..................................[ok]
Check in successfully!
Accumulated changes in [devl]:
--- --- ----------- ------------ ----------------------------------------------
SEQ CHG Module      Source Format File
--- --- ----------- ------------ ----------------------------------------------
001  A  mfgpro_cust  unencrypted  us/xx/xxsosomt.p
--- --- ----------- ------------ ----------------------------------------------
You can find the files info in [checkin_file_info_20190225160429924.crpt].
These file mappings are added successfully:
--- ------- ------------------------------- ------ -------------- ------------------------- ------------------- ----------------------------------------
SEQ ID      File Path                       Type   Source File    Source Path               Last Modified       Original Checksum for Encrypted Source
--- ------- ------------------------------- ------ -------------- ------------------------- ------------------- ----------------------------------------
001 1008026 mfgpro/xxsrc/src/us/xx/xxsosomt.p source us/so/sosomt.p mfgpro/xrc/us/so/sosomt.p 2016-08-06 14:03:48 d68be6084227d3125d1506477899c71757495c15
--- ------- ------------------------------- ------ -------------- ------------------------- ------------------- ----------------------------------------
```

XREF Index Analysis Process

CVC is integrated with Developer Self-Service Tools: XREF file parser and partial index checker. When users run CVC commands to add new code changes into version control, CVC compiles the code changes with XREF and then runs the XREF file parser and the partial index checker to check whether there are new whole-index issues in the new change set.

CVC uses a configuration setting, `enforceWholeIndexValidation`, to control whether developers are allowed to commit the files with whole-index issues. When the `enforceWholeIndexValidation` flag is set to `false`, CVC displays a warning message about the files that have whole-index issues. If the `enforceWholeIndexValidation` flag is set to `true`, CVC displays an error message and blocks the files to be committed.

In a JIRA-enabled environment, CVC packages the XREF analysis results into a `.zip` file and uploads it to the corresponding JIRA ticket after the files are checked in successfully. Then users can check the XREF results in the attachment panel of the JIRA ticket.

CVC keeps track of the whole-index status of each file including the information about the ticket and the user who introduces the whole-index issue of the file. The whole index-information can be found in the file report and promote available report.

Commands that Have Enabled XREF Index Analysis

The commands that have enabled XREF index analysis include:

- cvc dev-compile
- cvc rel-compile
- cvc checkin
- cvc register
- cvc revert
- cvc promote
- cvc backout

Output Files

The output files, which are generated by the XREF index analysis tools, are placed in the current ticket folder. Users can check those output files to view the detailed files that have index issues.

The table below explains each output file, where <*file*> refers to the name of the compiled file and <*ticket*> refers to the ticket that is currently in use:

Output File	Description
<file>_TT_WholeIndex_<ticket>	List of all Temp Table procedure files and source files that have index issues.
<file>_DB_WholeIndex_<ticket>	List of all Database procedure files and source files that have index issues.
<file>_DB_WholeIndex_warning_<ticket>	List of all Database procedure files and source files that have index issues flagged as warnings. The script determines any “for first”, “find first”, “for last”, and “find last” code without `where` clauses or queries to `dom_mstr` as warnings.
<file>_DB_WholeIndex_error_<ticket>	List of all Database procedure files and source files that have index issues. These are non-indexed queries that fall out of the warning range. These include all “for each” queries as well as all “for first” and “find first” queries with `where` clauses. **You should review the content in this file as a first priority. In addition, whole-index issues should be adjusted to use the correct index or be added with a comment to explain non-index use.**
<file>_Exemptions_<ticket>	List of all queries that were treated as exemptions based on the criteria provided as an optional input file.
<file>_Partial_Index_Matches_<ticket>	List of all queries that only partially match the fields in the index they use.
<file>_Could_Not_Evaluate_<ticket>	List of all queries that could not be evaluated due to parameters being substituted for database field names.

Command Examples

`cvc dev-compile` with files that have whole-index issues

```
$ cvc dev-compile
Collecting files from current folder.....................................[ok]
Validating files.........................................................[ok]
You are going to dev compile these files:
--- --- ---------- ------------------------------------------------------------
SEQ CHG Module     File
```

```
--- --- ---------- ------------------------------------------------------------
001  A  CUSTOM-MFG us/xx/xxtest.p
--- --- ---------- ------------------------------------------------------------
Do you want to dev compile these files?
  [Y]es/[N]o:y
Backing up files.....................................................[ok]
Updating work directory..............................................[ok]
Compiling............................................................[ok]
Running xref index analysis tools....................................[ok]
These files are compiled successfully:
--- --- ---------- ------------------------------------------------------------
SEQ CHG Module     File
--- --- ---------- ------------------------------------------------------------
001  A  CUSTOM-MFG us/xx/xxtest.p
--- --- ---------- ------------------------------------------------------------
WARNING: These files have WHOLE-INDEX issues:
--- --------------------------------------------------------------------------
SEQ File
--- --------------------------------------------------------------------------
001 us/xx/xxtest.p
--- --------------------------------------------------------------------------
$ find
.
./cvc.log
./us
./us/xx
./us/xx/xxtest.p_DB_WholeIndex_warning_CVC-500
./us/xx/xxtest.p_DB_WholeIndex_CVC-500
./us/xx/xxtest.p_Could_Not_Evaluate_CVC-500
./us/xx/xxtest.p
./us/xx/xxtest.p_Partial_Index_Matches_CVC-500
./us/xx/xxtest.p_DB_WholeIndex_error_CVC-500
./us/xx/xxtest.p_TT_WholeIndex_CVC-500
./us/xx/xxtest.p_Exemptions_CVC-500
./.cvclist
./.cvc
./.cvc/.info
./.cvc/.meta
./.cvc/.meta/xref
./.cvc/.meta/xref/us
./.cvc/.meta/xref/us/xx
./.cvc/.meta/xref/us/xx/xxtest.p
```

cvc checkin with files that have whole-index issues

```
$ cvc ci -t CVC-500 -n 'test index analysis'
Validating ticket....................................................[ok]
Validating files.....................................................[ok]
Validating file locks................................................[ok]
Validating if files are out of date..................................[ok]
Synchronizing environment............................................[ok]
You are going to check in these files:
--- --- ---------- ------------------------------------------------------------
SEQ CHG Module     File
--- --- ---------- ------------------------------------------------------------
001  A  CUSTOM-MFG us/xx/xxtest.p
--- --- ---------- ------------------------------------------------------------
Do you want to check in these files?
  [Y]es/[N]o:y
Backing up files.....................................................[ok]
Committing files to central repository...............................[ok]
Compiling............................................................[ok]
Running xref index analysis tools....................................[ok]
Validating if files have whole index issues..........................[ok]
WARNING: These files have WHOLE-INDEX issues:
--- --------------------------------------------------------------------------
SEQ File
--- --------------------------------------------------------------------------
001 us/xx/xxtest.p
--- --------------------------------------------------------------------------
```

```
Do you still want to check in these files?
  [Y]es/[N]o:y
Releasing file locks.....................................................[ok]
Updating files info in the database......................................[ok]
Uploading changes to central server......................................[ok]
Check in successfully!
Accumulated changes in [devl]:
--- --- ---------- ------------------------------------------------------------
SEQ CHG Module     File
--- --- ---------- ------------------------------------------------------------
001  A  CUSTOM-MFG us/xx/xxtest.p
--- --- ---------- ------------------------------------------------------------
You can find the files info in [checkin_file_info_20201022010248118.crpt].
```

cvc report --file with whole-index information

```
$ cvc report --file
Synchronizing environment................................................[ok]
Generating report.......................................................
 - Getting file information.............................................[ok]
.......................................................................[ok]
These are the details of the files:
--- ---- ---------- ---------------------- ------------------------------------------
------- ------------------ --------------- ---------------
SEQ Env  Module    File                Path                                      Last
Modified       Source File      Whole Index
--- ---- ---------- ---------------------- ------------------------------------------
------- ------------------ --------------- ---------------
001 devl CUSTOM-MFG us/xx/xxtest.p
systest/customizations/mfg/default/src/us/xx/xxtest.p 2020-10-22 01:02:31             yes
CVC-500 MFG
--- ---- ---------- ---------------------- ------------------------------------------
------- ------------------ --------------- ---------------
You can check the whole report in [files_20201022010316594.crpt]
```

cvc report --promote-avail with whole-index information

```
$ cvc report --promote-avail -e test
Synchronizing environment................................................[ok]
Generating report.......................................................
 - Getting promotion available list.....................................[ok]
.......................................................................[ok]
These tickets are available to be promoted:
(The files having whole index issues are marked with * in the file details column).
--- ------- ------------------ ---------- ------ ------------------- ---------- --------
-- ------------------------------------------------------
SEQ Ticket  Commit message     Depends On Author Last Updated       Whole Index File count
File details
--- ------- ------------------ ---------- ------ ------------------- ---------- --------
-- ------------------------------------------------------
001 CVC-500 test index analysis           MFG    2020-10-22 01:02:31 1 yes              1
*A systest/customizations/mfg/default/src/us/xx/xxtest.p
--- ------- ------------------ ---------- ------ ------------------- ---------- --------
-- ------------------------------------------------------
You can check the whole report in [promotion_available_list_20201022010610209.crpt]
```

Chapter 3

Command Manual

This chapter describes the usage of each CVC command.

cvc add

Description

The `cvc add` command allows developers to add files into custom modules before check-in. Then users can run the `cvc checkin` command to check in local changes. If a file already exists in the system, CVC prevents users from adding the file and asks them to check it out first.

Usage

Syntax

```
cvc add <files>... -t <arg> [-m <arg>] [-ns] [-s] [-c] [-l <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Module name.
`-ns,--non-src`	Add new files to the module's root directory instead of src directory.
`-s,--status`	Show status of local file addition.
`-c,--cancel`	Cancel the file addition.
`-l,--list-file <arg>`	List containing the files to be added.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To add the file `us/xx/xxsample1.p` to the custom module `mfgpro_cust`, run:

```
$ cvc add --ticket CVC-100 --module mfgpro_cust us/xx/xxsample1.p
```

To add the files specified in the `file.lst` file to the custom module `mfgpro_cust` using the `--list-file` option, run:

```
$ cvc add --ticket CVC-100 --module mfgpro_cust --list-file file.lst
```

The following is an example of the content in the `file.lst` file:

```
us/xx/xxsample1.p
us/xx/xxsample2.p
```

cvc analyze-rcode

Description

The `cvc analyze-rcode` command allows release managers to update program references by analyzing rcodes of specified files.

This command can also find and analyze all files from a specified module or environment.

Usage

Syntax

```
cvc analyze-rcode <files>... [-m <arg>] [-l <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Module name.
`-l,--list-file <arg>`	List containing the files to be analyzed.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To analyze rcode files of module `mfgpro_cust` in the `devl` environment, run:

```
$ cvc analyze-rcode --module mfgpro_cust --env devl
```

cvc backout

Description

The `cvc backout` command allows release managers to revert ticket changes in the target environment, especially from the previous promotion. This command uses tickets as backout units.

After reverting the changes in the target environment, the `cvc backout` command compiles all program files related to the tickets and then deploys the rcode files generated to the module's rcode directory.

Users need to make sure that all tickets to be backed out are backed out in the high-level environment first if they have been promoted. For example, there are four environments: `devl`, `test`, `sup`, and `prod`. `devl` is the first environment that developers make changes in, and `prod` is the last one. When you back out one ticket in the `devl` environment, you receive an error message if the ticket is already promoted to the `test` environment. You need to back out the ticket in the `test` environment first. You also need to make sure that all the dependencies related to the tickets have already been backed out, or you need to back them out together.

QAD recommends that you run the `cvc backout --simulate` command to obtain the accumulated file change list before you perform an actual backout.

In addition, if you run the `cvc backout` command in the first environment like `devl`, you obtain a backup copy of all the file changes saved in your current folder.

Usage

Syntax

```
cvc backout <tickets>... [-s] [-l <arg>] [-nc] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-s,--simulate`	Perform a dry run but not perform the actual backout.
`-l,--list-file <arg>`	List containing the tickets to be backed out.
`-nc,--no-compile`	Not compile after backout.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To back out the ticket `CVC-100` in the `test` environment without compiling, run:

```
$ cvc backout --env test --no-compile CVC-100
```

To back out the tickets in the `ticket.lst` file using the `--list-file` option in the `prod` environment with compiling, run:

```
$ cvc backout --env prod --list-file ticket.lst
```

The following is an example of the content in the `ticket.lst` file:

```
CVC-101
CVC-102
```

cvc check-consistency

Description

The `cvc check-consistency` command is available only in QDT and SE environments.

This command uses xcode to encrypt the module's src files and then compares the encrypted files' checksum with their corresponding xrc files'.

Usage

Syntax

```
cvc check-consistency -m <arg> [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Module name.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To check code consistency of the module `mfgpro_cust` in the `devl` environment, run:

```
$ cvc check-consistency --module mfgpro_cust --env devl
```

cvc check-env

Description

The `cvc check-env (ce)` command allows users to check whether the specified environment is clean and reports the change list if any version-controlled files were changed manually instead of using `cvc` commands.

Usage

Syntax

```
cvc check-env [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To check the current environment and make the system report whether the environment is clean (if not clean, report the change list as well), run:

```
$ cvc check-env
```

To check the `devl` environment and make the system report whether `devl` is clean (if not clean, report the change list as well), run:

```
$ cvc check-env --env devl
```

cvc checkin

Description

The `cvc checkin (ci)` command allows developers to check in local changes of customization into the version control system. To perform customization on existing files, you need to use the `cvc checkout` command to check out those files to your ticket folder first. Make sure that you have the file lock of the files you want to check in. If you lose your lock on a file, you can run the `cvc checkout --force` command again to reacquire the lock. To add new files, you need to run the `cvc add` command first to specify the custom module of those new files. To delete files, you can delete the local files in your ticket folder and check in the changes. The `cvc checkin` command triggers the compilation based on the module compile propath and then puts the rcode file in the rcode directory of the module. If there are several files but for different modules, CVC compiles the files separately with different compile propaths and rcode directories.

Usage

Syntax

```
cvc checkin <files>... -n <arg> -t <arg> [-l <arg>] [-nc] [-s] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-n,--note <arg>`	Note to check in files.
`-l,--list-file <arg>`	List containing the files to be checked in.
`-nc,--no-compile`	Not compile after check-in.
`-s,--simulate`	Perform a dry run but not perform the actual check-in.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To check in all the files in the ticket folder `CVC-100`, run:

```
$ cvc checkin --ticket CVC-100 --note "Check in all the files"
```

To check in the file `us/so/sosomt.p` in the ticket folder `CVC-100`, run:

```
$ cvc checkin --ticket CVC-100 --note "Check in us/so/sosomt.p"
us/so/sosomt.p
```

To check in the files specified in the `file.lst` file in the ticket folder `CVC-100` using the `--list-file` option, run:

```
$ cvc checkin --ticket CVC-100 --module mfgpro_cust --list-file file.lst --note "Check in
files in the list"
```

The following is an example of the content in the `file.lst` file:

```
us/xx/xxsample1.p
us/xx/xxsample2.p
```

cvc checkout

Description

The `cvc checkout(co)` command allows developers to check out files from custom modules and copy them to the ticket folder. You can only check out files from custom modules. If a file only exists in standard modules, you need to run the `cvc source` command to source the file first.

The application scenarios of this command are as follows:

- Normal checkout: Check out files to the ticket folder for customization. Once the files have been checked out, they become locked. This stops other developers from checking out the same files. However, the other developers could use the `–force` flag to forcibly check out the files and transfer the lock to themselves.
- Force checkout: Check out files that are locked by other developers. After a force checkout, you can acquire the file lock from other developers.
- Cancel checkout: Cancel the checkout operation and release the file lock. After a checkout is canceled, the specified files are deleted from the ticket folder.

Usage

Syntax

```
cvc checkout <files>... -t <arg> [-l <arg>] [-f] [-c] [-n <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-l,--list-file <arg>`	List containing the files to be checked out.
`-f,--force`	Forcibly check out files despite any lock.
`-c,--cancel`	Cancel the file checkout.
`-n,--note <arg>`	Note to check out files.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To check out the file `us/so/sosomt.p` to the local `CVC-100` folder, run:

```
$ cvc checkout --ticket CVC-100 us/so/sosomt.p
```

To forcibly check out the file `us/so/sosomt.p` to the local `CVC-100` folder, run:

```
$ cvc checkout --ticket CVC-100 --force us/so/sosomt.p
```

To cancel the checkout of the file `us/so/sosomt.p`, run:

```
$ cvc checkout --ticket CVC-100 --cancel us/so/sosomt.p
```

cvc config

Description

The `cvc config` command allows developers to view and set local configurations.

Users can check the JIRA project key and whether JIRA is enabled for the current environment by running the `cvc config -s` command.

Usage

Syntax

```
cvc config <options>... [-s] [-h] [-dh] [-y]
```

Option Explanation

`-s,--show`	Show all configurations.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To show all configurations, run:

```
$ cvc config --show
```

To set the local value of `reportWidth` to `100`, run:

```
$ cvc config reportWidth 100
```

cvc delete

Description

The `cvc delete (del)` command allows developers to delete custom files in the version control system. To delete existing custom files, you need to check out those files to your ticket folder first, and then use the `cvc delete` command to commit the changes to the version control system. The `cvc delete` command triggers compilation of the files based on their module's compile propath and then deploys rcode files. If the files to be deleted belong to different modules, CVC compiles the files separately with their own module's compile propaths.

Usage

Syntax

```
cvc delete <files>... -n <arg> -t <arg> [-l <arg>] [-nc] [-s] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-n,--note <arg>`	Note to delete files.
`-l,--list-file <arg>`	List containing the files to be deleted.
`-nc,--no-compile`	Not compile after deletion.
`-s,--simulate`	Perform a dry run but not perform the actual deletion.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To delete the file `us/so/sosomt.p` in the ticket folder `CVC-100`, run:

```
$ cvc delete --ticket CVC-100 --note "Delete us/so/sosomt.p" us/so/sosomt.p
```

To delete the files specified in the `file.lst` file in the ticket folder `CVC-100` using the `--list-file` option, run:

```
$ cvc delete --ticket CVC-100 --module mfgpro_cust --list-file file.lst --note "Delete files
in the list"
```

The following is an example of the content in the `file.lst` file:

```
us/xx/xxsample1.p
us/xx/xxsample2.p
```

cvc dev-compile

Description

The `cvc dev-compile (dc)` command allows developers to quickly compile programs before check-in. First, the `cvc dev-compile` command detects any file changes under the current ticket folder and copies the files to their respective modules' work directories. Then CVC compiles the files in batches using their respective modules' compile propaths. The `dev-compile` command does not affect program reference database records and deploys rcode to each module's work directory.

Usage

Syntax

```
cvc dev-compile <files>... [-l <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-l,--list-file <arg>`	List containing the files to be compiled.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To compile all programs under the current directory, run:

```
$ cvc dev-compile
```

cvc diff

Description

The `cvc diff` command displays the differences between two revisions. The revision should be specified by the commit SHA. The commit SHA can be found in file history reports through the `cvc report --file-history` command. Users can also find the commit SHA in the environment history reports using the `cvc report –env` command. In addition, users can specify a list of files to display the differences between two revisions of those files. CVC displays the differences on the screen and also keeps them in a report file in the current folder.

Usage

Syntax

```
cvc diff <files>... [-l <arg>] [-or <arg>] [-nr <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-l,--list-file <arg>`	List containing the files to obtain differences.
`-or,--old-revision <arg>`	Commit SHA of the old revision.
`-nr,--new-revision <arg>`	Commit SHA of the new revision.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To display the differences between the old revision `085cab4` and the latest revision of file `us/so/sosorp.p`, run:

```
$ cvc diff us/so/sosorp.p --old-revision 085cab4
```

To display the differences between the old revision `085cab4` and the new revision `a1af793`, run:

```
$ cvc diff --old-revision 085cab4 --new-revision a1af793
```

cvc download

Description

The `cvc download` command allows users to download files of a specific revision to the current folder from the central repository. The revision should be specified by the commit SHA. The commit SHA can be found in file history reports through the `cvc report --file-history` command.

The specified revision should contain changes to the list of files in the change set.

Usage

Syntax

```
cvc download <files>... [-l <arg>] [-r <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-l,--list-file <arg>`	List containing the files to be downloaded.
`-r,--revision <arg>`	Commit SHA of the revision to be downloaded.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To download the file `us/so/sosorp.p` of the revision `085cab4` to the current folder, run:

```
$ cvc download us/so/sosorp.p --revision 085cab4
```

To download the file `us/so/sosorp.p` of the last revision to the current folder, run:

```
$ cvc download us/so/sosorp.p
```

To download all the files of the revision `085cab4` to the current folder, run:

```
cvc download --revision 085cab4
```

To download all the files of the latest revision to the current folder, run:

```
$ cvc download --revision HEAD
```

cvc file-mapping

Description

The `cvc file-mapping` command allows administrator users and release managers to initialize, add, delete, and show file mappings in the system.

File mapping data is mainly used for overlap detecting in CVC. Such data describes the relationships between custom files and their corresponding source files.

Before you start to manage file mappings, you need to first set up the initial data of all existing files using the `initialize` option. The `cvc file-mapping --initialize` command scans all the files in the custom modules and tries to find all potential file mappings based on the filename convention and their modules' compile propath. You can manually add or delete the file mappings after the initialization if there is any missing or incorrect data. You can run the `cvc file-mapping --show` command to review all the file mappings in the system. In addition, you can use the `cvc file-mapping --add` , `cvc file-mapping --delete`, or `cvc file-mapping --clear-all` command to add or delete file mappings according to the actual situation.

You can run this command only in the `devl` environment.

Usage

Syntax

```
cvc file-mapping -i | -a | -d <arg> | -ca | -s [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-i,--initialize`	Initialize file mappings in the system.
`-a,--add`	Add a new file mapping.
`-d,--delete <arg>`	Delete the file mappings by IDs (separated by commas).
`-ca,--clear-all`	Clear all file mappings.
`-s,--show`	Show file mappings in the system.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc file-mapping --initialize`

```
cvc file-mapping -i [-e <arg>] [-h] [-dh] [-y]
```

`-i,--initialize`	Initialize file mappings in the system.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc file-mapping --add`

```
cvc file-mapping -a -f <arg> -sf <arg> -sp <arg> [-e <arg>] [-h] [-dh] [-y]
```

`-a,--add`	Add a new file mapping.
`-f,--file <arg>`	Custom file name.
`-sf,--source-file <arg>`	Source file name.
`-sp,--source-path <arg>`	Source file path (relative to the environment directory).
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc file-mapping --delete`

```
cvc file-mapping -d <arg> [-e <arg>] [-h] [-dh] [-y]
```

`-d,--delete <arg>`	Delete the file mappings by IDs (separated by commas).
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc file-mapping --clear-all`

```
cvc file-mapping -ca [-e <arg>] [-h] [-dh] [-y]
```

`-ca,--clear-all`	Clear all file mappings.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc file-mapping --show`

```
cvc file-mapping -s [-t <arg>] [-f <arg>] [-sf <arg>] [-e <arg>] [-h] [-dh] [-y]
```

`-s,--show`	Show file mappings in the system.
`-t,--type <arg>`	Specify the file mapping type to show file mappings (source, include or included_in).
`-f,--file <arg>`	Specify the custom file name to show file mappings.
`-sf,--source-file <arg>`	Specify the source file name to show file mappings.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To initialize the file mappings of all the custom files of the current environment, run:

```
$ cvc file-mapping --initialize
```

To add a file mapping between the custom file `us/so/sosorp.p` and the source file `/devl/apps/us/so/sosorp.p`, run:

```
$ cvc file-mapping --add --file us/so/sosorp.p --source-file us/so/sosorp.p --source-path
/devl/apps/us/so/sosorp.p
```

To delete the file mapping records whose IDs are 1000001 and 1000002, run:

```
$ cvc file-mapping --delete 1000001,1000002
```

To delete all the file mappings of the current environment, run:

```
$ cvc file-mapping --clear-all
```

To show all the file mappings of the current environment, run:

```
$ cvc file-mapping --show
```

To show the file mapping in the system of which the custom file is `us/so/sosorp.p` and its source file is `us/so/sosorp.p`, run:

```
$ cvc file-mapping --show --file us/so/sosorp.p --source-file us/so/sosorp.p
```

cvc id

Description

The `cvc id` command displays the current user's name, email address, and user roles in the CVC tool.

Usage

Syntax

```
cvc id [-h] [-dh] [-y]
```

Option Explanation

`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To display information about the current user, run:

```
$ cvc id
```

cvc module

Description

The `cvc module` command allows release managers to manage modules (including adding, updating, and deleting modules).

Usage

Syntax

```
cvc module -a | -u | -d | -s [-h] [-dh] [-y]
```

Option Explanation

`-a,--add`	Add a new module.
`-u,--update`	Update an existing module.
`-d,--delete`	Delete an existing module.
`-s,--show`	Show all modules in the system.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc module --add`

- YAB

```
cvc module -a -m <arg> -r <arg> [-work <arg>] [-p <arg>] [-c <arg>] [-bc <arg>] [-ac <arg>]
[-h] [-dh] [-y]
```

`-a,--add`	Add a new module.
`-m,--module <arg>`	Module name.
`-r,--root-dir <arg>`	Root directory to store source code.
`-work,--work-dir <arg>`	Work directory to put work files (relative to the environment directory).
`-p,--yab-process <arg>`	YAB process to compile (code-mfg-update, code-fin-update, and so on).
`-c,--code-instance <arg>`	YAB code instance name (mfg, fin, and so on).
`-bc,--before-compile <arg>`	Scripts to run before compile.
`-ac,--after-compile <arg>`	Scripts to run after compile.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

- QDT or SE

```
cvc module -a -m <arg> -t <arg> -r <arg> [-src <arg>] [-xrc <arg>]
                [-work <arg>] [-rcode <arg>] [-propath <arg>] [-pf <arg>]
                [-l <arg>] [-rl <arg>] [-sl <arg>] [-bc <arg>] [-ac <arg>]
                [-h] [-dh] [-y]
```

`-a,--add`	Add a new module.
`-m,--module <arg>`	Module name.
`-t,--type <arg>`	Module type (standard or custom).

`-r,--root-dir <arg>`	Root directory to store source code.
`-src,--src-dir <arg>`	Unencrypted source directory (relative to the root directory).
`-xrc,--xrc-dir <arg>`	Encrypted source directory (relative to the root directory).
`-work,--work-dir <arg>`	Work directory to put work files (relative to the environment directory).
`-rcode,--rcode-dir <arg>`	Directory to put rcode files (relative to the environment directory).
`-propath,--compile-propath <arg>`	Propath for compile (separated by commas).
`-pf,--compile-pf <arg>`	Compile pf file (relative to the environment's deploy tool directory).
`-l,--languages <arg>`	Languages code to compile module.
`-rl,--rcode-layout <arg>`	rcode layout (staggered, flat or source).
`-sl,--source-layout <arg>`	Source layout (twoletter or flat).
`-bc,--before-compile <arg>`	Scripts to run before compile.
`-ac,--after-compile <arg>`	Scripts to run after compile.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc module --update`

- YAB

```
cvc module -u -m <arg> [-n <arg>] [-r <arg>] [-work <arg>] [-p <arg>] [-c <arg>] [-bc <arg>]
[-ac <arg>] [-h] [-dh] [-y]
```

`-u,--update`	Update an existing module.
`-m,--module <arg>`	Module name.
`-n,--new-name <arg>`	New module name.
`-r,--root-dir <arg>`	Root directory to store source code.
`-work,--work-dir <arg>`	Work directory to put work files (relative to the environment directory).
`-p,--yab-process <arg>`	YAB process to compile (code-mfg-update, code-fin-update, and so on).
`-c,--code-instance <arg>`	YAB code instance name (mfg, fin, and so on).
`-bc,--before-compile <arg>`	Scripts to run before compile.
`-ac,--after-compile <arg>`	Scripts to run after compile.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

- QDT or SE

```
cvc module -u -m <arg> [-n <arg>] [-t <arg>] [-r <arg>] [-src <arg>]
                [-xrc <arg>] [-work <arg>] [-rcode <arg>] [-propath <arg>]
                [-pf <arg>] [-l <arg>] [-rl <arg>] [-sl <arg>] [-bc <arg>]
                [-ac <arg>] [-h] [-dh] [-y]
```

`-u,--update`	Update an existing module.
`-m,--module <arg>`	Module name.
`-n,--new-name <arg>`	New module name.
`-t,--type <arg>`	Module type (standard or custom).
`-r,--root-dir <arg>`	Root directory to store source code.
`-src,--src-dir <arg>`	Unencrypted source directory (relative to the root directory).
`-xrc,--xrc-dir <arg>`	Encrypted source directory (relative to the root directory).
`-work,--work-dir <arg>`	Work directory to put work files (relative to the environment directory).
`-rcode,--rcode-dir <arg>`	Directory to put rcode files (relative to the environment directory).
`-propath,--compile-propath <arg>`	Propath for compile (separated by commas).
`-pf,--compile-pf <arg>`	Compile pf file (relative to the environment's deploy tool directory).
`-l,--languages <arg>`	Languages code to compile module.
`-rl,--rcode-layout <arg>`	rcode layout (staggered, flat or source).
`-sl,--source-layout <arg>`	Source layout (twoletter or flat).
`-bc,--before-compile <arg>`	Scripts to run before compile.
`-ac,--after-compile <arg>`	Scripts to run after compile.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc module --delete`

```
cvc module -d -m <arg> [-h] [-dh] [-y]
```

`-d,--delete`	Delete an existing module.
`-m,--module <arg>`	Module name.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc module --show`

```
cvc module -s [-h] [-dh] [-y]
```

`-s,--show`	Show all modules in the system.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To add a new module, run:

Note Before you run this command, make sure that the module's root directory and work directory already exist in the system.

```
$ cvc module --add --module-name mfgpro_addon1 --root-dir mfgpro --work-dir mfgpro/work --yab-process "code-mfg-update"
```

To update an existing module, run:

Note Before you run this command, make sure that the new module's root directory and work directory already exist in the system.

```
$ cvc module --update --module-name mfgpro_cust --new-name mfgpro_1 --rootdir mfgpro_1 --work-dir work_1
```

To delete an existing module, run:

Note Make sure that all the files linked to the module have been unregistered before performing this operation. And you cannot undo this operation.

```
$ cvc module --delete --module-name mfgpro_cust
```

To display detailed module information, run:

```
$ cvc module --show
```

cvc module-show

Description

The `cvc module-show` command displays detailed module information.

Usage

Syntax

```
cvc module-show [-h] [-dh] [-y]
```

Option Explanation

`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To display detailed module information, run:

```
$ cvc module-show
```

cvc promote

Description

The `cvc promote` command allows release managers to apply the changes of customization from the source environment to the target environment. The `cvc promote` command uses tickets as promote units.

You only need to specify the tickets to promote as well as the target environment name in the command. The source environment is determined by the configuration of environments.

The `cvc promote` command compiles all program files related to the tickets after applying the changes to the target environment. Then it deploys the generated rcode files to the module's rcode directory.

Make sure that all tickets to be promoted have unpromoted commits in the source environment. For example, there are four environments: `devl`, `test`, `sup`, and `prod`. `devl` is the first environment that developers make changes in, and `prod` is the last. If you want to promote one ticket from `devl` to `prod`, you need to promote it to `test` first, then to `sup` and finally to `prod`. You also need to make sure that all the dependencies related to the tickets have already been promoted, or you need to promote them together.

When promoting tickets to a hotfix environment, the `cvc promote` command checks if the target environment has any unresolved hotfix or emergency tickets. If there are some unresolved hotfix tickets, you need to promote the hotfix tickets first to the target environment. When promoting hotfix tickets, the `cvc promote` command compares the file changes of the hotfix tickets between two environments. A warning displays when the file changes differ between tickets.

If you choose to continue the promotion, the `cvc promote` command performs an auto-backout for hotfix tickets in the target environment, and then promotes the new changes.

Usage

Syntax

```
cvc promote <tickets>... [-s] [-l <arg>] [-nc] [-a] [-all] [-ex] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-s,--simulate`	Perform a dry run but not perform the actual promotion.
`-l,--list-file <arg>`	List containing the tickets to be promoted.
`-nc,--no-compile`	Not compile after code promotion.
`-a,--avail-list`	Show the tickets available to be promoted.
`-all,--all-available`	Promote all the tickets available to be promoted.
`-ex,--exclude`	Exclude the tickets specified in the arguments and list file (used with -all).
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To promote the ticket `CVC-100` from environment `devl` to `test` without compiling, run:

```
$ cvc promote --env test --no-compile CVC-100
```

To promote the tickets specified in the `ticket.lst` file from environment `test` to `prod` with compiling by using the `--list-file` option, run:

```
$ cvc promote --env prod --list-file ticket.lst
```

The following is an example of the content in the `ticket.lst` file:

```
CVC-101
CVC-102
```

cvc register

Description

The `cvc register` command allows release managers to register files into a standard or custom module. Before using the `cvc register` command, make sure that all the files to be registered are placed in the module's predefined root directory; for instance, `/devl/apps/mfgpro/xxsrc`.

The `cvc register` command registers the files into the specified module. It also compiles the programs with the module's propath and reports errors if any of the generated rcode file is different from the existing rcode file.

The `cvc register simulation` command generates a log file that lists all the files to be registered. It also lists the files overlap between modules. After registration, you can use the `cvc checkout` command to check out the files from the custom module for customization.

Usage

Syntax

```
cvc register <files>... -m <arg> -t <arg> [-n <arg>] [-l <arg>] [-nc] [-dr] [-s] [-d] [-e
<arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Module name.
`-n,--note <arg>`	Note to register files.
`-l,--list-file <arg>`	List containing the include & exclude patterns.
`-nc,--no-compile`	Not compile after registration.
`-dr,--deploy-rcode`	Deploy rcode to rcode directory.
`-s,--simulate`	Perform a dry run but not perform the actual registration.
`-d,--delete`	Delete all file records from module.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To simulate the operation of registering the files in the `devl` environment into module `mfgpro_cust`, run:

```
$ cvc register --ticket CVC-100 --module mfgpro_cust --simulate --env devl
```

To register some files according to the `file.lst` file in the `devl` environment into module `mfgpro_cust`, run:

```
$ cvc register --ticket CVC-100 --module mfgpro_cust --list-file file.lst --env devl
```

The following is an example of the content in the `file.lst` file (only including the two listed directories):

```
exclude .*
include src/us/so/.*
include xrc/us/so/.*
```

The following is another example of the content in the `file.lst` file (excluding the `preSAQ022014` directory):

```
exclude .*preSAQ022014
```

To register the files in the `devl` environment into module `mfgpro_cust`, run:

```
$ cvc register --ticket CVC-100 --module mfgpro_cust --env devl
```

cvc rel-compile

Description

The `cvc rel-compile (rc)` command allows release managers to compile file of a specified module. First, the `cvc rel-compile` command finds programs in the module that is specified or that uses the programs you specified in the command. Then this command compiles programs in batches according to the module's compile propath. After that, the `cvc rel-compile` command updates program reference database records, and deploys rcode to the module's rcode directory.

Usage

Syntax

```
cvc rel-compile <files>... -m <arg> [-l <arg>] [-s] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Module name.
`-l,--list-file <arg>`	List containing the files to be compiled.
`-s,--simulate`	Perform a dry run but not perform the actual compile.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To compile all programs in module `mfgpro_cust`, run:

```
$ cvc rel-compile -m mfgpro_cust
```

To compile two files `us/so/sosomt.p` and `us/so/sosorp.p` in module `mfgpro_cust`, run:

```
$ cvc rel-compile -m mfgpro_cust us/so/sosomt.p us/so/sosorp.p
```

To simulate the operation of compiling two files `us/so/sosomt.p` and `us/so/sosorp.p` in module `mfgpro_cust`, run:

```
$ cvc rel-compile -m mfgpro_cust us/so/sosomt.p us/so/sosorp.p -s
```

cvc report

Description

The `cvc report` command allows developers or release managers to view various kinds of reports.

Usage

Syntax

```
cvc report -a | -f | -m <arg> | -e <arg> | -t <arg> | -fh | -pa | -ed | -j | -rf | -ou | -
o | -fm [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-a,--activity-log`	Show the report of user activity logs.
`-f,--file`	Show file information.
`-m,--module <arg>`	Show detailed module information.
`-e,--env <arg>`	Show the commits of the environment.
`-t,--ticket <arg>`	Show the commits of the ticket.
`-fh,--file-history`	Show the change history of the files.
`-pa,--promote-avail`	Show the tickets available to be promoted.
`-ed,--env-diff`	Show different commits between two environments.
`-rf,--recent-files`	Show recent file changes.
`-ou,--online-users`	Show online users.
`-j,--jira-status`	Show JIRA status of the tickets.
`-o,--overlap`	Show the files affected by code update.
`-fm,--file-mappings`	Show file mappings.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --file-mappings`

```
cvc report -fm [-t <arg>] [-f <arg>] [-sf <arg>] [-e <arg>] [-r <arg>]
               [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-fm,--file-mappings`	Show file mappings.
`-t,--type <arg>`	Specify the type to show file mappings (source, include or included_in).
`-f,--file <arg>`	Specify the custom file name to show file mappings.
`-sf,--source-file <arg>`	Specify the source file name to show file mappings.
`-e,--env <arg>`	Specify the environment to show file mappings.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.

`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --activity-log`

```
cvc report -a [-e <arg>] [-u <arg>] [-d <arg>] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-a,--activity-log`	Show the report of user activity logs.
`-e,--env <arg>`	Specify the environment to show activity logs.
`-u,--user <arg>`	Specify the user to show activity logs.
`-dd,--days <arg>`	Show the activity logs within the specific number of days (default: 100).
`-ds,--since <arg>`	Show the activity logs later than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-du,--until <arg>`	Show the activity logs earlier than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --file`

```
cvc report <files>... -f [-m <arg>] [-e <arg>] [-wi] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-f,--file`	Show file information.
`-m,--module <arg>`	Specify the module to show file information.
`-e,--env <arg>`	Specify the environment to show file information.
`-wi,--whole-index`	Only show the files that have whole-index issues.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --module`

```
cvc report -m <arg> [-e <arg>] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-m,--module <arg>`	Show detailed module information.
`-e,--env <arg>`	Specify the environment to show module information.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --env`

```
cvc report -e <arg> [-d] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-e,--env <arg>`	Show the commits of the environment.
`-d,--details`	Show the commits of the environment with file details.
`-dd,--days <arg>`	Show the commits of the environment within the specific number of days (default: 100).
`-ds,--since <arg>`	Show the commits of the environment later than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-du,--until <arg>`	Show the commits of the environment earlier than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --ticket`

```
cvc report -t <arg> [-d] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-t,--ticket <arg>`	Show the commits of the ticket.
`-d,--details`	Show the commits of the ticket with file details.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --overlap`

```
cvc report -o [-t <arg>] [-m <arg>] [-pd <arg>] [-r <arg>] [-mt <arg>]
              [-ms <arg>] [-h] [-dh] [-y]
```

`-o,--overlap`	Show the files affected by code update.
`-t,--type <arg>`	Specify the type to show overlap report (source, include or included_in).
`-m,--module <arg>`	Specify the module to show overlap report.
`-pd,--patch-dir <arg>`	Specify the patch directory.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --file-history`

```
cvc report <files>... -fh [-m <arg>] [-e <arg>] [-l <arg>] [-d <arg>] [-t <arg>] [-r <arg>]
[-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-fh,--file-history`	Report the change history of the files.
`-m,--module <arg>`	Specify the module to show file history.
`-e,--env <arg>`	Specify the environment to show file history.
`-l,--list-file <arg>`	List containing the files to find file history.
`-dd,--days <arg>`	Show the file history within the specific number of days (default: 100).
`-ds,--since <arg>`	Show the file history later than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-du,--until <arg>`	Show the file history earlier than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-t,--to <arg>`	Specify the name of the file where CVC prints the report to.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --promote-avail`

```
cvc report -pa [-e <arg>] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-pa,--promote-avail`	Show the tickets available to be promoted.
`-e,--env <arg>`	Specify the environment to show tickets available to be promoted.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.

`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --jira-status`

```
cvc report <tickets>... -j [-l <arg>] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-j,--jira-status`	Show JIRA status of the tickets.
`-l,--list-file <arg>`	List containing the JIRA tickets.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --env-diff`

```
cvc report <envs>... -ed [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-ed,--env-diff`	Show different commits between two environments.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --recent-files`

```
cvc report -rf [-e <arg>] [-d <arg>] [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-rf,--recent-files`	Show recent file changes.
`-e,--env <arg>`	Specify the environment to show recent file changes.
`-dd,--days <arg>`	Show the recent file changes within the specific number of days (default: 100).
`-ds,--since <arg>`	Show the recent file changes later than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-du,--until <arg>`	Show the recent file changes earlier than a specific date (Date format: *yyyy-MM-dd* or *MM/dd/yyyy*).
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).

`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc report --online-users`

```
cvc report -ou [-r <arg>] [-mt <arg>] [-ms <arg>] [-h] [-dh] [-y]
```

`-ou,--online-users`	Show online users.
`-r,--report-type <arg>`	Set the report type (default, raw, csv, xml or json).
`-mt,--mail-to <arg>`	Set the email address to receive the report file.
`-ms,--mail-subject <arg>`	Set the email subject to receive the report file (token: ${filename}, ${date}).
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To list all the user activity records with the environment, user IDs, time, `cvc` command names, and ticket information, run:

```
$ cvc report --activity-log
```

To list all the files with the module, environment, and file path information, run:

```
$ cvc report --file
```

To list all the files ending with `.md` linked to module `mfgpro_cust` in the `devl` environment, run:

```
$ cvc report --file --module mfgpro_cust --env devl *.md
```

To show the detailed information of module `mfgpro_cust` in key-value pairs, run:

```
$ cvc report --module mfgpro_cust
```

To show the commit history of ticket `CVC-100`, run:

```
$ cvc report --ticket CVC-100
```

To list the commit history in the `test` environment including the ticket message, date, changed file counts, and details, run:

```
$ cvc report --env test
```

To show the files of module `mfgpro_cust`, which overlap with the files in the patch directory, run:

Note The directory `/devl/apps/patch` is prepended to PROPATH when CVC detects the overlap.

```
$ cvc report --overlap --module mfgpro_cust --patch-dir /devl/apps/patch
```

To show the tickets available to be promoted to the `test` environment, run:

```
$ cvc report --promote-avail --env test
```

To show the information about JIRA ticket `CVC-101`, run:

```
$ cvc report --jira-status CVC-101
```

To show the file history of the file `us/so/sosorp.p`, run:

```
$ cvc report --file-history us/so/sosorp.p
```

To show the different commits between environments `devl` and `test`, run:

```
$ cvc report --env-diff devl test
```

To show the file changes in the `devl` environment during the last 10 days, run:

```
$ cvc report --recent-files --env devl --days 10
```

To show the online users, run:

```
$ cvc report --online-users
```

To show the file mappings whose custom file is `us/so/sosorp.p` and source file is `us/so/sosorp.p`, run:

```
$ cvc report --file-mappings --file us/so/sosorp.p --source-file us/so/sosorp.p
```

cvc resolve

Description

The `cvc resolve` command allows developers and release managers to mark file overlaps as resolved.

After a product upgrade, some source files may be updated or moved. You can run the `cvc report --overlap` command to view the custom files affected by the product upgrade. Then you need to review the overlaps and update the custom files if necessary. Finally, you need to run the `cvc resolve` command to mark the overlaps as resolved so that they are not listed in the overlap report again.

The `cvc resolve` command updates the file mappings of the files to be resolved with the latest information of the source files.

You should run this command only in the `devl` environment.

Usage

Syntax

```
cvc resolve <files>... [-l <arg>] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-l,--list-file <arg>`	List containing the files to be resolved.
`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To mark the overlaps of file `us/so/sosorp.p` as resolved, run:

```
$ cvc resolve us/so/sosorp.p
```

cvc resolve-all

Description

The `cvc resolve-all` command allows release managers to mark all file overlaps as resolved.

After a product upgrade, some source files may be updated or moved. You can run the `cvc report --overlap` command to view the custom files affected by the product upgrade. Then you need to review the overlaps and update the custom files if necessary. Finally, you need to run the `cvc resolve-all` command to mark all the overlaps as resolved so that they are not listed in the overlap report again.

The `cvc resolve-all` command updates the file mappings of the files to be resolved with the latest information of the source files.

You should run this command only in the `devl` environment.

Usage

Syntax

```
cvc resolve-all [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-e,--env <arg>`	Name of the environment to operate on.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To mark all the overlaps as resolved in the `devl` environment, run:

```
$ cvc resolve-all
```

cvc revert

Description

The `cvc revert` command allows administrator users to revert file changes by ticket. This command obtains old file revisions of each specified file before the ticket commits. Then it makes a new Git commit with those old file revisions.

After reverting the file changes in the target environment, the `cvc revert` command compiles all the program files reverted by the ticket. This command then deploys the generated rcode files to the module's rcode directory.

QAD recommends that you run the `cvc revert --simulate` command to obtain the accumulated file change list before you perform the actual reversion. In addition, if you run the `cvc revert` command in the first environment like `devl`, you obtain a backup copy of all the file changes saved in your current folder.

Usage

Syntax

```
cvc revert <files>... -n <arg> -t <arg> [-l <arg>] [-s] [-nc] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-n,--note <arg>`	Note to revert files.
`-l,--list-file <arg>`	List containing the files to be reverted.
`-s,--simulate`	Perform a dry run but not perform the actual revert.
`-nc,--no-compile`	Not compile after revert.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To revert the file `us/so/sosomt.p` by the ticket `CVC-100`, run:

```
$ cvc revert --ticket CVC-100 --note "Revert sosomt.p" us/so/sosomt.p
```

To revert the files specified in the `file.lst` file by the ticket `CVC-100` using the `--list-file` option, run:

```
$ cvc revert --ticket CVC-100 --note "Revert the files" --list-file
file.lst
```

The following is an example of the content in the `file.lst` file:

```
us/xx/xxsample1.p
us/xx/xxsample2.p
```

cvc source

Description

The `cvc source` command allows release managers to add source files to a custom module.

After the files are sourced, developers can check out files for customization. Only release managers can source files.

You can only add source files to custom modules. You need to prepare all the source files you want to add in the ticket directory. Or you can use the auto source or copy local option to make CVC find the source files within the module's propath.

If you run this command with the copy local option, those source files are copied to the local ticket folder. If not, those source files are copied to the specified module's directory based on their file format. If the file is a program file, it can be added to the module's src directory. Otherwise, it is added to the module's root directory.

Usage

Syntax

```
cvc source <files>... -m <arg> -t <arg> [-n <arg>] [-a] [-l <arg>] [-cl] [-e <arg>] [-h] [-dh] [-y]
```

Option Explanation

`-m,--module <arg>`	Add source files that belong to the module.
`-n,--note <arg>`	Note to source files.
`-a,--auto`	Perform an auto-source.
`-l,--list-file <arg>`	List containing the files to be sourced.
`-cl,--copy-local`	Copy the source files to the local ticket folder.
`-e,--env <arg>`	Name of the environment to operate on.
`-t,--ticket <arg>`	Ticket number.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To source the files in the ticket folder `CVC-100` to the custom module `mfgpro_cust` (the ticket folder contains the file `us/so/sosomt.p`), run:

```
$ cvc source --ticket CVC-100 --module mfgpro_cust --note "Source us/so/sosomt.p"
```

To scan the propath of `mfgpro_cust` to find the file `us/so/sosomt.p` and source it to the custom module `mfgpro_cust`, run:

```
$ cvc source --auto --ticket CVC-100 --module mfgpro_cust --note "Source us/so/sosomt.p"
```

To scan the propath of `mfgpro_cust` to find the file `us/so/sosomt.p` and copy it to local ticket folder `CVC-100`, run:

```
$ cvc source --copy-local --ticket CVC-100 --module mfgpro_cust --note "Source us/so/sosomt.p"
```

cvc status

Description

The `cvc status` command displays the ticket folder status information that is read from the CVC meta information file. This command also displays the file checkout status. You can filter the result by ticket, login ID, file list, or environment.

Usage

Syntax

```
cvc status <files>... -g | -s [-h] [-dh] [-y]
```

Option Explanation

`-g,--global`	Show global file checkout status.
`-s,--show`	Show the status of file addition/checkout in the current folder.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc status --global`

```
cvc status <files>... [-g] [-e <arg>] [-t <arg>] [-u <arg>] [-l <arg>] [-h] [-dh] [-y]
```

`-g,--global`	Show global file checkout status.
`-e,--env <arg>`	Filter status by environment.
`-t,--ticket <arg>`	Filter status by ticket.
`-u,--user <arg>`	Filter status by user.
`-l,--list-file <arg>`	List containing the file names to filter status.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Usage and Option Explanation of `cvc status --show`

```
cvc status <files>... -s [-h] [-dh] [-y]
```

`-s,--show`	Show the status of file addition/checkout in the current folder.
`-h,--help`	Display help information.
`-dh,--detailed-help`	Display detailed help information.
`-y,--yes-to-question`	Answer yes to all questions.

Example

To display the addition or checkout status of the current ticket folder, run:

```
$ cvc status --show
```

To display the checkout status information of files checked out by the administrator user using the ticket `CVC-100`, run:

```
$ cvc status ---global --ticket CVC-100 --user admin
```

To display the checkout status information of the file `us/so/sosomt.p` in the `devl` environment:

```
$ cvc status --global --env devl us/so/sosomt.p
```

Product Information Resources

QAD offers a number of online resources to help you get more information about using QAD products.

QAD Forums (community.qad.com)
Ask questions and share information with other members of the user community, including QAD experts.

QAD Knowledgebase (knowledgebase.qad.com)*
Search for answers, tips, or solutions related to any QAD product or topic.

QAD Document Library (documentlibrary.qad.com)
Get browser-based access to user guides, release notes, training guides, and so on; use powerful search features to find the document you want, then read online, or download and print PDF.

QAD Learning Center (learning.qad.com)*
Visit QAD's one-stop destination for all courses and training materials.

*Log-in required