AllChange Reference Manual

AllChange Reference Manual

Version 10.0, Sep 2014

Email: support@intasoft.net
Web: http://www.intasoft.net

DISCLAIMER

COPYRIGHT

TRADEMARKS

ACKNOWLEDGEMENTS

Table Of Contents

About the Reference Manual

The reference sections are an alphabetical set of sections for each command.

Each section describes the command by showing how to access the dialog for invoking the command (the *command dialog*) as well as describing the command line syntax.

A brief description of the function of the command is provided followed by one of the following classifications:

File:	Commands which act primarily on operating system files, such as building files from other files, promoting files to shared pools
Part:	Commands which operate on parts, such as adding a new part , deleting a part, renaming a part or changing a part's status; commands which are concerned with checking in and out of parts; and commands which operate on versions
Workspace:	Commands which are concerned with workspaces
Baseline:	Commands which are concerned with taking baselines
CR:	Commands operating on Change Requests
Monitor:	Commands which are concerned with placing monitors on events
Info:	Commands providing reporting/ querying facilities on all databases
Misc:	Miscellaneous commands such as setting your role, setting flags and options, issuing an operating system command

These classifications are also used in ACE to group commands into menus.

Each command dialog will perform (when the **OK** button is selected) an ***AllChange*** operation on the item(s) specified in the first control on the dialog. Other controls in the dialog select options or attributes which determine the precise function of the command.

Command Line Syntax

AllChange commands available from ACEare either *built in* commands or are implemented as ACCELfunctions.

The command line syntax for built in commands will usually be expressed in the following format:

command-name arguments objects

The command is performed on the object(s) specified with the argument(s) specified. Objects and arguments are separated by whitespace from one another.

The arguments may (usually) be supplied in any order before the objects for the command. Some commands have an additional argument which must be supplied after the objects.

- Any arguments which are optional will be enclosed in `[...]`.
- Any arguments which may be repeated will be enclosed in `{...}`.
- Any grouping of arguments is specified with `(...)`.
- Alternative arguments are separated with the | character

Arguments are of three basic types:

1. *-argument-name* — these are switches and are enabled by giving the argument.
2. *-argument-name value* — these are switches with a value. The switch is enabled by giving the argument and also requires a value to be given following the argument, separated by whitespace. If the value itself contains whitespace it must be quoted by a pair of ' (single-quote) or " (double-quote) characters.
3. *argument-name=value* — these assign the argument, which is usually a field name, the specified value. There is no hyphen preceding the argument. A value must be given: many commands treat

the word none specially, using it to mean clear out the argument. If the value itself contains whitespace then it must be quoted by a pair of ' (single-quote) or " (double-quote) characters.

For ease of use through theACCcommand line interface, in the case of some *-argument-name value* switches the *-argument-name* may sometimes be omitted and so is shown enclosed in `[]`. If it is omitted the *value* must come after all other *-argument-name*s, and if several *-argument-name*s are omitted their corresponding *value*s must come in the order shown in the usage section . For example, (part of) the usage of the **copy** command is:

```
copy ([-from] frompart) ([-to] newpart)
```

So the following are valid:

```
copy -from oldname -to newname
copy -to newname -from oldname
copy oldname newname
```

The command line syntax for ACCEL function will usually be expressed in the following format:

```
call(parameters)
```

where the *parameters* are a comma separated list of parameters.

If an individual object contains spaces (such as a part or file name) it must be quoted.

The above rules are not usually relevant when using the ACE interface since this deals with them automatically.

The command line for built in commands may be used in the ACC interface and is also used as the argument to the ACCEL function `interpret` in order to invoke ***AllChange*** commands from within ACCEL. It may also be used from the ***AllChange*** **Command** dialog.

The ACCEL functions may be invoked from a command line using the **Eval** built in command.

A description of the command dialog options and arguments is then given in the **Options** section.

A list of related commands is then given followed by a concepts section giving a list of references to other sections.

The section will provide a detailed description of the command, followed by examples.

There is also a section in the ***AllChange*** Administrator Manual which discusses the requirements for the command definitions for the command being described. The command definitions are kept in an external ***AllChange*** system file: they implement the conditions which must be satisfied and the actions to be executed when performing (most) ***AllChange*** commands. A default set of command definitions are supplied with the system; the System Administrator may have tailored these to suit the site's requirements, but their functionality should match the description given in this manual.

Scripts of Commands

In addition to executing individual commands interactively from ACC or ACE a sequence of ***AllChange*** commands may be placed in a file (with a standard text editor) and executed as a *script* by entering the command:

```
@file
```

at the ACC prompt or in the ***AllChange*** **Command** dialog from ACE. *File* should specify the operating system path to the file. This permits a complex series of commands to be repeated or run non-interactively.

It is also possible to run ACC (but not ACE), have it execute a script file, and then exit, without any interaction — see ACC.

Summary of Commands

File Commands

Autobuild	generates build dependencies
Build	performs a system build
Check In	checks files in
Check Out	checks files out into a selected directory
Diffs	displays the differences between two text files
Edit	Invokes an editor for a file
Keywords	scans files for keywords
Merge	allows three text files to be merged
Open	Opens a file
Promote	promotes objects from a workspace to a pool
Release	copies releasable objects from a pool to a release directory
Rename	Renames a file
Report BTs	pretty prints BT files
Seebuild	displays the dependency information held in a buildfile

Part Commands

Add	declares new parts to the system and adds them to the parts database.
Alter	update general fields of parts.
Alter Part Class	change the class of a part .
Alter Part Flags	change the flags of a part.
Alter Part Obsolescence	change the obsolete flag for a part.
Alter Instance	updates the general fields of an instance of a version of a component type part.
Alter Version	updates the general fields of a version of a component type part.
Alter Version Flags	change the flags of a version of a component type part.
Alter Version Obsolescence	change the obsolete state of a version of a component type part.
Archive Part	allows parts to be archived off line
Archive Version	allows an individual version to be archived off line
Check In	checks in a part from the current workspace.
Check Out	checks out a copy of a part to the current workspace
Delete	removes a part and all its children from the parts database.
Delete Instance	deletes an existing instance of a version of a component type part.
Delete Version	deletes an existing version of a component type part.
Diffs	compute the differences between two part versions.
Display Version Tree	show a tree view of a components versions
Get Version to Directory	extract versions to a directory unlogged
Copy	copies a part and all its children to a new part.
Edit	invokes your editor with the file corresponding to a part.
Import Part	imports an archived part
Issue	checks out a copy of a part to the current workspace.
Merge	merges the changes between two part versions into a version which is checked out for edit.
Move	moves a part to a new subsystem .
New Instance	creates new instances of a version of a component type part.
New Version	creates a new version of a component type part.
Rename	updates the name of a part.
Return	returns parts which have previously been checked out from the workspace back to the system.
Status	changes the current status of parts.
Use	specifies that one part uses another part.

Workspace Commands

Alter Check-out	updates the fields of a check-out record.
Attach Workspace	attaches the user to a workspace.
Check Check-out Changes	checks which workfiles for checked out versions have been changed.
Delete Check-out	deletes check-out records bypassing the normal check in/return process.
Update	updates a workspace to the latest version of parts.
Update Workspace	updates a workspace to the latest version of parts.
Workspace Registrations	specifies the registered versions and pools for a workspace.

Baseline Commands

Add	creates a baseline of parts according to specified criteria.
Alter	updates the general fields of a baseline header.
Alter Class	change the class of a baseline.
Alter Obsolescence	change the obsolete state of a baseline.
Archive Baseline	archives a baseline off line
Check Baseline Check-outs	checks whether the versions checked out to a workspace are the versions in a baseline
Copy	creates a new baseline based from an existing baseline.
Create Baseline Hierarchy	Creates a meta baseline hierarchy based on another baseline
CRs For Baseline	updates a baseline by the versions which implemented selected CRs
Delete	removes existing baselines.
Diffs	computes the differences between two baselines
Get Baseline to Directory	extract a baseline to a directory unlogged
Import Baseline	Imports an archived baseline
Rename	renames a baseline.
Status	changes the current status of baselines.
Update Baseline	update the details of a baseline.

CR Commands

Alter	updates the general fields of CRs.
Alter Class	changes the class of CRs.
Alter Obsolescence	changes the obsolete flag on CRs.
Assign	assigns a CR to a user.
Archive CRs	archives CRs.
Copy	create a copy of a CR.
Delete	deletes a CR.
Edit CR Text	allows the text of a CR to be edited.
Export to MS Project	exports CR data to MS Project .
Get CR Text	retrieves the text of a CR and puts it in a file.
Import CRs	imports previously archived CRs.
New	creates a new change request (CR).
Rename	renames a CR.
Put CR Text	updates the text of a CR from a file.
Status	changes the current status of CRs.

Item Relationship Commands

Alter Baseline Item Affected	updates the fields of a baseline item affected relationship
Alter CR Item Affected	updates the fields of a CR item affected relationship
Alter Part Item Affected	updates the fields of a part item affected relationship

Vote Commands

Add Vote	casts a vote
Alter Vote	updates the fields of a vote cast
Delete Vote	deletes a vote cast
Vote Pass Next	passes the vote to the next person/group in a serial vote

Monitor Commands

Add	places a monitor on an item for a particular event.
Delete	deletes a monitor on an item.

Information Commands

Column Report Wizard	creates a new tabular report format.
Eval	evaluates an expression and prints the result.
Find	finds parts according to search criteria
Report	produces user-defined database reports.
Report Wizards	guides you through the process of generating a report.
Report BTs	report on build threads

Miscellaneous Commands

OS Command	allows an operating system command to be issued from within the ***AllChange*** system.
Set Role	sets the current user role .
Set System Flag	sets system flags.
Version Control Command	allows an version control tool to be invoked from within the ***AllChange*** system.
@*file*	executes *file* as a script of ***AllChange*** commands.
Generate URL for selection	generates the URL for a currently selected item in a list.

Functions

Who	shows who is currently logged onto ***AllChange***.
Send Mail to User	allows an email to be sent to an ***AllChange*** user
Send Mail to AC Users	allows an email to be sent to all ***AllChange*** users
Read Functions	re-reads the definition of project specific functions
Read Dev Functions	re-reads the definition of names function definition files
Server Password	allows user names and passwords to be administered

Report Commands

Open	Opens an ACREPORT format file for modification
New	Creates a new ACREPORT format file

ACReport

ACREPORT — Overview

ACREPORT provides GUI based report generation facilities. ACREPORT involves two separate stages:

1. An ACREPORT *format file* is designed initially, which specifies the layout and content for a desired report.
2. An end-user runs a report using a particular ACREPORT format file to produce output from an actual ***AllChange*** database.

AllChange also supports text based report generation facilities which may be used, for example, for reports generated for integration with third party applications such as MS Word.

ACREPORT offers two notable advantages over text reports:

1. At the design stage, instead of having to use a text editor, together with a comprehensive understanding of ***AllChange*** text report syntax, to create report formats, ACREPORTs are created from within ACE in a WYSIWYG style, including wizards and help for setting up the desired format.
2. At the run stage, instead of having to produce plain text output, ACREPORT takes advantage of Windows fonts, printing capabilities etc. to produce ``professional'' looking output.

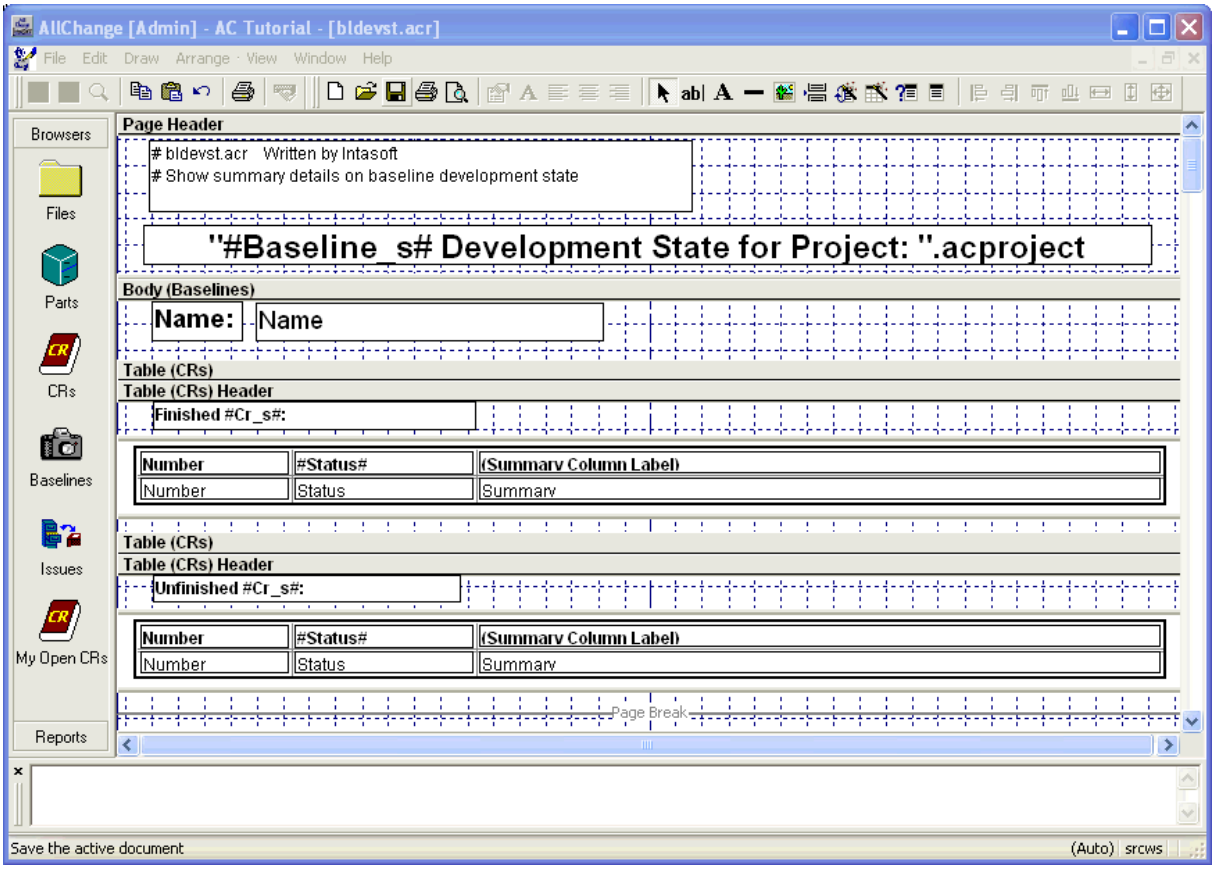
ACREPORT is available from the ACE **Reports** menu which allows acreports to be created, modified and executed/run. In addition acreports may be run from the standard ace report dialogs. The **Reports** menu offers:

New: allows a new acreport to be created

Open: allows and existing acreport to be opened

filenames: provides a list of the most recent acreport's that you have accessed and allows them to be opened.

Once an acreport is opened/created it will be shown in a window. When an ACREPORT window is up front ACE goes into ACREPORT mode and the menu changes to the ACREPORT menu. Other ACE windows are still available and selection of one of these will change back to normal ACE mode.



For full details of using ACREPORT see:

1. **ACReport Format Files: a description of report format file handling operations**
2. **Designing reports**: a description of how to design ACREPORT format files. This should be read by users who want to modify supplied ACREPORTs or design their own from scratch.
3. **Printing/Viewing Reports**: a description of how to run an ACREPORT once the format file has been created.

ACREPORT Format Files

ACREPORT is used to design report format files which are later used to create reports on demand.

ACREPORT files have a suffix of `.acr` and may reside in the ***AllChange*** system directory, the ***AllChange*** project directory or the users Home directory.

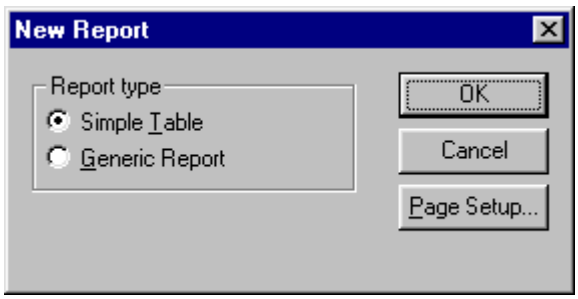
The format files contain the following information:

- The design and layout of the report
- The page size and orientation
- The default font
- Grid settings

Actions on report format files may accessed from the **File** menu when in ACREPORT design mode.

New Report

File | New allows a new report to be created.



Page Setup... allows the desired page size and orientation to be selected.

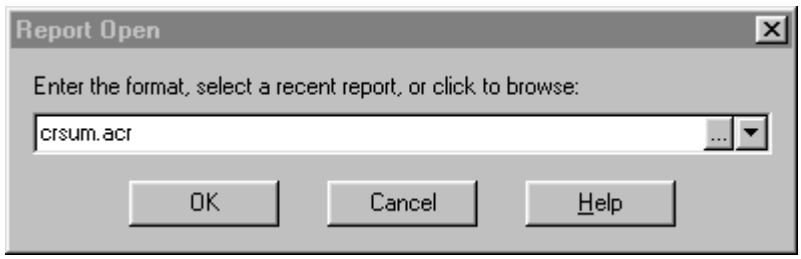
Two types of report may be created:

1. **Simple Table** is suitable where the data is to be lain out in simple row and column tabular format, such as in ACE browsers. Tables, although simpler than generic reports, do *not* allow any further sections, tables etc. to be placed inside them and so, for example, would not be suitable if a report on CRs is to contain a table of status log information within the information on each CR
2. **Generic Report** permits more detailed and complex output on each item reported on, such as in ACE viewers. Generic reports may include subsections and tables.

On selecting OK a new report design is created to contain a table or initial section according to the type selected. This will cause the New Table/Section Wizard to be presented, on completion of this the new report may be edited as required in design view.

Open Report

File | Open allows an existing report to be opened in design view.



Select the report that you wish to open

Close Report

File | Close closes the topmost open ACReport.

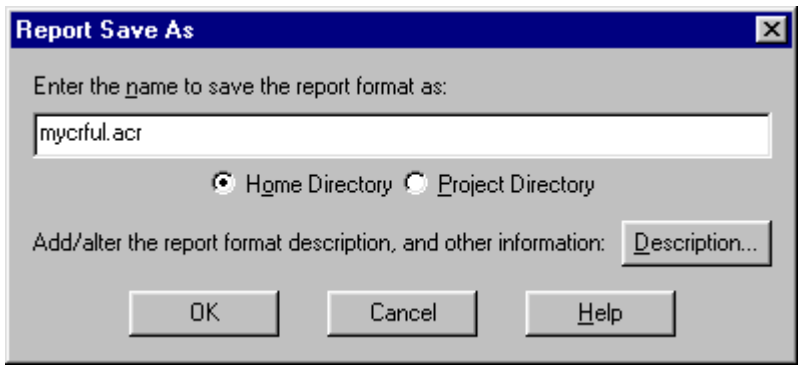
Save Report

File | Save saves the topmost open ACReport.

When you use **File|Save**, if you are an Administrator for the current ***AllChange*** project then a report format file read in from either the project or system directory will be saved back to the project directory , while a file read in from either your home directory or an explicit path will be saved back to where it came from. If you are not an Administrator then files read from the project or system directory are saved back to the home directory.

File | Save As saves the topmost open ACReport under a new name.

When a report format is saved the design of the report together with grid settings, page setup and default font are saved in the report format file (and are likewise restored on opening the report format).



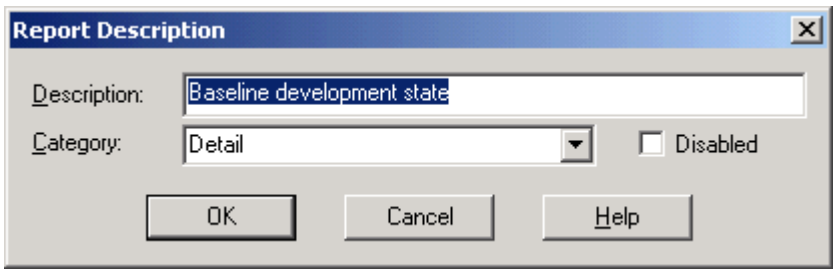
Save As may be used to save a copy of an ACReport to a new ACReport. The name of the new report should be specified. If no extension is specified then `.acr` is used.

The first two characters of the filename should be the 2 letter abbreviation for the database that is the primary subject of the report (**bl** for baselines, **cr** for CRs, **pa** for parts, **co** for check-outs) in order for the report to be made available in the correct report format dialogs.

If a full path is specified then the Home or Project directory selection is ignored.

If just a filename is specified then the report may be saved either in the users **Home directory** or in the **Project directory**. Reports that are in the Project directory will be available to all users, those in the home directory are only available to the user for the home directory. Only AllChange Administrators may save in the project directory (Note that the home directory may be set using **Misc | Options**)

The **Description** button may be used to specify additional information about the report, this will be used when the report is displayed for selection as a report format.



The **Category** defines the type of report and is used by the Report Wizard. Possible values are:

Detail: for detailed reports providing multiple lines of information for each item reported on.

Brief: for summary/tabular reports

Status Log: for reports on status logs

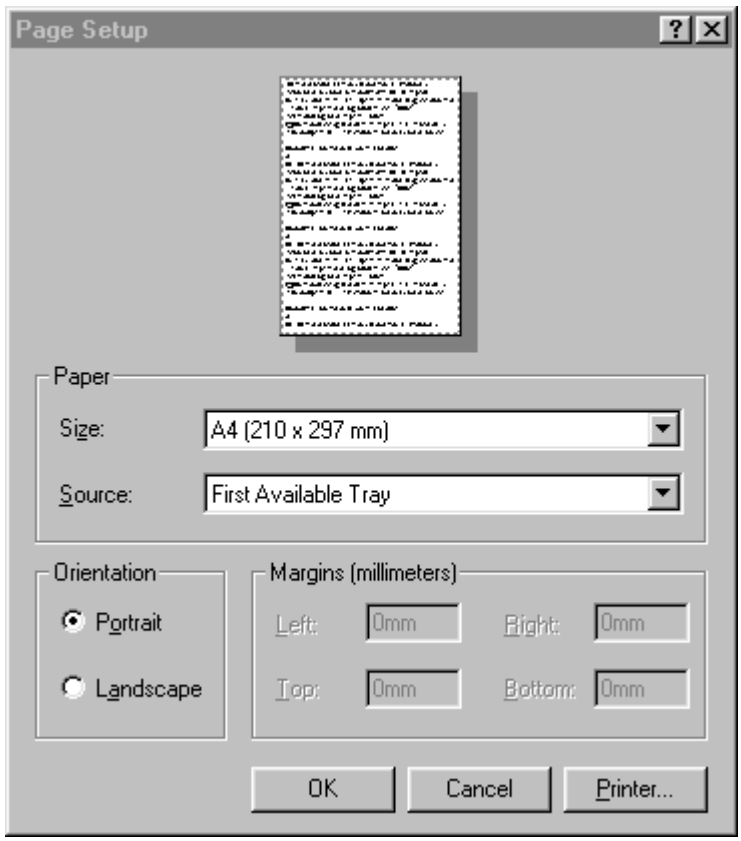
Other: for reports which do not fit any other category

If **Disabled** is selected then the report will not be made available in the report dialogs.

This description information may be modified (as well as additional parameters be set) by the AllChange administrator using ACCONFIG.

Page Setup

File | Page setup allows page attributes for the report to be defined.

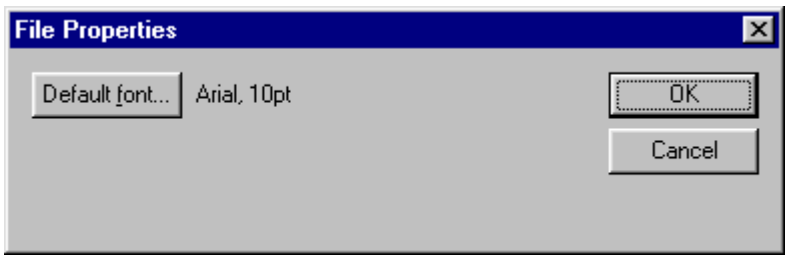


Paper size, source and orientation and the printer to be used are all supported. The setting of margins is not supported at the date of writing.

The page size/orientation is altered once there are elements in the report these will be moved to fit within the page boundaries if necessary, so it is best to fix this at the outset. If the end-user prints using a different page size elements may have to be moved in the output.

Report Format Properties

File | File Properties... allows the default font for the report to be specified.



Designing Reports

About Designing Reports

An ***AllChange*** Administrator, or advanced end-user, can use ACREPORT to tailor supplied report formats or create new ones from scratch. The saved report may then be run by other ***AllChange*** users.

Creating reports

Reports may be created in two ways:

1. From an existing report
2. From scratch

Creating a report from an existing report

Often the simplest way to create a new report is to start with an existing one which does the same sort of thing as you have in mind and then add, alter and delete elements as desired. Use **Reports | Open...** to read an existing ACREPORT format file, tailor as desired, then **File | Save As...** to save the new report under a different name.

Creating a report from scratch

To create an ACREPORT from scratch, select **Reports | New...** to bring up the **New Report** dialog and select the type of report required.

Whichever type of report you pick you will be presented with the first page of the new section/table wizard, **What to Loop Round**. Select the ***actual AllChange*** database (e.g. **cr**, **baseline**) or pseudo-database (e.g. **class** , **workspace**) which is the primary subject of the report.

For **Simple Table** reports there may be a second page in the wizard allowing you to populate the table from a browser, otherwise the number of rows and columns should be specified

A newly created report has three sections: **Page Header**, **Body** (or **Table**) and **Page Footer**. You are now in the same situation as you would be when editing a report, as described in Editing ACReports.

Report Elements

An acreport is made up of *elements*. There are two basic kinds of elements:

- *sections:* these do not correspond to anything that is finally seen in the output when a report is run: instead they determine what data is actually visited in the ***AllChange*** database. Sections may contain further (sub-)sections and/or objects
- *objects:* these do tend to generate something which can actually be seen in the output.

Editing ACReports

In design view, where you are editing the report, elements are added, modified and deleted via a WYSIWYG GUI interface. The following basic element manipulation functionality is supported:

- Elements may be selected using left click; multiple objects (not sections) may be selected using **SHIFT** and **CTRL** click
- Size and position may be edited by dragging and clicking.
- **CTRL** drag may be used to create a copy of and move selected objects.
- Arrow keys may be used to move/nudge objects by one pixel for fine positional adjustments.
- **SHIFT** and arrow keys may be used to resize objects by 1 pixel.
- The **TAB** key may be used to successively select objects in turn in document order (**SHIFT** + **TAB** may be used for reverse document order).
- **Cut**, **Copy**, **Paste** and **Delete** may be used on elements. Pasted elements appear in the same place as they were copied from (but slightly offset).This may be used to copy/move elements between different reports if multiple reports are open; however, note that if the destination report's sections are not as large as/in the same position as the source's this may cause unanticipated results, including some elements not being pasted.
- **Properties** may be modified from right click or **Edit** menu

When the report is run the elements in the design are executed/evaluated to produce the actual report output. The horizontal positioning of objects relates directly to their horizontal position in the output: an object 1 centimetre in from the left margin in design view produces output 1 centimetre in from the left margin in the output. The vertical positioning of elements does not bear quite such a direct relation as things like section bars are removed from the output, certain sections may not be executed or may be looped around to produce many repetitions in the output, there are page breaks, and so on. Essentially, though, the vertical distance (in white space) between consecutive elements is maintained in the output, so vertical positioning does matter.

In design view section markers, page breaks etc are shown within a *grid*. The grid can be a useful aid in creating your layout/design.

One of the first things you should do when designing a report is ensure the page size and orientation are as intended: use **File|Page Setup...** to check or set this. If you alter the page size once there are elements in the report these will be moved to fit within the page boundaries if necessary, so it is best to fix this at the outset. If the end-user prints using a different page size elements may have to be moved in the output.

You may also wish to check **File|File Properties...** to set the report's default font for text output.

Arranging Report Objects

The **Arrange** menu is used to access facilities for arranging multiple report objects. Facilities are supported for aligning objects, making objects the same size and use of the grid.

The last selected object is called the *dominant* object and its sizing handles' colour reflects this: the other selected objects are arranged to match the dominant object.

Align

Allows the currently selected objects to be aligned at their top, bottom, left or right edge

Make same size

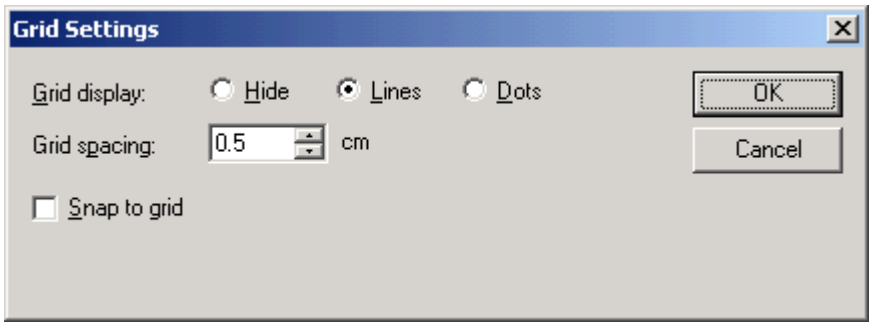
Allows currently selected objects to be made the same width or height or identical in size

Snap to Grid

Snap to grid determines whether elements are snapped to the grid as they are created and moved.

Grid

Allows attributes of the grid to be specified for the report design.



Grid display determines whether/how the grid is shown

Grid spacing determines the vertical and horizontal spacing (in centimetres).

Snap to grid toggles whether objects are snapped to the grid when they are created/moved

ACReport Sections

Sections in an ACReport determine what data is actually visited in the ***AllChange*** database. Sections may contain further (sub-)sections and/or objects.

You may create four kinds of sections in an ACREPORT:

1. Group Sections
2. Conditional Sections
3. SubSections
4. Tables

All ACReports contain:

1. A **Page Header** which is a special case of a section
2. A top-level section which is either a **Table** for a simple tabular report or the **Body** for a generic report.
3. A **Page Footer** which is a special case of a section

None of these sections may be deleted or moved and additional top level sections may not be added.

A section is shown with a title bar, a content area and a thin bottom bar.

Sections always extend the complete width of the page.

The space occupied by a section's title bar and bottom bar do not appear in the output when the report is run. All space within the content area does appear.

A section's title bar and bottom bar are highlighted if either is selected (by left clicking on it).

Selecting a section removes all other selections; only a single section (and no objects) may be selected at a time.

Objects and further sections may be inserted into the sections content area: if these cause the section to be enlarged the bottom of the section, and subsequent elements in the report, are moved downward.

Sections may be cut, copied or deleted by selecting the desired action from either the **Edit** menu, toolbar or the right click menu.

Each section has properties which determine various attributes of the section. Each type of section has different properties. Properties may be accessed by double clicking on the title bar or from the right click or **Edit** menus.

Moving a section

Left click and drag on the title bar to move the section within the report. As soon as you start to move a section it, together with its contents, are removed from the report, with subsequent sections moved up to fill the space it occupied.

As you move the mouse the section and its contents move with the mouse cursor.

Position the top of the section's title bar where you want it to be inserted and left click: the section is inserted there and subsequent sections moved back down, just as when inserting a new section.

Resizing a section

The size of a section may be modified by dragging the section's bottom bar up or down.

In addition if necessary a section will be enlarged when elements are inserted into the content area if there is not enough space for the new objects.

Sections have a minimum size (even if there is no content) beyond which they cannot be shrunk; a section cannot be shrunk so that it is too small to contain its last element.

Creating a new section

A new section is created by selecting the desired **SubSection Tool**, **Table Tool**, **Conditional Section Tool** or **Group Section Tool** in the toolbar (or from the **Draw** menu).

The mouse cursor changes to a crosshair: place this where the (top of) the new section is to be inserted in the report format and left click. The new section will be inserted at this position (if permitted, and possibly moved a little so as not to overlap certain other elements), and all elements below it will be moved down to accommodate the new section.

You will then be presented with a wizard for a subsection or table or the new section's **Properties** dialog for conditional or group sections which you can adjust as desired.

Page Header/Footer Sections

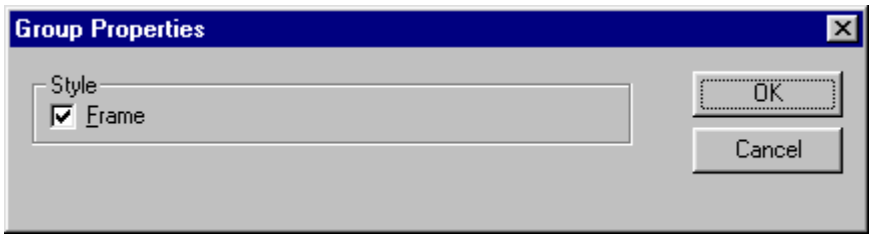
All reports have a **Page Header** and a **Page Footer** sections. The content of these is output in the header and footer areas on each page produced when the report is run, with a horizontal line below/above it. Unlike other sections, the size of the output these produce is fixed, rather than dependent on the size of the section in design view. Only very simple objects should be placed inside page headers/footers: typically fixed text, page number and/or date/time.

Group Sections

Group sections simply allow a number of elements to be grouped together for convenience or to frame them for visual effect.

On insertion of a group section the section properties dialog will be displayed.

Group sections have a single **Frame** property



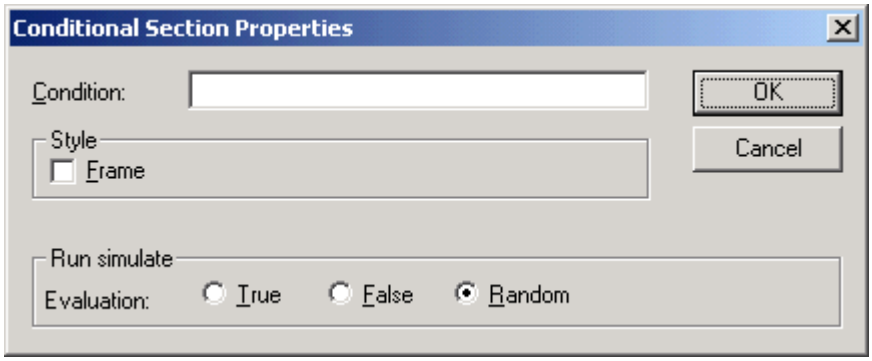
If Frame is set then a frame is shown in a design view around the group and will appear in the output when the report is run.

If a framed section flows across pages the frame is terminated and restarted on each page. Only one frame may be active at a time: *you should not nest further sections with frames inside a section which has a frame*.

Conditional Sections

Conditional sections allow a number of elements to be grouped together which, according to a condition, either will or will not appear in the final output.

On insertion of a conditional section the properties dialog will be shown allowing the condition to be defined.



The **Condition** may be any valid ACCELcode. If it evaluates to true the section is executed; if false it is not executed, and no space will appear in the output for the section.

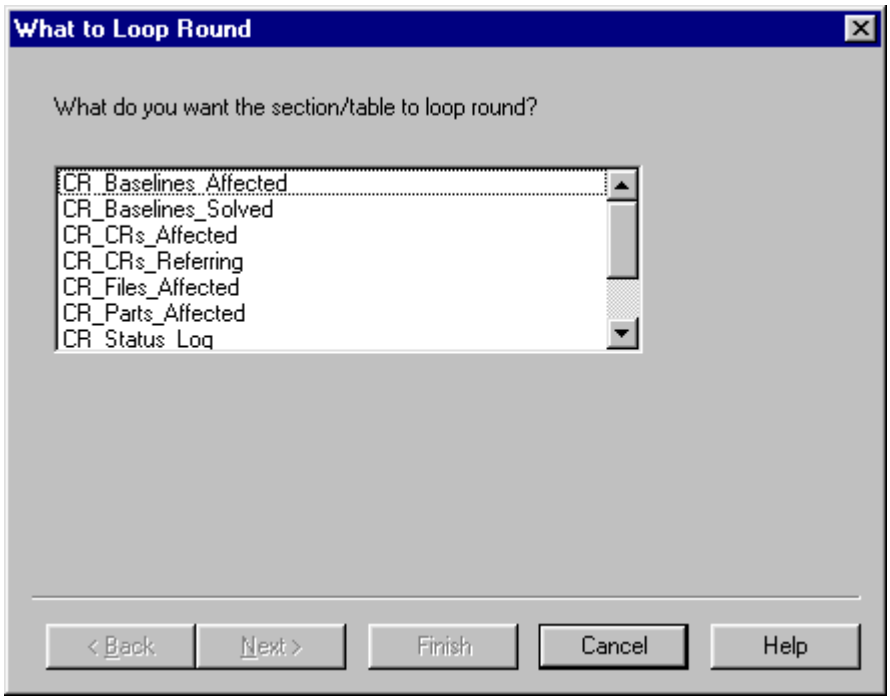
The **Frame** property causes the section to have a frame in the output

The **Run simulate** property allows you to control whether the condition should be treated as always returning true, always returning false, or returning either true or false (random) if simulating a report run.

SubSections

Subsections allow a number of elements to be grouped together which will be looped around as many times as there are items to visit (e.g. records in a database). For each time through the loop (``iteration'') the whole of the section is executed, restarting from the top each time. The top-level **Body** section of a generic report is a (slightly specialised) type of subsection.

On insertion of a subsection (or creation of a new generic report) the sub section wizard shows the **What to Loop Round** dialog.



Select here the items that are to be the subject of the section. This will normally be a section of the database but may also be certain special cases such as the *defined arbitrary fields*.

The choices available varies according the context showing only choices pertinent to the closest enclosing section's current item to be visited.

For example, if a new subsection is inserted within a section whose database is CRs these would include CRStatus Log, CR Parts Affected, CR Versions Solved etc.; choosing, say, CR Parts Affected would cause the new subsection to visit and produce output on the parts affected by the current CR it is reporting on. If the enclosing database has arbitrary fields then **Defined arbitrary fields** will be among the choices: this will produce a subsection visiting each arbitrary field defined at your site for that type of database.

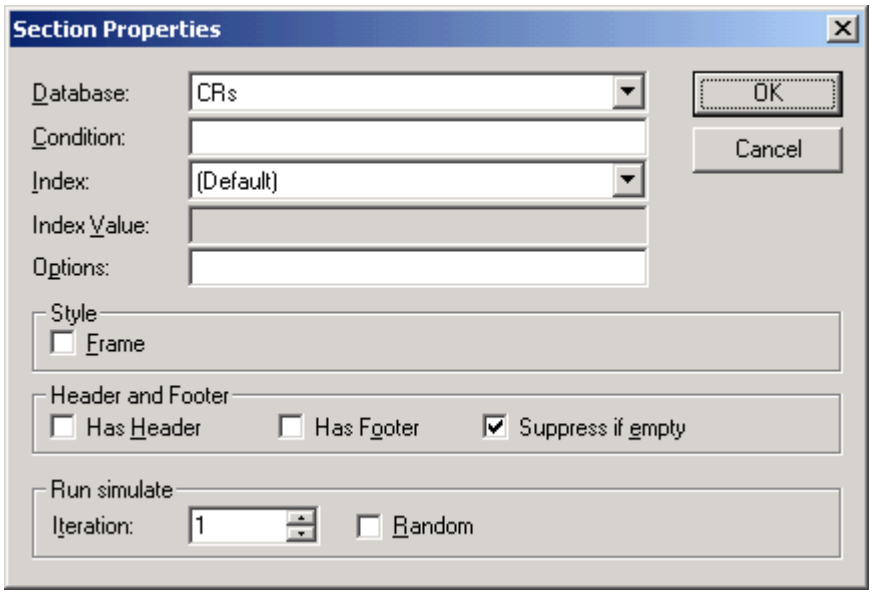
The selection will determine how ACREPORT populates the subsection's **Database**, **Condition**, **Index** and **Index Value** properties on conclusion of the wizard. You may alter these if required from now on; if there are no suitable offerrings in the context of where the new subsection is being inserted you will want to do so.

Select **Finish** to create the new section; you may later edit the subsection's properties further in the usual fashion. Selecting **Cancel** removes the section.

(In the unlikely case that you do not want the wizard, pick **SubSection** from the **Draw** menu, not **Sub-Section Wizard** or the toolbar.)

Subsection Properties

A subsection (or Body) properties determine the data that may be output in the section



The **Database** property specifies what ***AllChange*** database (or pseudo-database) is to be visited during the iteration.

The **Condition** property may be any valid ACCEL expression. It is reevaluated on each iteration. If it evaluates to true the section is executed; if false it is not executed, and no space will appear in the output for this iteration of the section.

The **Index** property determines which index to iterate through. It is only available for real ***AllChange*** databases.

The **Index Value** property allows the items visited when running the report to be limited to items those specified. The Index Value is evaluated as an ACCEL expression. Where an Index is specified the value must match the index entry, where no Index property is available the main field of the records visited must match the Index Value.

Examples:

- A subsection to visit all CRs assigned to the current user: **Database** is **cr**, **Index** is **Assignee**, and **Index Value** is **user**.
- Inside a subsection for Check-outs, a subsection to output (detailed) information on the workspace checked out to: **Database** is **workspace** and **Index Value** is **iu_wspcname**.

The **Frame** property causes the section to have a frame in the output. The frame grows to enclose the output from all the iterations (i.e. not a separate frame for each iteration: to achieve this, fill the subsection's content area with a Group Section containing all the elements and set the Frame property on this Group Section instead). If the subsection has a header or footer this is enclosed in the frame too.

The **Header and Footer** properties allow the subsection to have a header and/or footer. Checking these creates a header or footer area at the top or bottom of the subsection respectively; unchecking them removes the header/footer. The header section appears before any further elements in the subsection prior to the first iteration; the footer section appears after any elements in the subsection after the last iteration.

Header and footer are shown in design view with their own title bar and bottom bar; they are attached to the subsection's title bar and bottom bar respectively and cannot be moved. Header and footer sections may be enlarged or shrunk by dragging their bottom bar down or up, with all subsequent elements being moved correspondingly; dragging the *subsection's* bottom bar down or up moves the footer with it (retaining the footer's size).

Suppress if empty determines what happens if (for whatever reason) no items are visited during the iteration: if it is checked (the default) no header or footer is produced, unchecking it causes the header and/or footer to *always* be output. The header and footer of the top-level **Body** section of a generic report serve as the overall report header/footer.

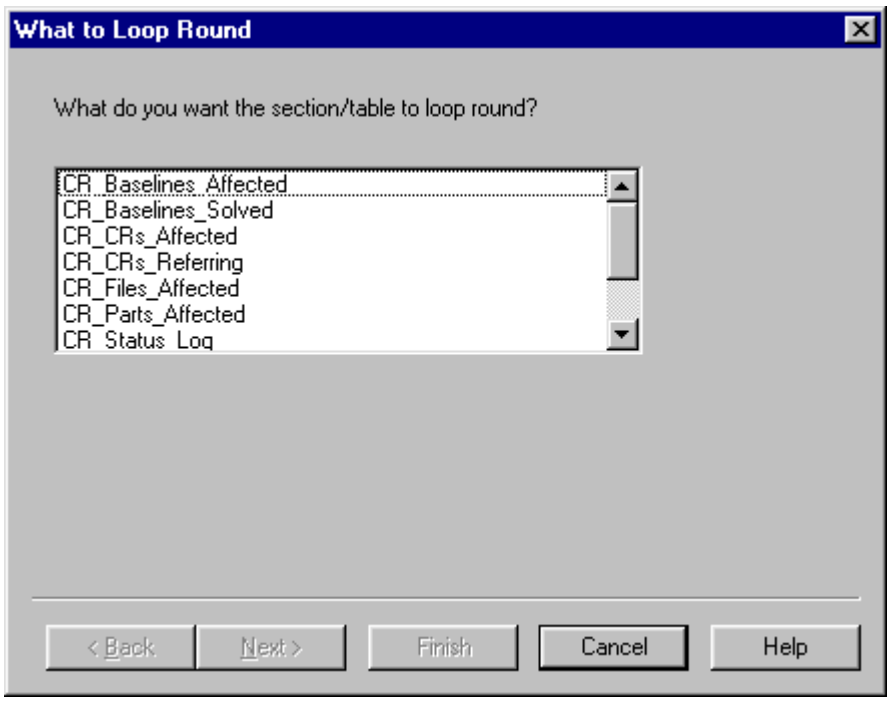
The **Run simulate** property allows you to set the number of iterations to a fixed number or to a random number from 0 up to and including the specified number if simulating a report run.

Tables

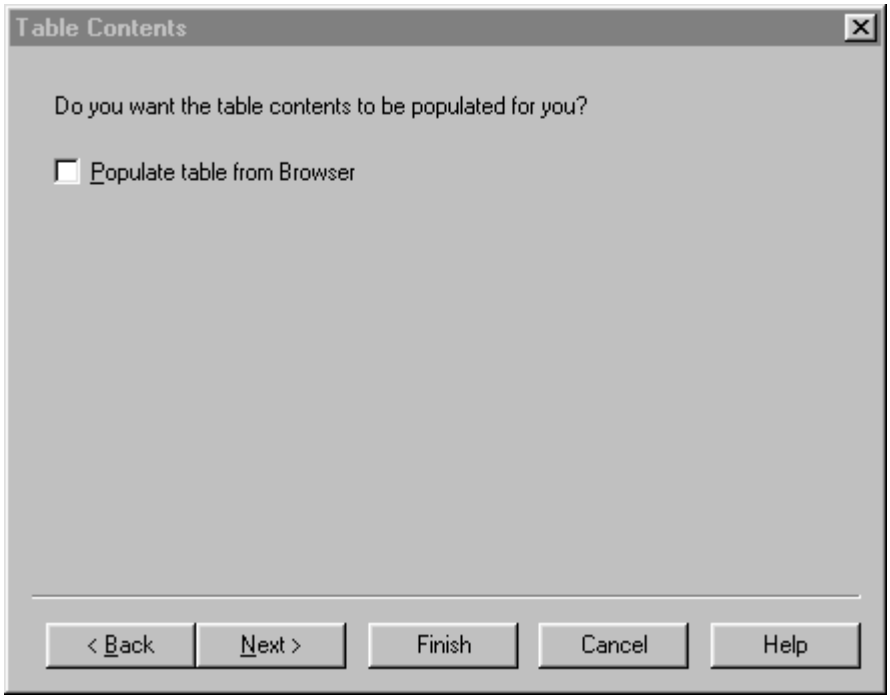
Tables are a specialised type of subsection. Like subsections, tables allow a number of elements to be grouped together which will be looped around as many times as there are items to visit (e.g. records in a database). They produce output in row and column format. For each time through the loop (``iteration'') the whole of the table is executed, restarting from the top each time. The top-level **Table** section of a simple table report is a (slightly specialised) type of table.

Unlike subsections, tables can contain only objects, not sections; and these objects can only appear in the table's *cells*. When you add objects into a table they will be positioned at the nearest cell. This is independent of any grid settings, and the grid is not shown inside tables. You may not alter row positions/heights. You may alter column positions/widths by clicking and dragging the ``splitter'' cursor which appears as the mouse is moved over these.

On Insertion of a table (or creation of a new simple table report) the table wizard shows the **What to Loop Round** dialog as for subsections.

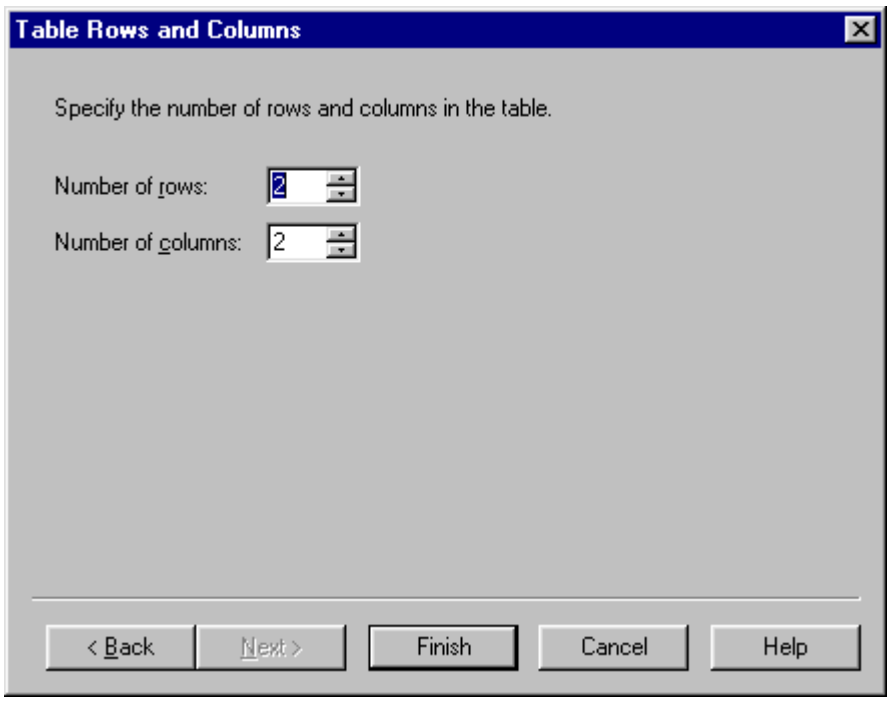


If the database chosen has a corresponding ``browser'' in ACE(i.e. a place where its records are displayed in tabular format in ACE, such as the CR browser or the **Parts** or **Status Log** tabs of the CR Viewer) then the Table Contents dialog is displayed when Next is selected:



Select **Populate table from Browser** to cause the table to be filled initially with rows, columns and cells like those seen in the browser; otherwise a blank table is created.

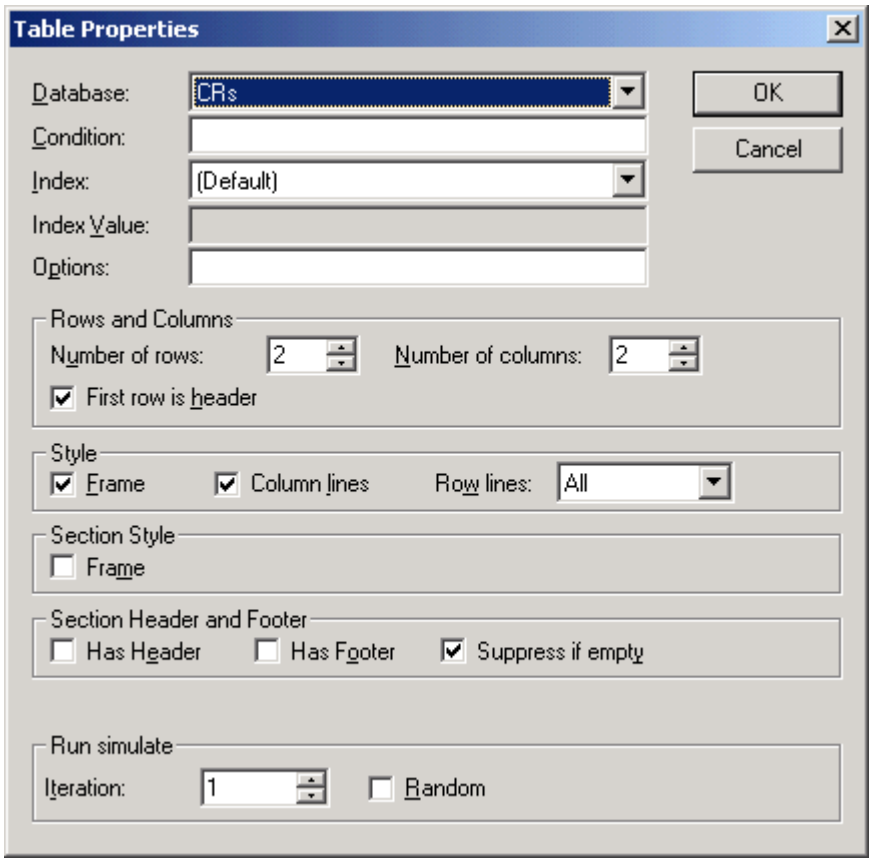
If not populating the table from a browser then the Table Rows and Columns dialog allows the number of Rows and Columns in the table to be specified:



The table properties may be modified later in the usual fashion

Table Properties

Many of a table's properties are those of a subsection and are described in Subsections. Only additional table properties are described here.



Number of columns sets the number of columns in the table. Once the table has objects in its cells, incrementing this number causes new columns (of minimal width) to be added into the far righthand side of the table; decrementing this number removes columns, together with their contents, from the far righthand side. There is a minimum of 1 column in a table.

Number of rows sets the number of rows in the (*design* of the) table. Once the table has objects in its cells, incrementing this number causes new rows to be added into the bottom of the table; decrementing this number removes rows, together with their contents, from the bottom. There is a minimum of 1 row in a table.

First row is header determines whether the first row in the table is treated as a header row: if it is checked (the default) the first row is the header and subsequent rows are the table contents; if it is not checked all rows are the table contents. The table header row is output at the start of the table *and* at the start of each new page onto which the table flows; it is used to show titles for each of the columns. Note that a table may have a section header (and/or footer) independently of whether it has a header row: these are produced before/after the whole table. Usually a table will have just one data row (i.e. 2 rows if the first row is header, 1 if not), though you can have more if you wish. Each data row is output once per iteration, so a table with 1 data row will produce 1000 rows of output if 1000 records are visited.

A table has several **Style**s of its own.

- **Frame** gives the table a frame (which looks better than giving it a subsection frame).
- **Column lines** produces lines between table columns.
- **Row lines** produces lines between table rows, and may have be:
  - **None**: no lines at all
  - **All:** lines between every row (including the header row)
  - Header only:. if the table has a header row a double line is be drawn below it.

In design view tables are shown with dashed lines between columns/rows (to show where cells are) even if the output will not have column/row lines.

ACReport Objects

ACREPORT objects are the elements of an ACReport which cause output to be generated when the report is run.

There are five types of objects in an ACReport:

1. Text Objects
2. Expression Objects
3. Image Objects
4. Page Break Objects
5. Line Objects

A new object is created by selecting the desired **Text Tool**, **Expression Tool**, **Page Break Tool, Image Tool** or **Line Tool** in the toolbar (or from the **Draw** menu). The mouse cursor changes to a crosshair: place this where you want the (top of) the new object to be inserted in the report format and left click; Text, Expression, Image and Line objects can be dragged to the desired size. The new object will be inserted at this position (if permitted, and possibly moved a little so as not to overlap certain other elements).

Do not let objects overlap one another as this may cause unpredictable output.

ACREPORT has support for nomenclature mappings. Text objects will automatically map text at the time the report is run. The text is shown with its markers at design time. Expression objects do not map automatically, but have a property called **Map Nomenclature** which, if enabled, causes the result of the expression to be mapped at run-time.

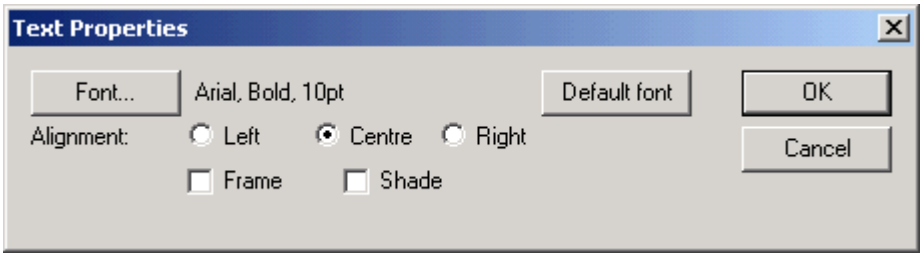
To manipulate objects see Editing ACReports

Text Objects

A Text object outputs fixed text into the report. For convenience, double-clicking a Text object lets you edit its text in place. A Text object can be multi-line if desired. You must ensure the object is wide enough and high enough to contain its text, otherwise it will be clipped in the output.

A Text object may be converted to an expression object by selecting **Convert to Expression** on the context (right click) menu. On conversion the text is used as the **Expression** property, the **Field** property is left blank, and all other properties specific to an expression are set to the default values for a new expression.

A Text objects' properties can be edited via **Properties** on the right-click or **Edit** menus.



The **Font...** property can be used to set an explicit font and colour for the text other than the default document font.

The **Default font** button resets it to use the default document font.

The **Alignment** property may be used to left, centre or right justify the text within the object's rectangle.

The **Frame** property creates a box around the area occupied by the object's rectangle.

The **Shade** property shades the background of the area occupied by the object's rectangle.

Expression Objects

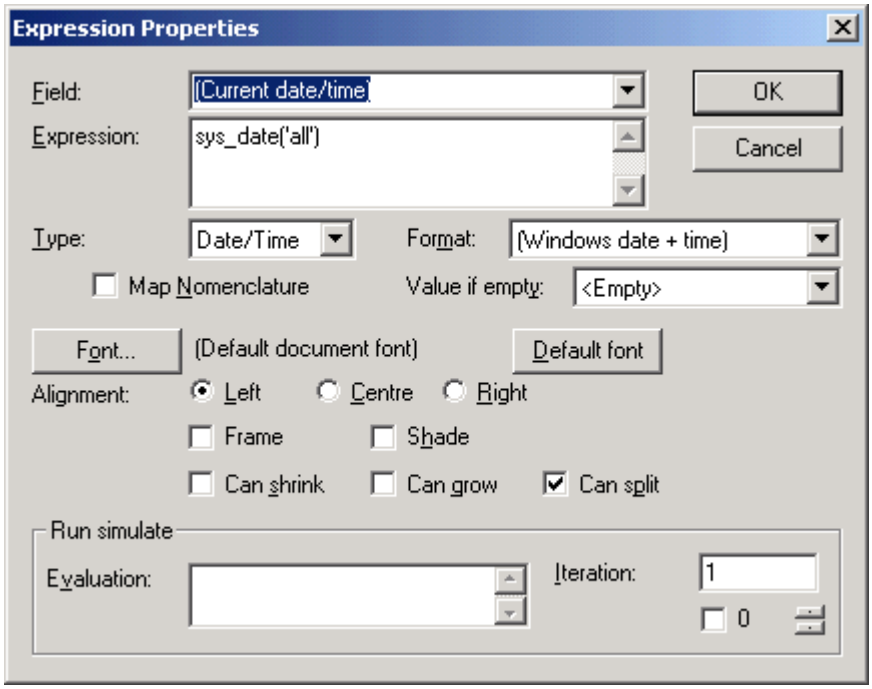
An Expression object causes an ACCELexpression to be evaluated and its textual result to be output into the report. An Expression object can be multi-line if desired.

The width of the object determines the width of the textual output when the report is run; if the width is exceeded further text will be wrapped to the next line or clipped.

The height of the object may or may not determine the height of the textual output, depending on the Expression object's properties. Note: unlike Text objects, the text you see on an Expression object in design view (which will be either a field name or some ACCEL code) has no significance for the final output; it does not matter whether it fits within the object's rectangle, since it will be the result of evaluating the expression that is actually output.

An Expression object may be converted to a text object by selecting **Convert to Text** on the context (right click) menu. On conversion the expression property is used as the **Text** property, and all other properties specific to an expression are ignored.

An expression objects' properties can be edited via **Properties** from right click or **Edit** menus, or by double-clicking on the object.



The **Field** property allows a *field'* to be selected whose content will be output in the report for the Expression object. The *field* may be genuine database fields, functions which return information which may be treated like a field, or certain special values. ACREPORT looks at where the Expression object appears in the report and offers choices according to the most appropriate enclosing subsection, to allow items pertinent to the enclosing section 's current item to be selected.

For example, if an Expression object is situated within a subsection whose database is CRs these would include CR Number, Summary, Text, Parts Affected etc.; choosing, say, Parts Affected would cause a list of the parts affected by the current CR it is reporting on to be output. Special entries include:

- **(Arbitrary Field Name)** use inside **Defined arbitrary field** sections
- **(Arbitrary Field Value)** use inside **Defined arbitrary field** sections
- **(Current Page Number)** use in **Page Header/Footer**
- **(Current date/time)** use in **Page/Body Header/Footer** to show the user that ran the report
- **(Current user)** use in **Page/Body Header/Footer**
- **(Current AllChange version)** use in **Page/Body Header/Footer**
- **( )project Current** use in **Page/Body Header/Footer** to show the AllChange project name
- **( )system directory Current** use in **Page/Body Header/Footer** to show the AllChange system directory

Choosing a field causes **Expression**, and possibly **Type** and **Format**, to be populated.

The **Expression** property contains the actual ACCEL code that will be evaluated for the Expression object. This is populated correspondingly whenever a **Field** is selected. You may edit this directly to alter what is there or to insert any ACCEL expression desired which does not correspond to a *field*; if you do so the content of **Field** is erased.

The **Type** property allows the type of the return result from evaluating the expression to be specified, which in turn affects the choices available for **Format**. **Type** is set automatically whenever a **Field** is selected, but should be set manually if you type something into **Expression**. ACREPORT recognises 6 possible **Type** s:

- **Date/time**: the result is a date/time. **Format** offers a variety of fixed date/time formats (i.e. does not vary according to who runs the report) and Windows date/time formats (i.e. does vary according to the end-user's Windows settings).

- **List**: the result is a list. **Format** offers **Not Sorted** (i.e. whatever order the list was generated in) or **Sorted**.
- **Truth**: the result is a truth. **Format** offers various ways to display the result: **Yes** or **No**, **True** or **False**, the name of the field or **Not** followed by the name of the field (e.g. if the field is named **Obsolete**, this would result in **Obsolete** or **Not Obsolete**), the name of the field or nothing.
- **Text**: the result is anything else. **Format** is disabled; the text is output as generated.
- **User**: the result is a user logon id. **Format** is disabled. The result is mapped from the logon id to the full username as defined in the User Registrations
- **User List**: the result is a list of user logon IDs. **Format** is disabled. The result is mapped, for each logon id, from the logon id to the full username as defined in the User Registrations

Map Nomenclature causes the result of Expression objects to be mapped at run-time based on the nomenclature selected.

Value if empty is available for all **Type**s other than **Truth**. It offers choices as to what to display if the result is empty: **<Empty>** means leave it empty, otherwise **None** or **Unallocated** may be output.

Font and **Alignment** set the font, colour and alignment of the text output, as for Text objects. Remember this applies to the text produced when the report is run; in design view the Expression Object will always be the **Field** (or the **Expression** if that is empty).

The **Frame** property creates a box around the area occupied by the object's rectangle.

The **Shade** property shades the background of the area occupied by the object's rectangle.

Can shrink and **Can grow** stipulate what can happen when the Expression object is evaluated during the report run and the resulting text is smaller or larger vertically than the object's rectangle. If it is smaller and **Can shrink** is checked then the rectangle is effectively reduced to the size of the text and subsequent elements are moved up so that no gap is left; if it is not checked then the text still takes up the same amount of space as the object does, subsequent elements are not moved, and there will be a gap following it. (An Expression object which evaluates to the empty string and has **Can shrink** checked takes up no room in the output; this can be used to insert arbitrary ACCEL expressions/calculations or comments into a report.) If the text is large and **Can grow** is checked then the rectangle is effectively increased to the size of the text and subsequent elements are moved down; if it is not checked then the text still takes up the same amount of space as the object does, subsequent elements are not moved, and the text will be truncated to fit, losing later lines.

Can split determines what can happen if the evaluated text would spill across to the next page from where it starts in the output: if it is checked the text starts wherever it is in the output and gets split across the page break; if it is not checked a fresh page is started before outputting the text (unless the text exceeds a whole page anyway). These properties are usually only relevant if the output is multi-line, such as for a list or the CR Text.

The settings of **Can shrink**, **Can grow** and **Can split** are only treated as *requests*. There are complex situations where ACREPORT cannot adhere to these requests, and may ignore them. For example, they are ignored on objects inside tables, and if an expression with **Can shrink** or **Can grow** properties is vertically overlapped (e.g. they are side-by-side) by another expression with **Can shrink** or **Can grow** properties the second expression's settings are ignored. Basically, if you only set **Can shrink** or **Can grow**, or clear **Can split**, on objects which have nothing (or no more than a simple Text object label) on either side of them the requests should be obeyed.

The **Run simulate** properties allow you to control what the Expression object evaluates to if simulating a report run. For **Evaluation** enter the text you want to see; if this is empty ACREPORT will output **XXX**. **Iteration** produces the specified number of copies of **Evaluation**, or a random number from 0 up to that number; this is useful for testing voluminous output. If **Evaluation** had a terminating newline this is included in the result; if **Iteration** is producing multiple copies the newline is included between each line (to produce multi-line output), otherwise a space character is used.

Image Objects

An Image Object allows images from files to be inserted into the report output. This allows reports to include company logos, diagrams etc. The images can be obtained either directly from a file, or from a file path returned by an ACCEL expression.

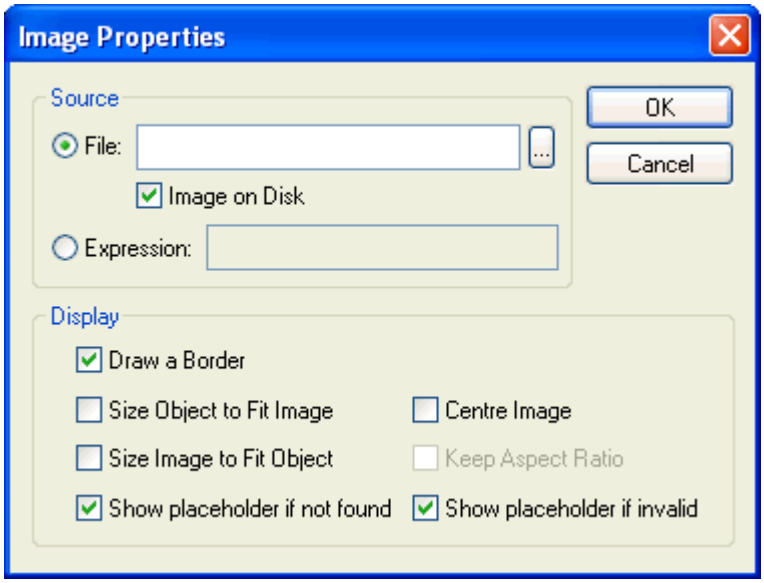
The image can be in one of the following formats:

- Windows Bitmap (*.bmp)
- Windows Meta File (*.wmf;*.emf)
- JPEG (*.jpg;*.jpeg)
- CompuServe Graphics Interchange (*.gif)
- Tagged Image File Format (*.tif;*.tiff)
- Portable Network Graphics (*.png)
- Zsoft Paintbrush (*.pcx)

When running a report, any images which are too tall to fit on the remainder of a page will force a page break, and the image will appear on the next page. Images which are too tall to fit on one page, do not force a page break, but are split across pages as required. Images which are too wide for a page will be drawn up to the edge of the page, with any overhanging parts of the image not appearing in the report output.

Files specified which are not images are not displayed; it is not possible for ACREPORT to convert a non-image file, e.g. a Word document, to an image.

An image objects' properties can be edited via **Properties** from right click or **Edit** menus, or by double-clicking on the object.



If **File** is selected then the file containing the image to be shown must be specified. If **Image on Disk** is checked then the file will be read from the disk on opening or running the report. If **Image on Disk** is not checked then the image will be embedded into the report. If the file is to be read from the disk, then care should be taken that the file is accessible from wherever the report is expected to be run.

If **Expression** is selected then the ACCEL expression must be supplied to return a file path specifying the image to use. The exact expression will depend on the context in which the image is to be calculated. The file whose path is returned should exist for the life of the report. The file can be a temporary file, into which the image is copied, and can then be deleted once the report is complete, or on exiting ***AllChange***. The

ACCEL expression could, for example, call a function to enable a CR or Baseline file attachment thumbnails to be included in a report. See File Attachment Example function for an example of this.

Image objects may have the following display options applied:

- **Draw a Border**: draws a border around the edge of the object's rectangle
- **Size Object to Fit Image**: causes the object's rectangle to be set to the size of the image. With this option set, the object may not be resized. This option is not available if the image source is from an ACCEL expression. Also, this option has no effect if the image is used in a table.
- **Size Image to Fit Object**: causes the image to be sized to fit the object's rectangle.
- **Keep Aspect Ratio**: ensures that the resized image retains its original aspect ratio. This is only available with the **Size Image to Fit Object** option
- **Centre Image**: ensures that the image is centered within the object's rectangle. Without this option, the image appears at the top-left of the object. This is available either when the **Keep Aspect Ratio** option is used, or when neither the **Size Object to Fit Image** nor the **Size Image to Fit Object** are used.
- **Show placeholder if not found**: determines whether a box is drawn containing the text "Image not found" if the image path is not found at the time the report is run.
- **Show placeholder if invalid**: determines whether a box is drawn containing the text "Invalid image" if the image path points to a file which is not an image, or is corrupt.

File Attachment Example

An example ACCEL function is shown below, which returns a path based on a file attachment of a CR or Baseline. This could be used in a CR or Baseline Files Affected report or table to add thumbnails of images to the report.

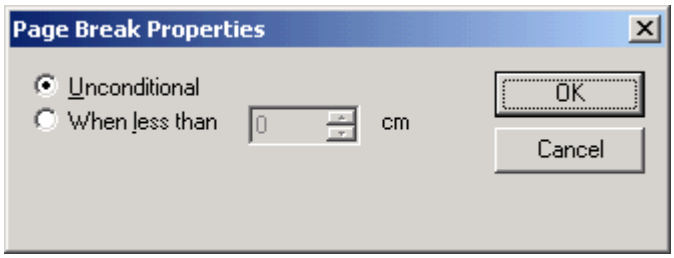
```
GetFileAffectedImage
# return the full path of a file attachment
# if the attachment is a link, then return the path directly
# else copy the attached file to a temp file, which we set to delete later
# p1 = database (cr or baseline)
# p2 = attached file
action
  local(db, dir, dir2, file, file2, link, attachname, thing);
  setvar(db, var(p1));
  setvar(file, var(p2));
traperror
  setvar(link, call(is_file_link, var(file)));
  if not var(link) then
    if var(db) == 'cr' then
      setvar(attachname, 'crattach');
      setvar(thing, cr_number);
    elsif var(db) == 'baseline' then
      setvar(attachname, 'blattach');
      setvar(thing, bl_name);
    else
      return '';
    endif;
    setvar(dir, join_server_paths(acserverprojdir, var(attachname)));
    setvar(dir2, join_server_paths(var(dir), attachment_dir-
name(var(thing))));
    setvar(file2, join_server_paths(var(dir2), lowercase(plain_file-
name(var(file)))));
    setvar(file, temp_filename(tmpdir, "acrimg", file_suffix(var(file))));
    appendlistvar(Global_DeleteFilesOnExit, var(file));
    getserverfile(var(file2), var(file));
```

```
  endif;
onerror
  setvar(file, '');
enderror;
  return unquote(var(file));
end
```

Page Break Objects

A Page Break object causes output to move to start of the next page when it is encountered during the report run. Page Break objects have a fixed height and extend the full width of the page. The space occupied by a Page Break object in design view is removed from the output. Page Break objects must not overlap other objects; an error will be raised if you try to run a report in such a situation.

Use a Page Break's **Properties** to choose between two different flavours.



Unconditional means that a new page is started no matter how far down the current page the output has currently reached; this is often used as the last element in the top-level **Body** of a report so that each item reported on starts on a fresh page.

Use **When less than** and a value (in centimetres) to start a new page if the distance remaining on the current page of the output is less than a certain amount; this may be used just before an element which may produce several lines of output (such as a table or multi-line expression) so that it does not end up outputting a small amount just before flowing onto the next page.

Line Objects

Line objects can be used as separators within reports. The width and height of the line can be set by resizing the object.

(Another possible way to insert spacing into a report is to add an empty text object the full width of the page).

Printing/Previewing Reports

Select **File|Print...** to send output to a printer or **File|Print Preview...** to view the output in the ACREPORT print previewer from the ACREPORT menu. In both cases you will first be presented with the **Print/Preview** dialog offering three options:

Run
: This actually runs the report, reading and outputting data from the ***AllChange*** database. It is what end-users will do when they run the report. Running reports is discussed in **Running reports**.

Simulate
: This simulates running the report. It does not read or output actual data from the ***AllChange*** database, but instead produces dummy output; this results in output which looks similar to what would happen if the report were actually run. It is useful in several situations:

- You want to see what the output would look like, but do not want to spend the time involved in actually running the report.
- You want to see what the output would look like, but do not have any actual useful data.
- You want to see what the output from an area of the report is but do not presently have any data which would cause that area to be executed (for example, you have a conditional section which would not be executed given your current data).

Design

This outputs the contents of the format file itself, i.e. the objects sections etc. you see in the design view.

Running reports

There are two ways to run an ACReport within ACE:

1. **Go via a standard ACE** report dialog, selecting the **ACReport** as the format file.
2. Use the **Reports** menu to open the desired ACREPORT format file, then **File|Print...** or **File|Print Preview...** from the ACREPORT menu.

Note that the output from an ACREPORT may only be previewed or printed. Unlike text reports, output *cannot* be sent, for example, to Word for further alteration. ACREPORT is designed so that its output should not require such post processing.

Using method 2, when you select Reports|Open... you will be presented with a dialog for opening an existing ACREPORT format file. Use the ... button to select from a list of the ACREPORT format files known to ACE ; these may be located in any of the user's home directory, the AllChange project directory or the AllChange system directory , and if running client/server they will be read client/server if located in the project or system directory. Alternatively, you may type a filename into the edit control directly; a plain name (no path) will still be sought as above, but if it has a path it will only be sought locally.

You may then choose **File|Print...** to send output to a printer or **File|Print Preview...** to view the output in the ACREPORT print previewer. In both cases you will first be presented with the Print/Preview dialog: ensure the Runto alter the printer, paper orientation etc., but remember the report will have been designed with a specific paper size in mind and so may not adapt very well to a change in orientation, for example. **File|Page Setup...** is predetermined by the report format file. Before printing/previewing you may use Database (i.e. item(s) to be reported on) if desired, otherwise all items will be included in the report. The Index Value or Index ,Condition tab to specify any Run Arguments tab. You may go to the General radio button is selected on the

When previewing an ACREPORT run the report is running in the background with further pages being produced on demand as you choose **Next Page** to move on (otherwise you might have to wait a long time to receive any output if you had to wait for the report to run to completion). There may be a slight delay when moving to a new page if many records have to be visited. The number of pages you can go back by via Prev Page is limited by a buffer size (currently 10 pages). When you choose **Close** to exit the previewer the report run terminates. The output seen in the previewer is only an approximation of what will appear on the printer if the report output is printed; in particular, fonts are likely to differ somewhat.

When printing an ACREPORT run output is sent to the printer immediately as it is produced.

Printing Reports to PDF Files

A report may be printed to a PDF file instead of to a printer using a PDF Printer Driver. A number of PDF printer drivers are available free of charge from the internet.

The following have been tested for use with AllChange:

1. **PDF Creator**: the home page is http://sector7g.wurzel6.de/pdfcreator/index_en.htm, but the actual driver is downloadable directly from https://sourceforge.net/projects/pdfcreator.
2. **cutePDF**: download from http://www.acrosoftware.com/Download.htm. For details see http://www.acrosoftware.com/products/cutepdf/default.asp for details.

Commands

Add Baseline

Function

Accessed from **Baseline | Add...** or via the toolbar. Allows a baseline to be taken.

Class

Baseline

Options

[Header]

Basename

Specifies the base name to be given to the baseline. If the baseline is not to have versions, then this should be the entire baseline name and must be unique. If the baseline is to have versions, then this may be the same as an existing basename (in which case a new version of the basename will be created) or a new basename for which the first version is to be created.

Version

Specifies the version identifier to be given to the baseline. If specified then the baseline will be a *versioned* baseline, but may be left blank if no versioning is required.

Type

Specifies whether the baseline is a **Design Baseline**, a **Release Baseline** or an **Instance Baseline** . Also specifies whether it is a **Meta Baseline**

Class

Allows the class for the baseline to be specified. The class may be any of the classes defined for a baseline and is used to determine the baseline's life-cycle .

Header Only

Specifies that the baseline is not to contain any items — i.e. it is an empty baseline.

Lock Baseline

Allows the baseline to be locked against change

Rel Date

Allows the baseline release date to be specified.

Top Part

Specifies the area of the parts tree that the baseline relates to. Parts outside the **Top Part** may not be included in the baseline.

Comment

Allows a comment for the baseline to be specified

Arb1...Arb40

Allows information to be assigned to the arbitrary fields. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

`status =`*status*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the status of the baseline to be set to *status* with no conditions or actions being invoked.

`user=`*user*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the baseline to be set to *user* instead of the user id of the user performing the operation.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the baseline was created to be set to *date* instead of the current system date.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the baseline to be taken from the filename specified.

[Text]

Specifies the actual text of the baseline. The ***AllChange*** Administrator will probably have set the system to use a template for the baseline text.

[Parts]

Parts
Specifies the subsystems from which to find parts to be included in the baseline. Each part and all its children are candidates to be included in the baseline.

Versions
Specifies which versions (if any) should be included in the baseline:

All
Causes all versions of a component type part to be baselined.

Default
Causes only the default version of a component type part to be baselined.

Registered
Causes only the registered or top version of a component type part to be baselined, ignoring any checked out version.

Top
Causes only the top version of a component type part to be baselined, ignoring any checked out or registered version.

Instances
Specifies which instances should be included in the baseline. Only valid for Instance baselines.

All
All instances should be included in the baseline

Top
Only the top (latest) instance is included in the baseline

Follow Uses
Causes uses type parts to be followed to the part used

Lock Parts
Specifies that each part added to the baseline will be *locked*. This is not applicable for instance baselines

Condition
Allows a condition to be specified for the parts to be baselined: any valid condition in ACCELmay be given, see ACCEL. The part being baselined is the current record of the parts database, so plain field

references for the current part should be used. In ACC this must be the last argument on the command line; the condition extends to the end of the line and should *not* be quoted.

[Baselines]

Baselines
Specifies the names of the baselines to be included in the meta baseline

Related Commands

Delete Baseline, **Rename Baseline**, **Alter Baseline**, **Change Baseline Status**, **Baseline Text**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Add Baseline allows a baseline to be taken which logs information about each item baselined. Baselining provides a mechanism to take a snapshot of a system or subsystem at a point in time. Baselines may contain parts (**Release** or **Design** baselines), instances (**Instance** Baseline), or may contain baselines (**Meta** baselines). A **Release baseline** includes only versions of components whilst a **Design baseline** includes no versions and may include the product structure (subsystems) as well as any required components. An **Instance baseline** may include only instances of versions.

Each baseline is given a name which is comprised of:

<basename>[;*<version>*]

The *version* is optional if specified the baseline is a *versioned* baseline.

Baseline versions may be used to identify, for example, different releases (*versions*) of a product and may be used to limit the number of versions of baselines viewed, reported on etc. A baseline version may be any string, ***AllChange*** does not put any interpretation on the string.

Various information about the baseline is stored, such as the user who took the baseline, the comment associated with the baseline, the date etc. Each part that is included in a part baseline is logged with information such as the full partname, the status of the part etc.

The actual text of the baseline, which may include any other fields of information required together with a full description of the baseline, may be edited, retrieved and stored — see Baseline Text.

Rel Date (release date) is used to determine the ordering of baseline versions. Since no interpretation is placed on the version identifier, this is used instead to determine which is the *latest* baseline version. The release date should therefore reflect the expected release date for the baseline, this may be modified at a later date. If a release date is not specified then the date that the baseline is created is the default value.

On creating a baseline, if the baseline's class has the **Target Release** field defined, then the field's value is set to itself, unless a value is specified on the command-line. Similarly, if the **Pred Baseline** field is defined for the class, its value is set to "(Not Applicable)", if no value is specified on the command-line.

If the **Lock Parts** option is used then taking a baseline *locks* each part. When a baseline locks a part, a count is incremented in the locked field of the part. When a part is removed from a baseline its lock count is decremented. A part is treated as locked so long as the lock count is greater than 0. This means that a part will remain locked, so long as it is included in any locking baseline. Locked parts/versions may not be deleted. This therefore ensures that a version may not be deleted whilst it is contained within a locking baseline. Note the Lock Parts is not applicable to instance baselines.

For a release baseline if **All** is selected for the **Versions** then *all* versions of a component type part are included in the baseline. This provides a detailed snapshot of the state of a component and all its versions at a point in time; however, it takes up a lot of space and does not identify any particular version as being of interest. The **Default**, **Registered** or **Top** version options cause the baseline to contain just a single version of each component; such a baseline thus identifies a particular version as being of interest and uses a

modest amount of space. The **Default** option causes the default version of a component to be included in the baseline. As explained in Default Versions, when attached to a workspace the default version is affected by any check-outs and/ or registrations in that workspace, which is usually what is wanted. When taking a baseline, however, this behaviour is often not desirable. The **Registered** option will ignore any check-outs but take into account any registrations when determining the version to include in the baseline. The **Top** will take the top version only, ignoring both check-outs and registrations.

For an instance baseline if **All** is selected for the **Instances** then all instances of the versions selected are included in the baseline. If **Top** is selected for the **Instances** then the top or latest instance of each version selected is included in the baseline.

The **Condition** option may be used to further limit what parts and/or version(s)and/or instances are included in a baseline.

Any baseline detail items which are obsolete will *not* be added to the baseline unless the Show Obsolete flag is set.

If **HeaderOnly** is selected then an empty baseline will be created. This may be useful in conjunction with the **Alter Baseline** command to implement an incremental baseline (see Alter Baseline).

If a class is specified then this determines the life-cycle of the baseline. If a life-cycle is associated with the specified class then the status of the baseline will be set to the initial status of the life-cycle. The status of a baseline may be changed by the **Change Baseline Status** command.

The **Alter Baseline** command may be used to alter baseline header information. It may also be used to add, delete and update baseline details.

Information about what baselines have been taken, and details about each baseline may be obtained using the baseline browser and viewer.

Baselines may be deleted by the **Delete Baseline** command.

The default version used when a part is referenced may be set to the version included in a baseline (see Workspace Registrations); a baseline may also be used in a specific version reference for a part (see Part Versions). For a baseline to be used for this purpose it must contain precisely one version of each component, e.g. as produced by one of the **Default**, **Registered** or **Top** options. A baseline may also be used in generating a list of parts/ versions included in a baseline (see Part Paths).

Command Line Syntax

baseline [`-all` | `-def` | `-reg` | `-top` | `-none`] [`-meta`] [`-uses`] [`-design`] [`-instance` [`-topinstance`|`-allinstances`]] [`-lockparts`] [`comment=`***comment***] [`locked=on`] [`class=`***class***] [`toppart=`***toppart***] [[`-linkfiles`] `filesaffected=`***file-list***] [`blinesaffected=`***baseline-list***] [***arb1***...***arb40***`=`***value***] [`text=`***filename***] `detail=`***parts*** | ***baselines*** | `none` [`status=`***status***] [`user=`***user***] [`date=`***date***] [`releasedate=`***date***] [`-blversion` ***version-idbasename*** [`-cond` ***condition***]

Examples

```
baseline -def -lockparts -blversion 1 detail=/product b1
```

takes a (release) part baseline with a basename of `b1` and a version of `1` — full name `b1;1` — containing the default version of all components in the parts tree starting from `/product` downward, incrementing the lock count on each item as it does so (which requires the user to have update permission for every item).

```
baseline -design detail=/ snapshot
```

takes a design (un versioned) part baseline — called `snapshot` — of all subsystems and components (but not versions) in the parts tree starting from `/` downward (i.e. a snapshot of all non-version parts in the database). Lock counts are not incremented, and so the user does not require update permission for any items.

```
baseline -blversion 1 -reg -lockparts 'comment=First release'
      locked=on 'detail=/Product1 /Product2' release
```

takes a (release) baseline — called `release_1` — of all registered versions of the parts tree hierarchies rooted at `/Product1` and `Product2`. The baseline has a comment associated with it. (The quotes are necessary for the `detail` and `comment` in ACC — but not ACE — since the values contain a space). The baseline will be locked against change.

```
baseline -top -blversion 2 detail=/Product release -cond pa_status ==
      'Complete'
```

takes a (release) baseline of all top (i.e. default ignoring check-outs and registrations) versions only in the parts tree rooted at `/Product` whose current status is `Complete` (the quotes should be present whether using ACC or ACE since this is being passed to ACCEL as a literal string).

```
baseline -meta 'detail=Bline1 Bline2' MetaBase
```

takes a meta-baseline called `MetaBase` containing `Bline1` and `Bline2`.

```
baseline -meta 'detail=none' Meta2
```

creates an empty meta-baseline called `Meta2`.

```
baseline -instance -def -topinstance detail=/Car CarInstance;10
```

creates an instance baseline called CarInstance;10 containing the top instance of the default version of each component in the ?Car subsystem,

Add Part

Function

Accessed from **Part | Add...** or via the toolbar. Allows new parts to be declared to the system and entered into the parts database. Also allows existing files to be imported.

Class

Part

Options

[General]

Parts

Should specify the names of the parts to be added as a space separated list. The part name will be resolved to a full path name (if not specified as a full path name), and both the name and the parent of the part are determined from this. Any number of parts may be specified and each will be created with the other arguments specified.

If existing files are to be imported into the ***AllChange*** database, then the file browser may be used to select the files from the current workspace and these will be translated into partnames and entered as the **Parts** on invoking the **Add Part** dialog.

Type

This *must* be either `component` or `subsystem`.

Class

Gives the class of the part. This is optional and may be any of the defined classes for the type of part. The class determines:

- the life-cycle for the part if there is to be one.
- the template to be used for initial versions of components (if there is one)
- whether component parts are to be kept check out read only
- whether component parts require a CR to authorise a change

Arb1...Arb40

Allows information to be assigned to the arbitrary fields. If used these should have been assigned

meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

Flags
Allows the **Not version controlled** and **No file** flag to be set for the part, see Part Flags. The flags only have significance to **component** type parts.

No Initial Version
Indicates that no initial version is to be created for a component. This argument may only be specified for parts of type **component**.

`-initial`
This argument is only available from the command line and is not presented as an option in ACE. It Indicates that there must be initial text. This will cause the contents of the file representing the part (which must be present in the current directory) to be stored as the initial contents of the first version of the part; an error will be raised if the file does not exist. This argument may only be specified for parts of type `component`.

`status =`*status*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the status of the part added to be set to *status* with no conditions or actions being invoked, instead of the initial status in the life-cycle being used.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the part was created to be set to *date* instead of the current system date.

`-recursive`
This argument is only available from the command line and is not presented as an option in ACE. This option causes any non-existent parent subsystems to be created recursively before creating the specified part(s). Note that these subsystems are created with empty fields except that if the primary part being created is a subsystem and a class is specified then any parent subsystems created will be given the same class.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the part to be taken from the filename specified.

[Text]

Specifies the actual text of the part. The ***AllChange*** Administrator will probably have set the system to use a template for the part text.

[1st Version]

This tab will only be present when adding a component and allows values to be specified for the first version of the component.

Baseline
If a baseline is specified then the 1st version of the new component will be added to the baseline. The baseline specified should be a *release* baseline.

Varb1...Varb40
Allows information to be assigned to the arbitrary fields for the 1st version of the component. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator.

`ver_text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the version to be taken from the filename specified.

[Text]

Specifies the actual text of the version. The ***AllChange*** Administrator will probably have set the system to use a template for the version text.

[Items Affected]

This tab allows the parts which are affected by the new parts to be specified.

Related Commands

Delete Part, **Alter Part**, **Rename Part**, **Use Part**, **Copy Part**, **New Version**, **Alter Version**, **Part Text**.

Concepts

Parts (Creating and Managing Parts).

Description

Add Part allows component and subsystem type parts to be added to the parts database. (Parts of type **uses**may be added to the parts database by use of the **Use Part** dialog.)

The options allow the fields of the part to be specified.

Note that once a flag has been set on a part, it may not be unset, although further flags may be added.

The part name must resolve to a unique full path to the part: multiple parts with the same full path name are not allowed and the parent of the part must already exist for the successful addition of the child.

The part is added into the parts database with the fields set according to the options specified. If the part added is of type **component** then an initial version of the part is also normally created (i.e. *partname*`;001` will be added into the database). If a workfile corresponding to the component already exists in the current directory it will be stored as the initial contents of the first version of the component; if no workfile exists and a template has been specified for this class then this is used otherwise the first version will be created empty.

If the **No Initial Version** option is used **component** type parts are created without any first version. No operating system commands to create the corresponding VC file will be executed. At a later date **New Version** may be used to create the component's first version and the VC file holding the first version. Note that while a component has no versions commands which require a version to operate upon (e.g. **Check In** , **Edit**) will error. This facility could be useful if it is desired to set up the parts hierarchy including components but the creation of corresponding operating system files is to be delayed until a later date.

If the part has a class which specifies a life-cycle for the part then the status of the part is set to the initial status of the life-cycle. This will also occur for the version of the part.

The actual text of the part, which may include any other fields of information required together with a full description of the part, may be edited, retrieved and stored — see Part Text.

If a baseline is specified then the first version of a new component will be added to the baseline.

Command Line Syntax

add `type=`*type* [`class=`*class*] [`location=`*location*] [`flags=no_file | no_vc`] [`-baseline` *baselinename*] [`partsaffected=`*part-list*] [*arb1*...*arb40*`=`*value*] [*varb1*...*varb40*`=`*value*] [`text=`*filename*] [`ver_text=`*filename*] [`-initial`] [`-noversion`] [`status=`*status*] [date=*date*] *parts*

Examples

```
add type=subsystem source
```

adds a new subsystem, called `source`, as a child of whatever the current working part is.

```
add type=component class=Csource -initial file1.c file2.c file3.c
```

adds three new components as children of the current working part, each of which has an initial version (`file1.c;001, file2.c;001, file3.c;001`). There should be three corresponding workfiles in the current directory (`file1.c`, `file2.c`, `file3.c`) to which the contents of the initial versions are set. The class of the parts and versions is set to `Csource` and their statuses are set to the first status of the life-cycle associated with class `Csource`.

```
add type=subsystem location=doc /product/documentation
```

adds a new subsystem whose location field is set to `doc`, meaning that the corresponding directory will be a sub-directory called `doc` of whatever the location of `/product` is.

Add Vote

Function

Accessed from **Vote | Cast Vote** or via the cycle viewer double click on a vote status. Allows a vote to be cast for which progression in the life cycle should be taken. Votes may be cast for Baselines, CRs and Parts.

Class

Baseline, CR and Part

Options

Status
The status of the item being voted on.

Item
The item being voted on. This may be a part, a CR or a baseline.

Vote
The status voted for. This may be one of the valid progressions in the life cycle or **(No Preference)** to indicate a vote but no preference as to which progression.

Role
The role/capacity in which the voter is voting.

Comment
A textual comment about the vote being cast

Arb1..Arb40
Allows information to be assigned to the arbitrary fields. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog

`user=`***user***
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the user who cast the vote to be set to *user* instead of the user id of the user performing the operation

`date=`***date***
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the date that the vote was cast to be set to *date* instead of the current system date.

Related Commands

Alter Vote, Delete Vote

Concepts

Voting

Description

Add Vote allows a vote to be cast for which progression in a life cycle should be taken. Votes may be cast for Part, Baselines and CRs according to vote definitions as set up by the ***AllChange*** administrator.

A vote of (No Preference) may be specified to indicate a vote, but that no preference as to which progression should be taken. This could be use, for example, to satisfy a vote being cast by a user who is specified as mandatory to vote.

If the vote is defined as a serial vote, then each voter may either vote for a status and terminate the vote at that point, or pass the vote on to the next voter, or declare the vote stopped or blocked. To pass the vote on a vote is cast of **(Next)** or similar as defined in the vote definition. To stop/block the vote a vote is cast of **(Stop)** or similar as defined in the vote definition.

Command Line Syntax

addbaselinevote| **addcrvote|** **addpartvote**	[date=<date>] [user=<user>] status=<status> vote=<vote> role=<role> [comment=<comment>] { <field>=<value> } <item>

Examples

```
addbaselinevote status=Created vote=Populated role=release_manager com-
      ment="I suggest we release this baseline" Baseline;3.2
```

Adds a new vote for Baseline;3.1 which is in the Created status with a vote for progressing the baseline to the Populated status representing the role release_manager.

```
addpartvote status=Review vote=Approved role=Reviewer /Pro-
      duct/Documentation/UserGuide.doc;003
```

Adds a new vote for /Product/Documentation/UserGuide.doc;003 in status Review with a vote for Approve with role Reviewer

```
addcrvote status=CAB vote=Approved role=CAB_member comment="I think
      this change needs to be made" RFC00010
```

Adds a new vote for RFC00010 in status CAB for Approve by a CAB_member specifying a comment that the change should be made.

Alter Baseline

Function

Accessed from **Baseline | Alter...** or via the toolbar. Allows the information held in a baseline to be altered.

Class

Baseline

Options

[Header]

Baseline

Specifies the names of the baselines to be altered. This should include both the basename and the version of the baseline.

TopPart

This allows the baseline's top part to be set.

Release Date
This allows the baselines release date to be set.

Comment
Allow a new comment to be specified.

Arb1...Arb40
The arbitrary fields of the baseline will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

Lock, Unlock
Allows the lock state of the baseline to be changed. If the lock state is changed then no other fields should be changed at the same time as the behaviour is undefined.

Command line only options

`status =`*status*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the status of the baseline to be set to *status* with no conditions or actions being invoked.

`user=`*user*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the baseline to be set to *user*.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the baseline was created to be set to *date*.

`-nocheck`
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows parts/ baselines which do not exist to be added to the baseline.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the baseline to be taken from the filename specified.

`detail=`*parts*|*baselines*
Allows the parts or baselines specified to be added to/updated in the baseline. It may be preceded by -delete to indicate that the items should be removed from the baseline

Dtarb1...Dtarb40
The arbitrary fields of the specified baseline details will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog. This may be preceded by -existingdetail. If this is specified then only details already in the baseline will have their arbitrary field updated any details specified which do not exist will cause an error - this option is only available from the command line

Related Commands

Add Baseline, **Update Baseline**, **Delete Baseline**, **Change Baseline Status**, **Check Out Part**, **Baseline Text**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Alter Baseline allows the header of an existing baseline to be altered.

For general fields of the baseline header, the contents of the field is replaced with the value specified.

Baselines may be *locked* when it is required to freeze the baseline against any further changes.

A locked baseline may not have any of the information about the baseline modified *except* its status which may be progressed through its life-cycle .

Alter Baseline is useful for amending multiple baselines if necessary.

Other fields may be modified using specialised commands/dialogs from **Baseline | Alter**.

The details of a baseline may be modified from the **Details** tab from the baseline viewer, or from **Baseline | Alter | Update**.

The relationship to other baselines and file attachments may be modified using the Relationships tab of the baseline viewer

Command Line Syntax

alterbaseline [`locked=on|off`] [`comment=`*comment*] [*arb1...arb40*`=`*value*] [`status=`*status*] [`user=`*user*] [`date=`*date*] [`releasedate=`*date*] [design=on|off][`top-part=`*toppart*] [`obsolete=on | off`] [`text=`*filename*] [[`-delete`] `detail=`*parts*/*baselines*] [[-existingdetail] dtarb1..dtarb40=*value*] [`class =` *class*] [`-recursive`] [`-def|-reg|-all|-top`] [`-allinstances|-topinstance`] [[`-deletefiles`] [`-linkfiles`] `filesaffected=`*file-list*] [[`-deleteblines`] `blinesaffected=`*bline-list*] [`-nocheck`] *baselines* [`-cond` *condition*]

Examples

```
alterbaseline 'comment=Test Baseline' b1 b2
```

alters baselines `b1` and `b2` to have the comment `Test Baseline` (the quotes are necessary in ACC — but not ACE — since the comment contains a space).

```
alterbaseline locked=on b2
```

sets the lock state of baseline `b2` to `on`. This means that the baseline is now locked against further change (i.e. it is frozen).

Alter Baseline Class

Function

Accessed from **Baseline | Alter | Alter Class** . Allows the class of a baseline to be changed.

Class

Baseline

Options

Baselines

Specifies the baselines to be changed.

Class

The class of the baseline will be changed to the new class specified. If the current statusis empty it

will be set to the initial status of the associated life-cycle(if there is one). Use `(None)` to set the part to have no class. Note that this may also change its life-cycle.

Related Commands

Alter Baseline, **Update Baseline Details**, **Alter Baseline Obsolescence**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Alter Baseline Class changes the class of the baselines specified.

Altering the class may also change the life-cycle for the baseline. Care should be taken if the life-cycle is to be changed as the baseline may be in a status which does not exist or has another meaning when the life-cycle changes. If the class of a baseline with no current class or status is altered (i.e. it is being given a class for the first time) then the status of the baseline is set to the initial status of the life-cycle associated if any. If the class of a baseline is changed and this does change the life-cycle and the baseline has a current status not present in the new life-cycle or with a different meaning, then behaviour may be unpredictable.

Command Line Syntax

alterbaseline [`class=`*newclass*] *baselines*

Alter Baseline Item Affected

Function

Accessed from **Baseline | View Baseline Item Affected**. Allows the arbitrary fields of an item affecting a baseline relationship to be viewed and modified.

Class

Part

Options

Arb1-Arb40

The arbitrary fields of the baselines' Item Affected relationship will be changed to the specified *value*s. If used, these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

`user=`*user*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the item affected relationship to be set to *user*.

`date=`*date*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the item affected relationship was created to be set to *date*.

Concepts

Baselines (Creating and Managing Baselines)

Description

Alter Baseline Item Affected allows the arbitrary fields of a Baseline relationship item to be modified/set. Select the relationship of interest in the Baseline Viewer or select a Baseline Items Affected relationship in the Baseline Items Affected Relationships browser and then use the context menu or Baseline Menu to view the details of the relationship and modify these details: remember to save your changes using **Misc | Update**.

Command Line Syntax

alterbaselineitemaffected [*arb1*...*arb40*=*value*] `itemaffected`=*item* `type=part|cr|baseline|file` [`-solved`]
[date=*date*] [user=*user*] *baselines*

Alter Baseline Obsolescence

Function

Accessed from **Baseline | Alter | Make Obsolete**. Allows the obsolete flag on a baseline to be changed.

Class

Baseline

Options

Baselines
Specifies the baselines to be changed.

Make Obsolete
Allows the **Obsolete** flag for the baseline to be set.

Clear Obsolete Flag
Allows the **Obsolete** flag for the baseline to be cleared.

Related Commands

Alter Baseline, **Update Baseline Details**, **Alter Baseline Class**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Alter Baseline Obsolescence allows the **Obsolete** flag of a baseline to be altered.

Command Line Syntax

alterbaseline [`obsolete=on | off`] *parts*

Alter Check-out

Function

Accessed from **Workspace | Alter Check-out...** or via the toolbar. Allows the fields of check-out logs to be altered.

Class

Workspace

Options

Parts
Specifies the part (s) whose check-outs are to be changed.

FromWorkspace
Specifies the workspace in which the parts are (currently) checked out.

ToWorkspace
Allows the workspace to which the part is checked out to be altered. This should also cause the work-file to be moved from the old workspace's area to the new workspace's area.

Comment
Allows the comment as to why the part is checked out to be altered.

User
Allows the user to whom the part is checked out to be altered.

For Edit
Allows a check-out for read only purposes to be turned into a check-out for edit. The user must be attached to the workspace to which the part is checked out. The workfile will be made writable and the check-outs database updated.

New Branch
This is only allowed with the **For Edit** option. Will create a new branch with the specified name and reserve a new version along that branch.

Optimistic
When used in combination with **For Edit** causes parts checked out for read only purposes to be turned into a check-out for edit using optimistic locking. The workfile for the specified *version* will taken out to the workspace unless the -noget option is specified.

`-noget`
This option is only available from the command line and is not presented as an option in ACE. This option is only valid with the **For Edit**,**version=<*version*>** or **Optimistic** options. If specified with the **For Edit** option, this option is ignored since it is implied by the behaviour of the **For Edit** option. If used with the **version=<*version*>** or **Optimistic** options, then -noget will prevent the specified version from being taken out to the workspace.

`version=<`*version*`>`
This option is only available from the command line and is not presented as an option in ACE. This option changes the version that is checked out to the specified *version*. If the check-out is a check-out for edit then the predecessor of the newversion associated with this check-out is also changed to the *version* specified. The workfile for the specified *version* will be taken out to the workspace unless the `-noget` option is specified.

Related Commands

Check In, **Check Out**, **Attach Workspace**.

Concepts

Part (Checking Files In and Out of ***AllChange***),

Description

Alter Check-out allows fields of check-out logs to be changed. The contents of the specified fields are replaced with the new values specified.

Specifying a subsystem among the parts whose check-outs are to be altered causes the **Alter Check-out** to descend the parts tree recursively downward from that point altering check-outs of the default version of all components encountered.

Parts must be checked out to the current workspace unless the **FromWorkspace** option is used to name which workspace they are checked out to.

Command Line Syntax

alterissue [`workspace=`*workspace*] [`edit=true` [`-branch` *branch*] [`-noget`] [`-optimistic`]] [`user=`*user*] [`comment=`*comment*] [`-inwspc` *workspace*] [`version=`*version*] *parts*

Examples

```
alterissue user=fred / component ;004
```

alters the check-out of `/component;004` — which must be checked out to the current workspace — to be checked out to `fred`.

```
alterissue edit=true /component;004
```

alters the check-out of `/component;004` — which must be checked out not for edit to the current workspace — to be checked out for edit.

```
alterissue workspace=newws -inwspc oldws /product
```

alters the check-outs of the default versions of all components anywhere within the `/product` hierarchy from being checked out to `oldws` to being checked out to `newws`, and moves the corresponding workfiles from `oldws`' area to `newws`' area.

Alter CR

Function

Accessed from **CR | Alter...** or via the toolbar. Allows the fields of CRs to be altered.

Class

CR

Options

[General]

CR Nums
Specifies the CRs whose fields are to be altered.

Top Part
Allows a new top part affected by the CR to be assigned. Use `none` to set the CR to have no top part affected.

Ref
Allows a new reference to be specified. Use `none` to set the CR to have no reference.

Summary
Allows the summary for the CR to be changed. Use `none` to set the CR to have no summary.

Arb1...Arb100
The arbitrary fields of the CRs will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the menu.

`status =`*status*
This argument is only available from the command line and is not presented as an option in ACE. This

argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the status of the CR to be set to *status* with no conditions or actions being invoked.

`originator=`*originator*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the originator of the CR to be set to *originator*.

`assignee=`*assignee*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the assignee of the CR to be set to *assignee*. Note that no actions associated with assigning a CR will take place (e.g. the assignee will not be emailed).

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the date that the CR was created to be set to *date* instead of the current system date.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the CR to be taken from the filename specified.

Related Commands

New CR, **Assign CR**, **Change CR Status**, **CR Text**.

Concepts

CRs (Creating and Managing Change Requests)

Description

Alter CR allows fields of CRs to be changed. The contents of the specified fields are replaced with the new values specified.

To alter the items related to a CR use the appropriate tab on the CR viewer.

The command line allows the items related to a CR to be modified by the altercr command - this is shown in the command line syntax below.

Command Line Syntax

altercr [`toppart=`*part*] [`ref=`*string*] [`class =`*class*] [`summary=`*string*] [*arb1...arb100*=*value*] [`assignee=`*assignee*] [`status=`*status*] [`originator=`*originator*] [`obsolete=on | off`] [`text=`*filename*] [date=*date*] [[`-deleteparts`] `partsaffected=`*part-list*] [[`-deleteparts`] `versionssolved=`*version-list*] [[`-deleteblines`] `blinesaffected=`*baseline-list*] [[`-deleteblines`] `blinessolved=`*baseline-list*] [[`-deletecrs`] `crsaffected=`*cr-list*] [[`-deletefiles`] [`-linkfiles`] `file-saffected=`*file-list*] *cr-nums*

Examples

```
altercr 'summary=Program crashes' toppart=/product CR00012
```

alters the summary and toppart fields of CR number CR00012. (The quotes are necessary for the summary in ACC — but not ACE — since the string contains a space.)

```
altercr -deleteparts partsaffected=file.c CR00012 CR00023
```

removes *current-working-part*`/file.c` from the list of parts affected by CRs CR00012 and CR00023.

```
altercr 'versionssolved=module1.c module2.c;003' CR00012
```

adds the default version of `module1.c` and version 3 of `module2.c` to the list of versions solving CR CR00012.

```
altercr versionssolved=all blinessolved=cr12bline CR00012
```

adds the default version of each part affected by CR CR00012 as the version solving it and adds the baseline `c12bline` as a baseline solving CR CR00012.

Alter CR Class

Function

Accessed from **CR | Alter | Alter Class** . Allows the class of a CR to be changed.

Class

CR

Options

CR Nums
: Specifies the CRs to be changed.

Class
: The class of the CR will be changed to the new class specified. If the current status is empty it will be set to the initial status of the associated life-cycle (if there is one). Use `(None)` to set the part to have no class. Note that this may also change its life-cycle.

Related Commands

Alter CR, **Alter CR Obsolescence**.

Concepts

CRs (Creating and Managing Change Requests)

Description

Alter CR Class changes the class of the CRs specified.

Altering the class may also change the life-cycle for the CR. Care should be taken if the life-cycle is to be changed as the CR may be in a status which does not exist or has another meaning when the life-cycle changes. If the class of a CR with no current class or status is altered (i.e. it is being given a class for the first time) then the status of the CR is set to the initial status of the life-cycle associated if any. If the class of a CR is changed and this does change the life-cycle and the CR has a current status not present in the new life-cycle or with a different meaning, then behaviour may be unpredictable.

Command Line Syntax

altercr [`class=`*newclass*] *CRs*

Alter CR Item Affected

Function

Accessed from **CR | View CR Item Affected**. Allows the arbitrary fields of an item affecting a CR relationship to be viewed and modified.

Class

CR

Options

Arb1-Arb40
The arbitrary fields of the CRs' Item Affected relationship will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the menu.

`user=`*user*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the item affected relationship to be set to *user*.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the item affected relationship was created to be set to *date*.

Concepts

CRs (Creating and Managing CRs)

Description

Alter CR Item Affected allows the arbitrary fields of a CR relationship item to be modified/set. Select the relationship of interest in the CR Viewer , or select a CR Items Affected relationship in the CR Items Affected Relationships browser and then use the context menu or CR Menu to view the details of the relationship and modify these details: remember to save your changes using **Misc | Update**.

Command Line Syntax

altercritemaffected [*arb1...arb40*=*value*] `itemaffected`=*item* `type=part|cr|baseline|file` [`-solved`] [date=*date*] [user=*user*] *cr-nums*

Alter CR Obsolescence

Function

Accessed from **CR | Alter | Make Obsolete**. Allows the obsolete flag on a CR to be changed.

Class

CR

Options

CR Nums
Specifies the CRs to be changed.

Make Obsolete
Allows the **Obsolete** flag for the CR to be set.

Clear Obsolete Flag
Allows the **Obsolete** flag for the CR to be cleared.

Related Commands

Alter CR, Alter CR Class.

Concepts

CRs (Creating and Managing Change Requests)

Description

Alter CR Obsolescence allows the **Obsolete** flag of a CR to be altered.

Command Line Syntax

altercr [`obsolete=on | off`] *CRs*

Alter Instance

Function

Accessed from **Part | Instance | Alter Instance**. Allows information about instances to be changed.

Class

Part

Options

Instance
Specifies the instance whose fields are to be changed

Arb1...Arb40
Allows information to be assigned to the arbitrary fields. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the date that the instance was created to be set to *date*.

`user=`*user*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the user who created the instance to be set to *user*.

Related Commands

Delete Instance, New Instance

Concepts

Parts (Creating and Managing Parts)

Description

Alter Instance allows the arbitrary fields for instances to be changed.

Command Line Syntax

alterinstance [*arb1..arb40*=*value*] [`date=`*date*] [`user=`*user*] [`obsolete=on|off`] *instances*

Examples

```
alterinstance Serial_No="12B456E"
      "/Car/Accessories/CD Player;002:0001"
```

changes the Serial Number arbitrary field to 12B456E for instance /Car/Accessories/CD Player;002:001

Alter Instance Obsolescence

Function

Accessed from **Part | Instance | Make Obsolete** or via the toolbar. Allows the obsolete flag on an instance to be changed.

Class

Part

Options

Instances
Specifies the instances to be changed.

Make Obsolete
Allows the **Obsolete** flag for the instance to be set.

Clear Obsolete Flag
Allows the **Obsolete** flag for the instance to be cleared.

Related Commands

New Instance, Alter Instance, Delete Instance.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Instance Obsolescence allows the obsolete flag on an instance to be changed.

Command Line Syntax

alterinstance `obsolete=on|off`

Examples

```
alterinstance obsolete=on "/Car/Accessories/CD Player;002:0001"
```

Marks /Car/Accessories/CD Player;002:0001 as obsolete

Alter Part

Function

Accessed from **Part | Alter...** or via the toolbar. Allows the fields of the specified parts to be changed.

Class

Part

Options

Parts
The parts whose fields are to be altered.

Arb1...Arb40
The arbitrary fields of the part will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

`status=`*status*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the status of the part to be set to *status* with no conditions or actions being invoked.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the part was created to be set to *date*.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the part (component) to be taken from the filename specified.

`ver_text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the version to be taken from the filename specified.

Related Commands

Change Part Status, **Rename Part**, **Alter Version**, **Part Text**.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Part allows the fields of one or more parts to be changed. Only subsystem and component type parts may be altered.

This allows the arbitrary fields to be modified on multiple parts in one go.

Other fields of a part may be altered by specialised commands.

To alter the parts affected by a part use the appropriate tab on the Part viewer.

The command line allows the parts affected by a part to be modified by the alter command - this is shown in the command line syntax below.

Command Line Syntax

alter [`class=`*newclass*] [`location=`*newlocation*] [`flags=no_file | no_vc`] [[`-delete-parts`] `partsaffected=`*part-list*] [*arb1...arb40*=*value*] [`obsolete=on | off`] [`text=`*filename*] [`ver_text=`*filename*] [`status=`*status*] [date=*date*]*parts*

Examples

```
alter location=newloc /product/part
```

alters the location field of `/product/part` to `newloc`, meaning that the corresponding operating system location will be a file/ directory named `newloc` in whatever the location of `/product` is. If `/product/part` is of type component the corresponding file's location is altered; if it is of type subsystem then not only is the corresponding directory's location altered but so, consequently, are the locations of descendants of `/product/part` (unless they have an absolute location).

Alter Part Class

Function

Accessed from **Part | Alter | Alter Class** . Allows the class of a part to be changed.

Class

Part

Options

Parts
Specifies the parts to be changed.

Class
The class of the part will be changed to the new class specified. Use `None` to set the part to have no class. Note that this may also change its life-cycle .

Related Commands

Alter Part, **Alter Part Obsolescence**, **Alter Part Flags**.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Part Class changes the class of the parts specified.

Altering the class may also change the life-cycle for the part. Care should be taken if the life-cycle is to be changed as the part or its versions may be in a status which does not exist or has another meaning when the life-cycle changes. If the class of a component with no current class or status is altered (i.e. it is being given a class for the first time) then the status of the component plus that of all of its versions will be set to the initial status. If the class of a part is changed and this does change the life-cycle and the part or its versions have a current status not present in the new life-cycle or with a different meaning, then behaviour may be unpredictable.

Command Line Syntax

alter [`class=`*newclass*] *parts*

Alter Part Flags

Function

Accessed from **Part | Alter | Alter Flags**. Allows the flags on parts to be modified.

Class

Part

Options

Parts
Specifies the parts to be changed.

Not Under Version Control
Allows the **Not Under Version Control** flag to be set, see Part Flags.

No File
Allows the **No File** flag to be set, see Part Flags.

Related Commands

Alter Part, Alter Part Obsolescence, Alter Part Class.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Part Flags allows an additional flag to be added to a part . The flags only have significance to **component**type parts.

The flags that may be set on a part include **No File** and **Not Under Version Control** — see Part Flags. Care should be taken when adding an additional flag to a part as it may cause unexpected behaviour.

If the **No File** flag is set on a component type part after the part has been created and versions of the part have been created then you will have a partial version history of the contents of the part which is no longer accessible. All future versions will not add to the version history of the contents. If the **Not Under Version Control** flag is set on a component type part after some versions have been created, then there is an inconsistency in that you have a part that has some versions but also has the **Not Under Version Control** flag set.

Command Line Syntax

alter [flags=no_file | no_vc]

Alter Part Item Affected

Function

Accessed from **Part | View Part Item Affected**. Allows the arbitrary fields of an item affecting a Part relationship to be viewed and modified.

Class

Baseline

Options

Arb1-Arb40

The arbitrary fields of the Parts' Item Affected relationship will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the menu.

`user=`*user*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the item affected relationship to be set to *user*.

`date=`*date*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the item affected relationship was created to be set to *date*.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Part Item Affected allows the arbitrary fields of a part relationship item to be modified/set. Select the relationship of interest in the Part Viewer or select a Part Item Affected relationship in the Part Items Affected Relationships browser and then use the context menu or Part Menu to view the details of the relationship and modify these details: remember to save your changes using **Misc | Update**.

Command Line Syntax

alterpartitemaffected [*arb1*...*arb40*=*value*] `itemaffected`=*item* `type=part|cr|baseline|file` [`-solved`] [date=*date*] [user=*user*] *parts*

Alter Part Location

Function

Accessed from **Part | Alter | Alter Location**. Allows the location field of a subsystem or component to be modified

Class

Part

Options

Part

Specifies the part whose location field is to be modified

Location

The location field of the part will be changed to the specified new location. Use `none` to set the part to have no explicit location (i.e. it reverts to the default behaviour of inheriting its location from its parent plus its own name). Note that changing the location of a subsystem type part could affect any children of that part.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Location allows the location field of a part (subsystem or component) to be changed.

The location of a subsystem or component type part defines the physical directory or file storing the physical contents of the part.

The location of a subsystem type part should refer to an operating system directory.

The location of a component type part should refer to an operating system file: this file is (normally) a VC file which contains the complete version history of the component.

It is important to understand that the directory/subsystem is the place (normally) where the version history (VC) files/ components will be stored. These files are an integral part of the ***AllChange*** parts database and should only be manipulated by ***AllChange***.

If no location field is given then the actual name of the part is taken as the value for the location field. This means that in the normal case, it is not necessary to specify or alter the location field. Normally only the database administrator will set or change a location field and by default this command is only available to ***AllChange*** administrators.

The location field may need to be set under the following circumstances:

- The root part will usually have a location to define the directory where all directories and files for parts in the database will be stored. If the part name required exceeds the length allowed for the

operating system file or directory it represents, then the part name may be the full desired name and the file name may be specified to be some abbreviation of this.

- It is desired to have a different directory hierarchy to the parts hierarchy.

Altering the location field should only be done when the physical location representing the part is to be changed. If the part is of type *subsystem* then altering the location alters the location of all of the children of the part if they inherit the location of the parent. Care should be taken that all component type parts affected by an alter location should have their physical files altered accordingly (this may need to be done manually).

Care should therefore be taken when altering the location of a subsystem since the reasons for altering may vary. Possible reasons include:

- The path to a directory has been changed and this must be reflected in the parts database: in this case there in no real need to perform any command actions since the external filing system is already as required and all that is necessary is that the parts database reflects this.
- It is desired to re-organise the physical filing system, in which case new directories may need to be created and files moved around. A change of this nature will affect children of parts etc., and any necessary renaming of files should be made manually.

Further details may be found in the AllChange Administrator Manual.

Command Line Syntax

alter `location=`*newlocation parts*

Alter Part Obsolescence

Function

Accessed from **Part | Alter | Make Obsolete**. Allows the obsolete flag on a part to be changed.

Class

Part

Options

Parts
Specifies the parts to be changed.

Make Obsolete
Allows the **Obsolete** flag for the part to be set.

Clear Obsolete Flag
Allows the **Obsolete** flag for the part to be cleared.

Related Commands

Alter Version Obsolescence, **Alter Part**, **Alter Part Flags**, **Alter Part Class**.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Part Obsolescence allows the **Obsolete** flag of a part to be altered. The flag may be used in conditions to restrict actions to only those versions which are not obsolete, for example. Obsolete parts are not shown in the Parts Browser unless **View | Show Obsolete Items** is selected.

The out-of-the-box configuration requires a CR to be specified when making a controlled part obsolete. The part and default version are then associated with the CR and the Obsolete arbitrary field on the relationship (if the field is defined) is set to Yes to indicate that the part was made obsolete as a part of the changes required to implement the CR.

Command Line Syntax

alter [obsolete=on | off] *parts*

Alter Version

Function

Accessed from **Part | Alter Version ...** or via the toolbar. Allows the fields of versions to be altered.

Class

Part

Options

Parts

Specifies the versions which are to be altered.

Varb1...Varb40

The arbitrary fields of the version will be changed to the specified *value*s. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

status =*status*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the status of the version to be set to *status* with no conditions or actions being invoked.

symbname=*symbname*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the symbolic name of the version to be set to *symbname* .

user=*user*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the user who created the version to be set to *user*.

date=*date*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the date of the version to be set to *date*.

pred_ver=*pred_ver*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the predecessor of the version to be set to pred_ver, the ID of the predecessor version, e.g. `002` or `Branch1.001`. *This should be used with care; no validation is performed on the new value*.

text=*filename*

This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the version to be taken from the filename specified.

-noget
This option is only available from the command line and is not presented as an option in ACE. This causes no workfile to be extracted. This is useful when the workfile already exists (e.g. supplied by a third party). *date*.

Related Commands

Add Part, **Alter Part**, **Check Out**, **Return Part**, **Part Text**.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Version allows the version arbitrary fields of version(s) to be altered.

Command Line Syntax

alterversion [*varb1*...*varb40*=*value*] [flags=no_ver | dup_ver | none] [-comment *comment*] [-noget] [status=*status*] [user=*user*] [symbname=*symbname*] [date=*date*] [pred_ver=*pred_ver*] [obsolete=on | off] [text=*filename*] [[-deleteparts] partsaffected=*version-list*] *versions*

Alter Version Flags

Function

Accessed from **Part | Version | Alter Flags**. Allows the flags on a version to be modified.

Class

Part

Options

Parts
Specifies which part versions are to be changed. A version must be explicitly specified.

No Flags
Will remove the **NoVersion** flag.

NoVersion
Will change a **DuplicateVersion** flag to the **NoVersion** flag.

Comment
Specifies a comment to be used when clearing the **NoVersion** flag.

Related Commands

Alter Version, **Alter Version Obsolescence**, **Newversion**, **Checkout**, **Checkin**.

Concepts

Parts (Creating and Managing Parts)

Description

Alter Version Flags allows the flags on a version to be changed. A version's flags can be altered from **DuplicateVersion** to **NoVersion** by specifying **Flags** of **No Version** (this is what happens when an **issue -edit** is carried out during a status change which creates a new version of a component); or from **NoVersion** to nothing by specifying **No Flags** (this is what happens when a **return -edit** is performed). This

command should not normally be used to alter flags, since flags will usually be altered via the **check out** and **check in**commands. To prevent its accidental use the user must have **dbadmin** permission on the part when altering flags. There are times, however, when this facility may be useful.

Command Line Syntax

alterversion [flags=no_ver | dup_ver | none]

Examples

```
alterversion flags=None /subsystem/component;004
```

removes the no_ver flags from /subsystem/component;004 thus making the version a permanent version instead of a reserved one.

Alter Version Obsolescence

Function

Accessed from **Part | Version | Make Obsolete**. Allows the obsolete flag on a version to be changed.

Class

Part

Options

Parts
Specifies the part versions to be changed.

Make Obsolete
Allows the **Obsolete** flag for the part to be set.

Clear Obsolete Flag
Allows the **Obsolete** flag for the part to be cleared.

Related Commands

Alter Part Obsolescence, **Alter Version**, **Alter Version Flags**, **Newversion**,

Concepts

Parts (Creating and Managing Parts)

Description

Alter Version Obsolescence allows the **Obsolete** flag of a specific version of a part to be altered. The flag may be used in conditions to restrict actions to only those versions which are not obsolete, for example.

Command Line Syntax

alterversion [obsolete=on | off] *versions*

Alter Vote

Function

Accessed from **Vote | Cast Vote** or via the cycle viewer double click on a vote status or via the Vote Viewer. Allows a vote to be changed. Votes may be for Baselines, CRs and Parts.

Class

Baseline, CR and Part

Options

Date
The date/time on which the vote was cast

User
The user who cast the vote.

Item
The item that the vote is related to. This may be a part, a CR or a baseline.

Vote
The status voted for. This may be one of the valid progressions in the life cycle or **(No Preference)** to indicate a vote but no preference as to which progression.

Role
The role/capacity in which the voter has voted.

Comment
A textual comment about the vote being cast

Arb1..Arb40
Allows information to be assigned to the arbitrary fields. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog

`newdate=`***date***
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the date that the vote was cast to be changed. Note that date, user and vote must be specified on the command line to uniquely identify the vote record when this option is used. The new date should be specified in local format.

`status=`***status***
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the status that the item had when the vote was cast to be changed.

Related Commands

Add Vote, Delete Vote

Concepts

Voting

Description

Alter Vote allows details of a vote identified by the date and the user who cast the vote to be changed. Votes on Parts, Baselines and CRs may be changed.

The vote cast (the status voted for) and the role may only be changed whilst the vote is still open and by the user who cast the vote.

The Comment and arbitrary fields may be modified at any time.

Command Line Syntax

alterbaselinevote| **altercrvote|** **alterpartvote**	date=<date> user=<user> [status=<status>] [new-date=<date>] [vote=<vote>] [role=<role>] [com-ment=<comment>] { <field>=<value> } <item>

Examples

```
alterbaselinevote date="2008/06/23 11:31:10" user=mary vote=Pass Base-
      line;3.2
```

Changes the vote for Baseline;3.2 cast by mary on 23rd June 2008 at 11:21:10 to vote for Pass.

```
alterpartvote date="2008/06/02 14:30:58" user=fred comment="This needs
      some additional work" /Product/Documentation/UserGuide.doc;003
```

Changes the comment on the vote for /Product/Documentation/UserGuide.doc;003 cast by fred on 2nd June 2008 at 14:30:58 to "This needs some additional work"

```
altercrvote date="2008/03/10 11:15:31" user=mary vote=Approved
      RFC00010
```

Changes the vote for RFC00010 cast by mary on 10th March 2008 at 11:15:31 to Approved.

Archive Baseline

Function

Accessed from **Baseline | Archive Baselines | Archive**. Allows Baselines to be archived to an external file allowing them to be deleted from the database. They may be re-imported at a later date (see Import Baseline).

Class

Baseline.

Options

Baselines

Specifies the Baselines that are to be archived.

Related Commands

Import Baselines, **Delete Baselines**, **Creating Baselines**, **Modifying Baselines**,

Concepts

Baselines (Creating and Managing Baselines)

Description

Archive Baseline allows Baselines to be archived to an external file allowing them to be deleted from the database. You may wish to use this facility if there are a large number of old Baselines which are no longer of any interest.

Baseline Archiving is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators. Furthermore, by default the Baseline archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

Archive Baseline takes the specified Baselines and uses the ***AllChange*** reporting facilities to save all information about the Baselines to an external archive file called `blarch.exp` in your project directory. Each time baselines are archived they are appended to this file. Note that information is saved about all file attachments associated with the baselines, but only the non link attached files themselves are archived.

The report format used is the supplied `blexport.rep`. All information about the Baselines is written to this file.

Having exported the baselines, when you are happy that you no longer want the baselines in the current baseline database you should use **Baseline | Delete** to actually delete them.

This facility is implemented as an ACCELscript in the function file `archfunc.ac`.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Archive_Baselines**)

call(**ArchiveBaselines**, *archive-file*, *baselines*)

Archive CR

Function

Accessed from **CR | Archive CRs | Archive**. Allows CRs to be archived to an external file allowing them to be deleted from the database. They may be re-imported at a later date (see Import CR).

Class

CR.

Options

CRs
Specifies the CRs that are to be archived.

Related Commands

Import CRs, **Delete CRs**, **Creating CRs**, **Modifying CRs**,

Concepts

CRs (Creating and Managing Change Requests)

Description

Archive CR allows CRs to be archived to an external file allowing them to be deleted from the database. You may wish to use this facility if there are a large number of old CRs which have been closed and are no longer of any interest.

CR Archiving is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators. Furthermore, by default the CR archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

Archive CR takes the specified CRs and uses the ***AllChange*** reporting facilities to save all information about the CRs to an external archive file called `crarch.exp` in your project directory. Each time CRs are archived they are appended to this file. Note that information is saved about all file attachments associated with the CRs, but only the non link attached files themselves are archived.

The report format used is the supplied `crexport.rep`. All information about the CRs is written to this file, including the CR text.

Having exported the CRs, when you are happy that you no longer want the CRs in the current CR database you should use **CR | Delete** to actually delete them.

This facility is implemented as an ACCELscript in the function file `archfunc.ac`.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Archive_CRs**)

call(**ArchiveCRs**, *archive-file*, *crs*)

Archive Part

Function

Accessed from **Part | Archive Parts | Archive**. Allows Parts to be archived to an external file allowing them to be deleted from the database. They may be re-imported at a later date (see Import Part).

Class

Part.

Options

Parts
Specifies the Parts that are to be archived.

Related Commands

Import Parts, **Delete Parts**, **Creating Parts**, **Modifying Parts**, **Archive Version**

Concepts

Parts (Creating and Managing Parts)

Description

Archive Part allows Parts to be archived to an external file allowing them to be deleted from the database. You may wish to use this facility if there are old Parts which are obsolete and are no longer of any interest.

Part Archiving is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators. Furthermore, by default the Part archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

Archive Part takes the specified Parts and uses the ***AllChange*** reporting facilities to save all information about the Parts to an external archive file called `paarch.exp` in your project directory. Each time Parts are archived they are appended to this file. The VC files for each part are also saved in an archive subdirectory of the project directory .

If a subsystem is specified then it and all its children will be archived.

The report format used is the supplied `paexport.rep`.

Having exported the Parts, when you are happy that you no longer want the Parts in the current Part database you should use the **Part | Delete** to actually delete them.

This facility is implemented as an ACCELscript in the function file `archfunc.ac`.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Archive_Parts**)

call(**ArchiveParts**, *archive-file*, *parts*)

Archive Versions

Function

Accessed from **Part | Archive Parts | Archive Version** . Allows specific versions to be archived to an external file allowing them to be deleted from the database. They may *not* be re-imported at a later date but may be checked out for read only purposes if they remain on-line.

Class

Part.

Options

Versions
Specifies the versions that are to be archived.

Related Commands

Import Parts, **Delete Parts**, **Creating Parts**, **Modifying Parts**, Archive Part

Concepts

Parts (Creating and Managing Parts)

Description

Archive Version allows versions of parts to be archived to an external file allowing them to be deleted from the parts VC file . You may wish to use this facility if there are old versions which are obsolete and are no longer of any interest,particularly if the file is a binary file and each version is large. By removing the versions from the VC file this may provide a performance enhancement for checkin/out.

Version Archiving is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators. Furthermore, by default the Version archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility. In addition the there must exist a version arbitrary field named **ArchType**.

Archive Version takes the specified Part versions and extracts a workfile copy of the version and saves it in an archive subdirectory of the project directory. The version is then deleted from the VC file for the part and the **ArchType** field for the version is set to **OnLine**.

OnLine archived versions may be checked out for read only purposes.

The workfiles for archived versions may be moved off-line if desired and then the ArchType arbitrary field should be set to **OffLine**.

This facility is implemented as an ACCELscript in the function file `archfunc.ac`.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Archive_Version**)

call(**ArchiveVersion**, version)

Assign CR

Function

Accessed from **CR | Assign...** or via the toolbar. Allows change requests (CRs) to be assigned to a user.

Class

CR

Options

CR Nums
Specifies a list of the CR numbers to be assigned to the user.

User
Specifies the user/ group to whom the CRs are to be assigned. Use `none` to set the CRs to have no assignee.

Related Commands

New CR, **Alter CR**, **Change CR Status**, **CR Text**.

Concepts

CRs (Creating and Managing Change Requests).

Description

Assign CR assigns CRs to a user/ group, logging against the CR the current assignation. This command should be used both for assigning a CR for the first time and for re-assigning a CR.

For each CR assigned the user will be informed of this using the host mailing system. If the CR is assigned to a group then the members of the group will be mailed.

In addition to informing a user that a CR has been assigned to them, the assignment is also logged in the status log for the CR. The status will be **Assigned**, the date and time of the assignment is logged, the user doing the assigning is logged and the user/ group assigned to is logged in the status log arbitrary field.

Command Line Syntax

assigncr ([`-user`] *user*) *cr-numbers*

Examples

```
assigncr jim CR00010 CR00020 CR00030
```

assigns CR numbers CR00010, CR00020 and CR00030 to user `jim`.

Attach Workspace

Function

Accessed from **Workspace | Attach Workspace...** or via the toolbar. Allows a user to attach to a workspace .

Class

Workspace

Options

workspace

Specifies the name of the workspace to which to attach.

Related Commands

Build, **Change Working Part**, **Check Out**, **Promote Object**, **Check In**.

Concepts

Part (CheckingFiles In and Out of ***AllChange***),

Description

Attach will attach the user to a selected workspace. The workspace must be a valid workspace for the current user.

Attaching to a workspace will change the current working part to the part associated with the workspace. It will also change the user's current working directory to that associated with the workspace.

Users must be attached to a workspace in order to perform commands which operate on workfile in a local workspace such as **Check Out**, **Check In**, **Build** etc.

Command Line Syntax

attachws *workspace*

Examples

```
attachws productws
```

attaches to the workspace named `productws`, changing the current working part and the current working directory to those associated with the workspace.

Autobuild

Function

Accessed from **File | Build | Autobuild** or via the toolbar. Allows a *buildfile* to be generated or the dependencies in an existing buildfile to be updated.

Class

File

Options

Target

Specifies the name of the target file, i.e. the file which is ultimately to be constructed e.g. the name of an executable program. Only one target file may be specified at a time, but additional targets may be added to the buildfile by repeated use of **Autobuild**. If the target is not already mentioned in the buildfile it will be added, otherwise its source files and dependencies will be updated. If no target is specified then all targets in an existing buildfile will be updated.

Source

Specifies the source files on which the **Target** ultimately depends.

Buildfile

Specifies the name of the buildfile to be created or updated. If not specified then `buildfil` will be used.

Include dirs

Specifies additional directories to be checked before those specified in the template file. If this option is not used and no directories are given in the template file the current directory is searched.

Macro defs

Allows macros to be defined in the same way as for the build tool, see Build

Do not recalculate dependencies

If selected prevents recalculation of the final dependency lists that appear at the end of the buildfile: these will just be copied from the old buildfile. This is useful in order to prevent recalculation of the dependency lists each time if **Autobuild** is to be called repeatedly to add/ delete/ replace source files for several different targets. **Autobuild** can then be run once *without* this option at the end, saving a considerable amount of time.

Source Files

if **Add** is selected then any **Source** files specified for the **Target** are added to the dependencies for the target. If **Replace** is selected then **Source** files specified will replace any currently defined dependencies for the **Target** If **Delete** is selected then any **Source** files will be deleted from the dependencies for the **Target**

Related Commands

Build, **SeeBuild**.

Concepts

File (File Operations)

Description

Autobuild allows a *buildfile* for use by **Build** to be set up and maintained.

In order to generate the buildfile using **Autobuild**, all the parts corresponding to the source files involved in the build must be checked out to the current workspace .

If a buildfile does not already exist (in the current directory) one is created using information from a template file together with that given on the command line. From then on **Autobuild** will maintain the buildfile, recalculating dependency lists, updating source and object macros and adding new targets whenever needed.

For full details see How to Set Up a Buildfile.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Autobuild**)

call(**AutobuildSelectedFiles**, *files*)

Baseline Text

Function

Allows the text of a baseline to be edited, imported and exported to file.

Class

Baseline

Related Commands

Add Baseline.

Concepts

Baselines (Creating and Managing Baselines).

Description

Baseline Text allows the text of the specified baseline to be extracted to a file, edited and returned to the system. Three ***AllChange*** commands are available for editing, importing and exporting the baseline Text to file.

These are only available via the ***AllChange*** command line

The text of a baseline may be any arbitrary text. When the baseline was created, the ***AllChange*** Administrator will probably have set the system so as to create a template for the baseline text. This should be completed as required by your site.

`editbltext` invokes your editor on the text file corresponding to the specified baseline.

`getbltext` will retrieve the text of the baseline from the system and place it in an external file.

Unless otherwise specified, the name of the file is generated from the baseline name. The file name is the name of the baseline with any punctuation characters removed, e.g. the text for baseline name Release;1.0 would be placed in a file named `Release10`. The rules are explained in detail in the ***AllChange*** Administrator Manual.

Unless otherwise specified the file will be created in the current directory, and may then be examined/ altered using any tools available.

The baseline text may be returned to the baseline database when changes have been made using `putbltext`.

If the baseline text was only retrieved in order to examine it, when finished with it the file may simply be removed.

`putbltext` updates the text of a baseline to that from a file.

It takes the text of each baseline specified from a file and replaces the existing baseline text, if any, with this. The filename used will be computed in the same way as `getbltext` unless otherwise specified.

Command Line Syntax

editbltext *baseline-names*

getbltext [`-file` *filename*] *baseline-names*

putbltext [`-file` *filename*] *baseline-names*

Examples

```
editbltext Release;1.0 Product;1.1 Design;1
```

extracts, invokes your editor on, and then replaces, the text of the specified baselines.

```
getbltext Release;1.1 Meta;2.3 Design;Proto
```

extracts the text of baseline names Release;1.1, Meta;2.3 and Design;Proto from the baseline database, placing them in files `Release11`, `Meta23` and `DesignProto` respectively.

```
putbltext Release;1.1 Meta;2.3 Design;Proto
```

stores the text of CR numbers Release;1.1, Meta;2.3 and Design;Proto into the baseline database, taking them from files `Release11, Meta23 and DesignProto` respectively.

Branch Editor

Function

Accessed from **Part | Edit Branches**. Allows permissible branch names to be added, updated and deleted.

Class

Part

Options

Branch

This is the name of the branch.

Part

This is the top level part (subsystem) with which the branch is to be associated. The branch name is only valid for use with components within this subsystem.

Concepts

Parts (Creating and Managing Parts), (Checking Files In and Out)

Description

The branch editor allows the list of predefined branch names to be modified. Permission to **Add**, **Delete** and **Edit** branch names may be set by the ***AllChange*** administrator.

ACCEL Syntax

This facility is implemented via ACCEL:

call(EditBranches)

Build

Function

Accessed from **File | Build...** or via the toolbar. Allows a system to be built.

Class

File

Options

Target
Specifies the object to be built. If omitted the default target (specified in the buildfile) will be used.

Desired BT Only
Only generates the desired Build Tread.

Force New Desired BT
Forces full regeneration of the desired Build Thread. Normally, in the interests of speed, if the desired BT already exists it is merely updated to reflect the desired versions of components already named in the file. This option should be used if the relationship between parts and files has changed, e.g. a file or part has been added, deleted, renamed or moved.

No Action
This option tells **Build** just to echo to the screen what actions are to be performed, rather than actually performing them. Even lines beginning with `@` will be echoed, while lines beginning with `+` are still performed — see Action Line Modifiers.

It is very useful to see what would happen if the **Build** were actually to be executed and/ or to discover which files are out of date.

Debug
This option causes **Build** to print out debugging information on the screen while it is trying to build targets.

The information is not usually of interest to the user, but it may help to show up an error in the buildfile's dependencies if **Build** is not doing what it was expected it to do. Included in the output is an indication of which dependent(s) cause a target to be built. The output includes an (internal) representation of files' last modification times; `-1` indicates that a file does not exist.

Print Information
This option causes **Build** to print out all macro definitions and target descriptions on the screen instead of trying to build a target.

Like the **Debug mode** option, this is not usually of interest to the user but may be useful in understanding what **Build** is trying to do.

Ignore Errors
Normally, **Build** will cease trying to build a target if an error code is returned from issuing a command to the operating system.

Using this option tells **Build** to ignore error codes returned from actions issued to the operating system. It should be used if the operating system returns many inappropriate error codes.

Error codes returned by individual actions can be ignored by preceding the actions with - — see Action Line Modifiers.

Continue on Errors

Normally, **Build** will cease trying to build all targets if an error code is returned from issuing a command to the operating system, or if a file must be built but the file does not exist and there are no applicable rules stating how to build it.

Using this option tells **Build** to continue trying to build other targets which do not depend on the current entry. It is useful if there are several targets to build and **Build** is to perform as many actions as possible, even though some of these may return errors, e.g. if one source file fails to compile other source files could still be compiled.

Build will inform the user of any files which were not remade because of errors.

Unconditional

This option forces all targets, and their dependents, to be rebuilt unconditionally, i.e. without regard to their last modification times.

It might be used if, say, a new version of a compiler had been installed, or to recompile with different compilation flags: in either case all object files should be recompiled regardless of source file modification times.

Related Commands

Attach Workspace, **Check Out**, **Promote Object**, **Check In**.

Concepts

Files (File Operations)

Description

Build allows you to "build" a system. In the general case this means compile source modules and link object modules together to create one or more executable programs. It may also be used for other activities such as processing text documents.

The ***AllChange*** build allows a particular configuration of a system to be built based on the versions of the parts that are required to build the system. The versions required are determined by the default versions defined for each part involved in the build process (see Default Versions) for the current workspace .

AllChange will determine which versions of all the parts needed to build the target are required and will rebuild those objects which do not yet exist composed of those desired versions.

An object is looked for first in the current workspace and then in each of the default pools (see Workspace Registrations).

The **Build** command from ***AllChange*** will cause the generation of a list of the required versions of the parts that are to be used in the build process: this is known as the *desired BT*. It will then call an external build utility to actually perform the operations necessary to do the build.

The external build utility uses the concept of *build threads* (BTs) to determine whether a target needs to be rebuilt, rather than time stamps. In this way the objects are only rebuilt if the existing object (either in the current workspace or in the pools specified) is not composed of the right versions of all its component parts. Time stamps are only used for items which are checked out for edit and are therefore being changed without changing the version.

For each target built a BT is created which describes exactly what versions of each element were used to make up this target, together with the compiler options etc.

The build determines whether an object needs to be rebuilt based on the BT for an existing instance of the object and the required BT for the object. If there is a difference in any of the required versions of the dependents or in the actions, then the object must be rebuilt. The build will look for an acceptable object, first in the current workspace, then in each of the pools registered for the workspace. This is achieved using the external build utility's `VPATH` feature.

This means that the ***AllChange*** build facility can maintain multiple configurations simultaneously. It is also able to detect changes such as changes in the compiler flags used (e.g. compiled with optimisation/debug) and know to rebuild.

Full details of the ***AllChange*** build facilities may be found in Building.

Command Line Syntax

build [`-btonly`] [`-force`] [*target*] [`-options` *options-for-build*]

Change Working Part

Function

cwpchanges the current working part .

Class

Misc

Options

part

Specifies the part that is to be the new current working part.

Related Commands

Attach Workspace.

Concepts

Parts (Creating and Managing Parts).

Description

In ***AllChange*** there is a concept of a current working part: this is analogous to the current directory in an operating system.

The current working part may be changed by the **cwp** command; the **Attach Workspace** command also causes the current working part to change.

If the current working part is changed to a uses type part then the current working part will be changed to the part used.

Command Line Syntax

cwp *part*

Examples

```
cwp subsys
```

changes the current working part to the `subsys` child of whatever the current working part is at present.

Check Baseline Check-outs

Function

Accessed from **Baseline | Check Baseline Check-outs**. Allows a check to be performed on the difference between versions checked out to the current workspace and the versions in a baseline .

Class

Baseline

Options

Baseline
Specifies the baseline to be checked against

Concepts

Baselines (Creating and Managing Baselines)

Description

Check Baseline Check-outs performs a check as to the difference between versions checked out to the current workspace and the versions in the specified baseline.

By default this is only available to ***AllChange*** administrators, but may be made generally available if required, see the Menu Items section of the ***AllChange*** Administrator Manual.

It is implemented using the **VisDiffs** tool to display any discrepancies.

This is useful to ensure that the correct baselined versions are checked out to the current workspace.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Check_Baseline_Issues**)

Check Check-out Changes

Function

Accessed from **Workspace | Check Check-out Changes**. Performs a check on which workfiles for parts checked out to the current workspace have been changed.

Class

Part

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Check Check-out Changes reports whether the contents of each workfile checked out to the current workspace have changed from the version checked out. This is particularly useful to determine whether read-only files have been tampered with.

Each file that has been modified will be reported in the output window. If not output is produced then no files have been changed.

By default this is only available to ***AllChange*** administrators, but may be made generally available if required, see the Menu Items section of the ***AllChange*** Administrator Manual.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Check_Issue_Changes**)

Check In

Function

Accessed from **File | Check In** , **Part | Check In**, Explorer **Check In**, Word **CheckIn**, MCSCCI**CheckIn**. Allows a individual files, a directory tree, or parts to be *checked in*to ***AllChange*** and placed under ***All-Change*** control.

Class

Part/ File.

Options

Directory
This should specify a directory to check in from. This may only be specified if no **Parts** are specified.

Files
This should specify one or more files to check in. This may only be specified if no **Parts** are specified.

File Pattern
This should specify a space separated list of patterns which files must match before they are checked into ***AllChange***. This is prompted for if a **Directory** is specified.

Parts
This should specify one or more Subsystems or Components to be checked in. Parts may only be specified if **Files** and/ or a **Directory** have not been specified.

Check Into Subsystem
This is prompted for when checking in files/directories which are not in a defined workspace to specify the subsystem into which the files/directories should be placed.

Check In Action
This is prompted for when one or more items (parts or files) are currently checked out for edit. You should specify whether to:

- **Discard** all changes
- **Check In** all changes

If **Check In** is selected then:

- if the part does not have a life- cycle associated with it it will be **Return**ed from edit.
- if the part does have a life-cycle then the status will be progressed to the status which has the **CheckIn** attribute and this will in turn cause the part to be **Return**ed from edit

If **Discard** is selected then the workfile will be removed and the componentversion created by the proceeding check out for edit will also be removed.

Newversion Action
This is prompted for when one or more items (parts or files) are currently *not* checked out (for either readonly or edit purposes), but a workfile exists for the corresponding part. You should specify whether to:

- **Skip** files/parts which match this criteria
- **Create New Version** the file/part creating a new version in ***AllChange***

If **Create New Version** is selected then a new version will be created with the default version as its predecessor.

Add Part Action

This is prompted for when one or more files have been selected to be checked in, but for which there is not a corresponding part. You should specify whether to:

- **Skip** files which match this criteria
- **Add** the files as new components to ***AllChange***

Baseline

This specifies the baseline to which any new versions (created as a result of a checkin) should be added. This will appear as an option on the **Add Part Action**, **Newversion Action** and **Check In Action** dialogs.

Class

This specifies the class which any new parts added should have. This will appear as an option on the **Add Part Action** dialog.

Branch

This specifies the name of a new branch to be created on which a new version will be checked in. This will appear as an option on the **Newversion Action** dialog. If no branch is specified then a new version on the trunk will be created.

Check Out afterwards

This determines what action (if any)should be taken after the check in:

(Default): this will check out again read only if the part is of a class which has the **Keep checked out** attribute

Read-only: this will check out again read only

For Edit: this will check out again for edit

CRs For Change

This prompts for the CRs which authorise a change. Those CRs which are currently assigned to you, not locked and in a *valid for change* status will be shown. **CRs For Change** will be prompted for if a new part, or a new version of a part is to be created for a part which is of a class which *requires a CR*.

Compulsory Arbitrary Field

If a new part is to be added or a new version created/ checked in and there are arbitrary fields for that part/ version which have been specified as being *compulsory* then these will be prompted for in turn. The same value will be used for all parts of that class which have been added/new versions created/ checked in.

Related Commands

Check Out, **Add Part**, **New Version**, **Return Part**, **Issue Part** ,

Concepts

Parts (Creating and Managing Parts, CheckingFiles In and Out of ***AllChange***)

Description

CheckIn will check into ***AllChange*** the items specified according to the action selections specified.

It may be accessed from:

- Within ACEfrom the **File** menu when files and/ or directories have been selected in the **File Browser**.
- Within ACE from the **Part** menu when parts have been selected in the **Part Browser**

- Within ACE on context menus and on drag and drop operations as applicable (e.g. drag from the **File Browser** to the **Part Browser**, context menu when items are selected in the **File** or **Part Browsers**).
- From Windows Explorer on the ***AllChange*** context menu when some files and/or directories have been selected , or drag and drop from Explorer to ACE, (requires that the Explorer interface has been installed).
- From Word on the ***AllChange*** **| CheckIn** menu option when a document is open to check in the current document (requires that the Word interface has been installed)
- From development environments which are MCSCCI compliant such as Visual C, Visual Basic. The **CheckIn** option on their menus will invoke the **Check In** function in ***AllChange*** if the MCSCCI interface has been installed.

If a **Directory** or **Subsystem**is specified then the directory or subsystem tree will be descended with every file/ component encountered being submitted for check in, provided it matches any pattern specified.

When checking in a directory ***AllChange*** will include *hidden* files and directories only if the Explorer setting to **Show hidden files** is selected.

Any components or subsystems encountered which are *obsolete* will be skipped unless the **Show Obsolete Items** option is selected.

Files which have a `. bt`extension and the certain special files (`register.ac`, `depends.ac`, `desired.ac` and `pooldirs.ac`) will be excluded since these will be regarded as ***AllChange*** configuration and **Build Thread** files respectively which may reside in work directories and which would not require to be checked in.

When checking in parts, these will be checked in from the *current*workspace , you must therefore be attached to a workspace.

When checking in files the function will calculate which workspace the files specified reside in and automatically attach to that workspace. If the files do not reside in any workspace directory then 2 scenarios are possible:

1. If the **Allow check-in with no workspace** configuration option is not selected (default) then an error will be issued.
2. If the **Allow check-in with no workspace** configuration option is selected then a dialog will be presented allowing the selection of the part subsystem into which the files/directories are to be checked in.

If the workspace is an FTP workspace, then a file/ directory browser will be presented showing the files/ directories on the remote machine/ directory corresponding to the workspace. The files/ directories to be imported may then be selected.

If a part/file is checked in which was checked out with optimistic locking the new version to be created is calculated at this time. If any new versions have been checked in since the workfile was checked out then an error will be issued informing that a merge is required before the part may be checked in.

When checking in the function presents dialogs requesting information as appropriate to the files it finds. On finding the first occurrence of a criteria it will prompt for the required action for this and all further occurrences of this criteria.

The criteria are:

1. Exists as a part which is checked out for edit
2. Exists as a part which is checked out read only
3. Exists as a part but is not checked out
4. Does not exist as a part

The corresponding action which may be taken in each of the above cases is:

1. CheckIn (**Return from edit**, progress status to status with **CheckIn** attribute) — prompt for Discard or Check In changes
2. CheckIn (return from readonly) — performed automatically
3. Newversion — prompt for create new version or skip
4. Add part — prompt for add new part or skip

This is implemented as an ACCELfunction and therefore may be tailored to site specific requirements.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**CheckIn**)

Check Out

Function

Accessed from **File | Check Out** , **Part | Check Out**, Explorer **Check Out**, Word **Check Out**, MCSCCI **Check Out**. Allows individual files, a directory tree, or parts to be *checked out* of ***AllChange*** for read or modification purposes.

Class

Part/ File.

Options

Directory
This should specify a directory to Check Out into.

Files
This should specify files which are to be checked out.

Parts
This should specify one or more Subsystems or Components to be checked out. If a **Subsystem** is specified then the part tree from the subsystem will be descended with each component being checked out if it is not already. If a **Directory** was specified then a dialog is presented which allows **Entire Subsystem** or **Selected Components** to be specified to be checked out. If **Selected Components** is chosen then the components to be checked out should be selected from the list displayed. Note that only those components which are not already checked out for edit will be listed. If **Files** were specified then a dialog is presented showing the parts which correspond to the files.

For Edit
This should be selected if the components are to be checked out for modification purposes. If this is selected then:

- if a part does not have a life- cycle associated with it it will be directly **checked out** for edit.
- if a part does have a life-cycle then the status will be progressed to the status which has the **Check Out** attribute and this will in turn cause the part to be directly **checked out** for edit

In either case a new version will be created as a place-holder with the **Noversion** flag and an entry will be made in the Check-outs database.

If **For edit** is not selected then the components will be checked out for read only purposes.

Locking
The locking option selected determines the new version to be created on check out **For Edit**. Locking may be either **Pessimistic** or **Optimistic**. With pessimistic locking the new version that will be checked in at a later date is determined at check out time and this new version is created as a place holder. With optimistic locking the new version is not determined until check in time, a version is still

Default is selected then the locking used is determined by that specified for the class of the part.

New Branch
This specifies the name of a new branch to be created on which the new version created (as a result of the checkout for edit) will be placed.

CRs For Change
This prompts for the CRs which authorise a change. Those CRs which are currently assigned to you, not locked and in a *valid for change* status will be shown. **CRs For Change** will be prompted for if a component which is of a class which *requires a CR*is checked out for edit. The CR(s) selected which authorise the change are stored in the check-out comment and will be used during the checkin to associate any new versions created with the appropriate CRs.

Related Commands

Check In, **StatusReturn** ,**Issue** ,

Concepts

Parts Creating and Managing Parts, CheckingFiles In and Out of ***AllChange***.

Description

CheckOut will Check Out of ***AllChange*** the items specified into a workspace directory.

It may be accessed from:

- Within ACEfrom the **File** menu when a directory or files have been selected in the **File Browser**.
- Within ACE from the **Part** menu when parts have been selected in the **Part Browser**
- Within ACE on context menus and on drag and drop operations as applicable (e.g. drag from the **Part Browser** to the **File Browser**, context menu when items are selected in the **File** or **Part Browsers**).
- From Windows Explorer on the ***AllChange*** context menu when a directory or some files have been selected (requires that the Explorer interface has been installed).
- From Word on the ***AllChange*** **| CheckOut** menu option (requires that the Word interface has been installed)
- From development environments which are MCSCCI compliant such as Visual C, Visual Basic. The **CheckOut** option on their menus will invoke the **Check Out** function in ***AllChange*** if the MCSCCI interface has been installed.

If **Entire Subsystem** is specified to be checked out then the subsystem tree will be descended with every component encountered being submitted for check out .

For each item submitted for check out it will be checked out providing:

- it is not already checked out for edit
- it is not already checked out read only and **For Edit** is *not* selected.
- it is not *obsolete* and the **Show Obsolete Items** option is not selected.

If **ForEdit** is selected and any component is already checked out read only, then it will automatically be checked in before checking out.

When checking out parts, these will be checked out to the *current* workspace, you must therefore be attached to a workspace.

When checking out into a directory the function will calculate which workspace the directory forms a part of and automatically attach to that workspace. If the directory does not form part of any workspace directory, then an error will be issued.

This is implemented as an ACCELfunction and therefore may be tailored to site specific requirements.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**CheckOut**)

Clear Field

Function

Accessed from **Misc | Administrator Tools | Clear Field**. Allows data in an arbitrary field to be cleared (set to blank).

Options

Database
The database for the arbitrary field to be cleared.

Class
The class for the item whose arbitrary field is to be cleared.

Number
The number of the arbitrary field to be cleared. This may be a number between 1 and 40 for all databases except CR, and 1 and 100 for the CRs database.

Related Commands

Move Field, Fix Field Values

Description

Clear Field allows an arbitrary field's values to be set to blank. This is useful if an arbitrary field definition is deleted. Any data existing for the that field should be blanked so that if the field is re-defined at a later date it does not inherit invalid data .

This function is implemented as an ACCEL script in the function file admintools.ac. By default this is disabled and available only to administrators. See the Administrator manual for how to enable the admin tools.

Command Line Syntax

call(ClearField)

Column Report Wizard

Function

Accessed from **Info | Column Report Wizard**. Allows tabular report formats for Excel and Word to be generated.

Class

Info.

Related Commands

Report, **Report Wizard**,

Concepts

Reports and Queries

Description

Column Report Wizard allows you to generate a new report format file which may be accessed from the **Report Wizard** and **Report** dialogs. The report formats generated are for column/ tabular based reports output to Word or Excel.

You will need to specify:

- Which database is to be reported on.
- Which fields of the database are to be columns of the report. Note that the order in which the **Selected Columns** is shown (top to bottom) is the order in which the report output will occur (left to right).
- Whether you wish the output of your report to go to MS Word or MS Excel.
- The name of the report format file to be generated and a **Summary** describing the report

This information will be prompted for in a series of dialogs.

The Summary will be shown in the report format list from the **Report** dialogs.

The name of the file will determine when the report is offered on format lists of available reports. If you are an ***AllChange*** administrator then the format file will be saved in the ***AllChange***project directory and will be available for all users. If you are not an ***AllChange*** administrator then the format file will be saved in your home directory and will only be available for your use.

The first 2 letters must reflect the database that is being reported on (e.g `pa` for parts, `cr`for CRs). The file name should have a `.rep` extension and this will be added automatically for you.

To run your new report use either the **Report Wizard** or the **Report** option on the appropriate database menu. Your new report format will be present in the list of reports shown on the Excel/ Word Category reports in the report wizard and on selecting the **Format** button on the report dialog.

This facility is implemented via ACCEL:

call(**Column_Report_Wizard**)

Copy Baseline

Function

Accessed from **Baseline | Copy** or via the toolbar. Allows a new baseline to be created based on an existing one.

Class

Baseline.

Options

Old Baseline
The name of the baseline to copy

New Baseline
If **New Version** is selected then a new version of the **Old Baseline** is to be created and only the **Version** needs to be completed with the new version number/ identifier. If **New Baseline** is selected then an entirely new baseline is to be created which may be of a different type. This allows, for example, a release baseline to be created from a design baseline (or vice versa). The **Basename** and, if required, the **Version** for the new baseline should be specified.

Release Date
Specifies when the new baseline is to be released.

Update Details to
Specifies what versions to add to the new baseline.

Related Commands

Add Baseline,

Concepts

Baselines (Creating and Managing Baselines)

Description

Copy Baseline allows a new baseline to be created based on an existing baseline.

Any type of baseline may be copied (design, release or meta) and release baselines may be created from design baselines and vice versa.

The following information will be copied from the baseline header:

- Class
- Type (i.e. meta or part)
- Design
- LockParts
- Locked
- Comment
- TopPart
- All Arbitrary Fields
- Text

Copying the baseline will set the new baseline's **Pred Baseline** field to the name of the baseline copied from, if the field is defined. Also, if a **Target Release** field is defined, then its value is set to the new baseline's name.

The **Update Versions to** selections allow control over the specific versions of parts which are to be added to the new baseline.

If **None** is selected then an empty baseline will be created (i.e. header only), which may be populated at a later date. This is useful for creating new versions of existing baselines for releases which will occur in the future, for which you do not yet know the content.

In all other cases (No Change, Default, Registered or Top Version) then only parts which are *not* obsolete will be copied to the new baseline unless the Show Obsolete flag is set.

If **No Change** is selected then the new baseline will have exactly the same details as the baseline copied from.

If **Default Version**, **Registered Version** or **Top Version** are selected then the version added to the new baseline will be the specified version of the component in the old baseline.

If a version is specified and the baseline being copied is a design baseline, then the new baseline will be a release baseline, otherwise it will be a design baseline.

If a detail item in the baseline being copied from does not exist (e.g a part no longer exists but is still referenced by the baseline being copied), then it is not copied to the new baseline.

Furthermore, if a detail item in the baseline being copied from is obsolete then it is not copied to the new baseline unless the Show Obsolete flag is set.

This allows, for example, a design baseline containing just components to be created to define what the content of a release should be and this can be used to create release baselines of specific versions of those components.

Another use would be to create new releases as the latest version of the items from the previous release.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Copy_Baseline**)
: displays the Copy Baseline dialog, where the user may set the options described above

call(**lowCopyBline**, *old-baseline*, *new-baseline*, *what-version*, *release-date*, *type, class, meta*)
: the baseline to copy is specified in *old-baseline*, with the new baseline's name (including its version) passed in *new-baseline*. The *what-version* argument specifies the version to which the details should be updated, as described above. This should be one of "-none" (or empty), "-def", "-reg" or "-top". This argument may also be a baseline name, preceded by a '!', to indicate that the new baseline's parts should be updated to the versions on the specified baseline. If the *release-date* is specified then the new baseline's release date is set to the date specified, else the current date is used. *Type* specifies 'design' or 'release', or may be left empty to use the source baseline's type. The *class* may be specified, else the source baseline's class is used. If *meta* is ***true***, then the new baseline created will be a meta-baseline.

Copy CR

Function

Accessed from **CR | Copy** or via the toolbar or context menu. Allows a new CR to be created as a copy of an existing CR.

Class

CR

Options

CR Number
: Specifies the CR to be copied

Related Commands

New CR, **Change CR Status**, **Alter CR**, **Assign CR**, **Rename CR**, **CR Text**.

Concepts

CRs (Creating and Managing Change Requests).

Description

Copy CR creates a new CR as a copy of an existing CR.

Copy CR is supplied as a site modifiable function.

The default behaviour is to copy most of the fields of a selected CR into a newly created CR. The supplied function copies the following information:

- class
- reference
- toppart
- summary
- arbitrary fields
- text
- attachments
- partsaffected
- baselinesaffected
- crsaffected

This may have been modified to suit site specific requirements.

Note that only partsaffected,baselinesaffected and crsaffected which are *not* obsolete are copied unless the Show Obsolete flag is set.

Command Line Syntax

`call`(**Copy_CR**)

Copy Part

Function

Accessed from **Part | Copy...** or via the toolbar. Allows you to copy a part and all its children to a new part.

Class

Part

Options

From Part
The path to the part to be copied.

To Part
The path to the part to be created. If an existing subsystem is specified then the part will be copied *into* the **To Part** with the same name as the part copied. If a part is specified which does not exist then the copy will take place to create the **To Part** as a copy of the **From Part** with the new name.

`location=`*location*
Specifies a new location for the new part. This is only available from the command line

Related Commands

Add Part, **Delete Part**, **Rename Part**, **Use Part**.

Concepts

Parts (Creating and Managing Parts).

Description

Copy Part copies a complete design tree or part of a design tree. **FromPart**, which must exist, will be copied to **ToPart**.

All descendants of **FromPart** — subsystems, components and versions — will be copied.

If any component copied is of a class which requires a CR for change a CR will be prompted to authorise the copy. The new component created will then be associated with the CR and all versions will be associated with the CR as versions solved.

No subsystem included in the copy may have an *absolute* path in its location field since this would result in parts of the new subsystem having the same location as parts in the existing subsystem.

If **FromPart** itself has an absolute path as its location field, or if it has a relative path as its location field and the actual location of **NewPart** will be the same as the location of the **FromPart**, then the **`location`** option *must* be used to specify a new location for **ToPart**: this means that the **Copy Part** dialog may not be used and the command required should be issued using the **Misc | AllChange Command** using the command line specified below. It is envisaged that only administrators will have permission to set/modify the location field.

If the part copied to, or any of its children, have a resulting location the same as the corresponding part copied from an error will be issued and the copy aborted.

If the **FromPart** is of type uses , then this part only will be copied to the newpart, creating a new part using the same part as the frompart. The location option is not used when copying a uses type part.

Command Line Syntax

copy [`-location` *location*] ([`-from`] *frompart*) ([`-to`] *newpart*)

Examples

```
copy /product1/document /product2/document
```

produces a copy of the parts tree hierarchy starting at `/product1/document` as `/product2/document`. `/product2` must already exist; `/product2/document` must not. The resulting location for the new part must be different from that of the existing part. Hence `/product1/document` must not have an absolute location in its location field; either it must have an empty location field, or if it has a relative location the locations of `/product1` and `/product2` must differ. In any case no descendants of `/product1/document` may have an absolute location.

```
copy -location newloc subsys1 subsys2
```

copies the `subsys1` subsystem of the current working part to a new subsystem, `subsys2`, of the current working part. The location of the new subsystem will be a subdirectory called `newloc` of whatever the location corresponding to the current working part is.

Create Baseline Hierarchy

Function

Accessed from **Baseline | Create Baseline Hierachy**. Allows a meta baseline hierarchy to be created based on a source meta baseline.

Class

Baseline.

Options

Source meta-baseline
Specifies the name of the baseline hierarchy to be used as the source.

New Basename
Specifies the basename for the new meta baseline to be created

New version
Specifies the version for any baselines created

Allow use of existing baselines
Specifies whether baselines which already exist should be re-used or whether only new baselines should be created

Type
Specifies the baseline type for the new baselines created. This may be either **Design** or **Release**

Class
Specifies the class of any new baselines created. If this is not specified then the class of the source baseline will be used

Predecessor baseline
Specifies the predecessor baseline to be used to populate any part baselines created

Related Commands

Update Baseline, **Add Baseline**, **Copy Baseline**, CRs For Baseline

Concepts

Baselines (Creating and Managing Baselines)

Description

Create Baseline Hierarchy allows a meta baseline hierarchy to be created based on a source meta baseline. This may be useful, for example, to create a design baseline hierarchy from a meta baseline or to create a release baseline hierarchy from a design baseline.

A new (meta) baseline is created of the **New Basename** specified with the **New version** specified and of the **Type** and **Class** specified. For each baseline in the **Source meta baseline** tree new baseline of a name which is prompted for is created (of **Type** and **Class** specified) and added to the parent meta baseline, this creating a new baseline hierarchy.

For any part release baselines created, if a **Predecessor baseline** has been specified then the versions of parts added to the new baseline are those in the predecessor. If no predecessor is specified then the default version is used. The **Pred <baseline-nomenclature>" (e.g. Pred Baseline)** (if defined) will also be set to the specified predecessor if any.

If **Allow use of existing baselines** is specified then existing baselines may be re-used instead of new ones being created.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**CreateBaselineRecursiveDialog**)

call(Do_CreateBaselineRecursive, *source_baseline*, *new_basename*, *new_baseline_version*, *new_baseline_type*, *allow_existing, predecessor*, *class*)

CRs For Baseline

Function

Accessed from **Baseline | CRs For Baseline**. Allows a release baseline to be updated based on the parts which implemented specified CRs

Class

Baseline.

Options

Baseline Name
Specifies the name of the baseline to be modified.

CRs
Specifies the CRs which are to be included in the baseline.

Related Commands

Update Baseline, **Add Baseline**, **Copy Baseline**,

Concepts

Baselines (Creating and Managing Baselines)

Description

CRs For Baseline allows a release baseline to be updated to the versions created as a result of certain change requests. It also allows versions to be removed from a baseline if they have become obsolete (or the component has become obsolete) as a result of a change request.

CRs For Baseline supports meta-baselines and part baselines.

CRs For Baseline is a site modifiable function (defined in `crsforbline.ac`) which as supplied populates the specified baseline with the **versions solved** by specified CRs and associates the baseline with each CR as a **baseline solved** by that CR. In addition if there is a field called "Pred <baseline-nomenclature>" (e.g. **Pred Baseline**) then the content of the predecessor specified is copied to the baseline prior to population with the versions from the CRs.

Thus for a part baseline, if this is empty and there is no predecessor the resulting baseline will contain those versions changed by the CRs applied. If there is a predecessor then the resulting baseline will contain the predecessor plus those versions changed by the CRs applied.

The CRs to be included in the baseline are prompted for. This list of CRs will include only those CRs which are *valid for release in the baseline*. Those CRs which you wish to release should be selected from this list.

CRs which are *valid for release* are those which:

- are in a status which has been defined as a *release* status This attribute is associated with a status by the ***AllChange*** administrator when defining the CR life-cycles.
- have a **Target Release** of the baseline. See below for details.
- have at least one part associated with it which is within the baseline **Top Part**.

A CR's **Target Release** field, if defined, will specify the release that the CR's change is targeting. When looking for CRs which match the released Baseline, the CR's **Target Release** should match the baseline name, or the baseline's **Target Release** field's value. The name of the arbitrary field used as the **Target Release** field for the baseline is specified in the **BaselineTargetReleaseFields** configuration option. This allows CRs targeting a particular release to be included in, for instance, a Beta release, by setting Beta release's **Target Release** to the ultimate final release's name. The name of the arbitrary field which is used for the CR's **Target Release** is specified in the **CRTargetReleaseFields** configuration option. Different field names may be used for different baseline and CR classes. The default field name is **Target Release** and may be modified by the ***AllChange*** administrator; see the ***AllChange*** Administrator Manual for details. If no field is specified in the **CRTargetReleaseFields** option then any releasable CR will be considered for the release.

Once the CRs you wish to release have been chosen, the function will perform the following checks:

1. For each component version associated with a CR a check is made to ensure that there are no other CRs which are associated with an earlier version of the component which have *not yet been released*. If there are and these CRs are not *valid for release*, you will be presented with a list of these and the function will be aborted. If there are and these CRs are *valid for release* then you will be prompted to inform you that these CRs will also be released.

   A CR is regarded as having *been released* if it has a baseline solved associated with it and it is in a status which is a *release* status

2. If the **CRsForBaselineIncludeLinks** configuration option is *True* then all CRs related to the CRs selected must also be included if they are *valid for release*, otherwise an error is given. The default value for **CRsForBaselineIncludeLinks** is *True* and may be modified by the AllChange administrator, see the ***AllChange*** Administrator Manual for details.

It will then calculate a list of the all the latest versionssolved associated with the selected CRs. You will be prompted with a list of all of these which are *not* obsolete (neither the version nor the component) to ask if you wish to add these to the selected baseline. You will then be prompted with a list of all of those versionssolved which *are* obsolete (either the version or the component) and candidates to be removed from the baseline, to ask if you wish to remove them. A version is regarded as a candidate to be removed if it or the component are obsolete and, if there is a CR Item Affected arbitrary field called Obsolete defined, and this has a value of Yes.

If there is a CR Item Affected arbitrary field called Obsolete defined, this is set to the value Yes when a part or version is made obsolete and the CR is specified as the authority for that change. Thus the Item Affected relationship records that it was that CR against which the part was made obsolete. If this field is

not defined, then once a part is made obsolete applying any of the CRs which are associated with the part to a baseline using CRs For Baseline will cause the part to be removed from the baseline.

Selecting **Cancel** to either of the above prompts cancels the entire operation and no changes will be made to the baseline.

If the baseline is a part baseline, then any parts associated with a CR which do not lie within the baseline toppart will be excluded from the baseline and a warning will be issued.

Design baselines may also be used to define the content of the release baseline. If a **Design From** field is defined for the baseline's class, then its value should be set to the Design baseline which defines its content. If a part is not contained within the design baseline then it will not be included in the resulting release baseline.

For Meta baselines, design baselines may be used to define the structure of your release baseline hierarchy, the design baselines may be empty and simply used to define the structure, or they may be populated with the components which are to be included.

If the predecessor baseline is defined for a meta-baseline and its sub-baselines, then the baseline will be populated from new versions of sub-baselines resulting from the changes on the CRs or with the appropriate sub-baseline(s) from the predecessor tree where there are no changes. If there is no **Pred Baseline** field, the baseline will be populated from new baselines prompted for, to contain the versions changed by the CRs. If a Design baseline is specified, then the hierarchy is taken from the design structure.

If the baseline is a meta-baseline then the predecessor baseline is examined to find an appropriate sub-baseline (recursively) for each part, if an appropriate baseline is found based on the baseline toppart then you will be prompted for a new version for that baseline which will contain the predecessor baseline content updated with any appropriate versions. Any sub-baselines which have not changed are copied as sub-baselines in the new meta-baseline tree. If any parts are left over which do not fit into any sub-baseline then a new baseline is prompted for until all such parts are used up. Where a part baseline is contained within a sub-meta-baseline, a new version of the sub-meta-baseline (recursively up the tree) is prompted for. For any brand new baselines created, the toppart will be prompted for and if left blank will be calculated to the highest part for the parts contained within the baseline.

If the baseline is a meta-baseline then the CRs are associated with each sub-baseline as appropriate for the parts in the baseline and all the CRs selected are associated with the top level meta-baseline.

If the baseline is a meta-baseline and an error occurs at any point, the user will be given the options to delete any new baselines/baseline versions created so far.

On confirmation to modify the baseline, the baseline(s) will be updated to the versionssolved and the CRs will have the baseline associated as a *baseline solved*.

If the baseline is a meta-baseline, and Design baselines are not being used, then AllChange determines which parts apply to which sub-baselines according to the baseline top-parts. It is important that none of the top parts for the sub-baselines overlap otherwise the first encountered will be chosen and this could be incorrect.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**CRs_For_Baseline**)

call(**CRsForBline**, *baseline*, *already-selected-crs, existing-crs*)

call(**CRsForMetaBline**, *meta-baseline, existing-crs)*

CR Text

Function

Allows the text of a CR to be edited, imported and exported to file.

Class

CR

Related Commands

New CR.

Concepts

CRs (Creating and Managing Change Requests).

Description

CR Text allows the text of the specified CR to be extracted to a file, edited and returned to the system. Three ***AllChange*** commands are available for editing, importing and exporting the CR Text to file.

These are only available via the ***AllChange*** command line

The text of a CR may be any arbitrary text. When the CR was created, the ***AllChange*** Administrator will probably have set the system so as to create a template for the CR text. This should be completed as required by your site.

`editcrtext` invokes your editor on the text file corresponding to the specified CR.

`getcrtext` will retrieve the text of the CR from the system and place it in an external file.

Unless otherwise specified, the name of the file is generated from the CR number and depends on the numbering scheme used. In the default case — where CRs are just a plain 5 digit number — the file is named `cr`*nnnnn*, where *nnnnn* is the number of the CR, e.g. the text for CR number 00006 would be placed in a file named `cr00006`. The rules are explained in detail in the ***AllChange*** Administrator Manual.

Unless otherwise specified the file will be created in the current directory, and may then be examined/altered using any tools available.

The CR text may be returned to the CR database when changes have been made using `putcrtext`.

If the CR text was only retrieved in order to examine it, when finished with it the file may simply be removed.

`putcrtext` updates the text of a CR to that from a file.

It takes the text of each CR specified from a file and replaces the existing CR text, if any, with this. The filename used will be computed in the same way as `getcrtext` unless otherwise specified.

Command Line Syntax

editcrtext *cr-numbers*

getcrtext [`-file` *filename*] *cr-numbers*

putcrtext [`-file` *filename*] *cr-numbers*

Examples

```
editcrtext CR00011 CR00022 CR00033
```

extracts, invokes your editor on, and then replaces the text of the specified CRs.

```
getcrtext 00005 00055 00555
```

extracts the text of CR numbers 00005, 00055 and 00555 from the CR database, placing them in files `cr00005`, `cr00055` and `cr00555` respectively.

```
putcrtext 00005 00055 00555
```

stores the text of CR numbers 00005, 00055 and 00555 into the CR database, taking them from files `cr00005`, `cr00055` and `cr00555` respectively.

Delete Baseline

Function

Accessed from **Baseline | Delete...** or via the toolbar. Allows baselines to be deleted.

Class

Baseline

Options

Bline

The names of the baselines that are to be deleted.

Related Commands

Add Baseline, **Rename Baseline**, **Alter Baseline**, **Change Baseline Status**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Delete Baseline deletes the specified baselines from the baselines database.

If a release baseline is deleted then any parts which the baseline locked when it was taken will have their lock count reduced.

Note that when a baseline is deleted all its associated information such as the baseline text, the parts/ baselines in the baseline and the status log are also removed. However, unless the user chooses to update such references when they delete the baseline, any items which refer to the baseline are unchanged. e.g. if a CR has a baseline as a baseline affected (and/ or baseline solved), the CR will still refer to the baseline after it has been deleted. The fact that the baseline does not exist will be shown in the CR viewer via an icon.

Command Line Syntax

deletebaseline *baseline-names*

Examples

```
deletebaseline bline1
```

deletes baseline `bline1` from the baselines database, decrementing the lock count on all parts in the baseline if it was a release baseline.

Delete Check-out

Function

Accessed from **Workspace | Delete Check-out**. Allows an check-out log record to be deleted bypassing the normal check in mechanism.

Note that the term check-out may have been mapped to site specific nomenclature (e.g. issue) in which case all references to check-out will show using the mapped nomenclature instead.

Class

Part

Options

Parts
Specifies the part (s) whose check-outs are to be deleted.

Workspace
Specifies the workspace in which the parts are (currently) recorded as checked out

User
Specifies the user to whom the parts are checked out.

Include check-outs for edit
If checked specifies to delete check-out records which are checked out for edit. If not specified check-out for edit records are skipped.

Related Commands

Check In, **Check Out**, **Attach Workspace**, **Issue**, **Return**.

Concepts

Part (Checking Files In and Out of ***AllChange***)

Description

Normally, check-out log records are deleted when a **return**/**CheckIn** command is executed. This is the correct procedure in all *normal* cases.

Occasionally, an ***AllChange*** Administrator wants to delete check-out log records to a workspace or user directly. This might occur if a user or workspace is to be/has been deleted. The normal case for removing check-out records is to attach to each workspace to which the part is checked out in turn and perform a return/check in. After all check-outs to the workspace/user have been returned/checked in the workspace/user may be deleted. The **Delete Check-out** command can be used to remove unwanted check-outsdirectly, without having to attach to workspace(s) however, where a normal return would also delete the file from the workspace, the **Delete Check-out** command will not delete the workfile. This should therefore, only be used in unusual situations, and as supplied requires "dbsuperuser" permission to execute and is only available to Admin users.

If **Include check-outs for edit** is used, check-outs for edit will be deleted (like **Check In** with **Discard Changes**), otherwise check-outs for edit are skipped, for safety

If a **Workspace** is specified then only check-outs to the specified workspace are deleted.

If a **User** is specified then only check-outs to the specified user are deleted.

Parts may specify subsystems, components or versions; only check-outs matching the parts specified are deleted. Use "/" to match any/all parts.

Command Line Syntax

deleteissue [`-unedit`][`-wspc` *workspace*][`-user` *username*] *parts*

Examples

```
deleteissue /subsys/component
```

deletes all (non-edit) check-outs of /subsys/component --- perhaps in preparation for an emergency delete of the component (which would fail if there were any outstanding check-outs).

```
deleteissue -unedit -user goneaway /
```

deletes *all* check-outs to user "goneaway" of any/all parts.

Delete CR

Function

Accessed from **CR | Delete...** or via the toolbar. Allows change requests (CRs) to be deleted.

Class

CR

Options

CR Nums
Specifies a list of the CR numbers to be deleted.

Related Commands

New CR.

Concepts

CRs (Creating and Managing Change Requests).

Description

Delete CR deletes the specified CR(s), including the associated CR text and attachments, from the CR database. This may be useful if an old CR is of absolutely no further interest, or perhaps if a CR has been erroneously added in the first place.

Use with care: once deleted, neither the CR's information nor its text may be may be recovered (unless it has been archived).

Note that when a CR is deleted all its associated information such as attachments and the status log are also removed. However, unless the user chooses to update such references when they delete the CR, any items which refer to the CR are unchanged. e.g. if another CR has a CR as a CR affected, the (other) CR will still refer to the deleted CR after it has been deleted. The fact that the CR does not exist will be shown in the CR viewer via an icon.

Command Line Syntax

deletecr *cr-numbers*

Examples

```
deletecr CR00098 CR00099
```

deletes CR numbers CR00098 and CR00099, together with their CR texts.

Delete Instance

Function

Accessed from **Part | Instance | Delete Instance**. Allows instances to be deleted.

Class

Part

Options

Instances
The full path and id of the instances to be deleted. Only used if **-all** is not specified.

-all
If specified then all instances of a specified version are deleted.

Versions
Specified with the -all option, specifies the full path to the versions for which all instances are to be deleted.

Related Commands

New Instance, Alter Instance

Concepts

Parts (Creating and Managing Parts)

Description

Delete Instance allows instances to be deleted. Either an instance should be specified in which case this is deleted, or a version should be specified together with -all and all instances of the version will be deleted.

Command Line Syntax

deleteinstance [`-all` *versions*] [*instances*]

Examples

```
deleteinstance "/Car/Accessories/CD Player;002:0001"
```

Deletes instance 00001 of version 002 of /Car/Accessories/CD Player

```
deleteinstance -all "/Car/Accessories/CD Player;002"
```

Deletes all instances of version 002 of /Car/Accessories/CD Player

Delete Monitor

Function

Accessed from **Monitor | Delete Monitor...** or via the toolbar. Deletes a monitor on an item.

Class

Monitor

Options

Item Type
Specifies the type of item for the monitor to be deleted.

Event
Specifies the event of the monitor to be deleted. The available events will vary depending on the **Item Type**.

Any
If selected causes monitor to be removed to be on *any* item.

Item
Specifies the objects monitored that are to be deleted.

Arb1
Specifies the arbitrary field value for the monitor to be deleted.

Related Commands

Monitor.

Concepts

Monitors.

Description

Delete Monitor deletes the monitors placed by the current user on the specified event for each item specified.

The details of the monitor are removed from the monitors database.

Command Line Syntax

deletemonitor `event=`*event* `user=`*user* [*arb1*=*value*] *items*

Examples

```
deletemonitor event=cr CR00006
```

deletes any cr-event-type monitors placed on CR`CR00006` by the user.

```
deletemonitor event=newversion *.c
```

deletes any newversion-event-type monitors placed by the user on any children of the current working part ending in `.c`.

Delete Part

Function

Accessed from **Part | Delete...** or via the toolbar. Allows parts to be deleted from the parts database.

Class

Part

Options

Parts

Specifies the parts to be deleted.

Related Commands

Add Part, **Copy Part**, **Rename Part**, **Use Part**, **Delete Version**.

Concepts

Parts (Creating and Managing Parts).

Description

Delete Part removes a hierarchy from the parts database. Each part will be deleted, *together with all its descendants*.

If the part is of type subsystem then each child will be deleted in turn (which in turn will have each of its children deleted etc.) and then the part itself will be deleted.

If the part is of type component , then all of the versions of the part are deleted followed by the part itself.

If the part is of type uses , then just the uses part will be deleted.

Note that when a part is deleted all its associated information such as the part text and status log are also removed. However, unless the user chooses to update such references when they delete the part, any items which refer to the part are unchanged. e.g. if a CR has a part as a part affected (and version solved),

the CR will still refer to the part/ version after the part has been deleted. The fact that the part does not exist will be shown in the CR viewer via an icon.

Command Line Syntax

delete [*parts*]

Examples

```
delete /subsystem
```

deletes the whole part hierarchy — subsystems, components and versions — starting from `/subsystem` downward.

```
delete *.c
```

deletes all `.c` parts — and any versions of these — in the current working part.

Delete Version

Function

Accessed from **Part | Delete Version ...** or via the toolbar. Deletes a version of a component type part .

Class

Part

Options

Parts

Specifies the versions which are to be deleted.

Related Commands

Delete Part, **Check out**, **New Version**, **Check In**.

Concepts

Parts (Creating and Managing Parts).

Description

Delete Version deletes the specified version(s) of component type parts; it will also remove the actual contents of the version from where it is stored under version control. This may be useful if an old version is of absolutely no further interest, or perhaps if a version has been erroneously added in the first place.

It is not permissible to delete a version which has a branch stemming from it, e.g. `part;004` could not be deleted if it were the predecessor of `part;branch1.001`, though it could if its only successor were `part;005`. Note too that, assuming the part is kept under version control, deletion of the last (i.e. only) version will not prove possible.

Use with care: once deleted, neither the version in the parts database nor the actual version of the file associated with it may be recovered.

Note that when a version is deleted all its associated information such as the version text and the status log for the version is also removed. However, any items which refer to the version are unchanged. e.g. if a CR has a version as a version solved, the CR will still refer to the version after it has been deleted. The fact that the version does not exist will be shown in the CR viewer via an icon.

Command Line Syntax

deleteversion *versions*

Examples

```
deleteversion *.c;001
```

deletes version 1 of all components ending in `.c` in the current working part from the parts database, together with the actual version in the associated version control file. No branches must stem from version 1, nor may version 1 be the only version of the component.

Delete Vote

Function

Accessed from **Vote | Delete** . Allows a vote to be deleted. Votes may be for Baselines, CRs and Parts.

Class

Baseline, CR and Part

Options

Date
The date/time on which the vote was cast

User
The user who cast the vote.

Item
The item that the vote is related to. This may be a part, a CR or a baseline.

Related Commands

Add Vote, Alter Vote

Concepts

Voting

Description

Delete Vote allows a vote cast identified by the date and the user who cast the vote to be deleted. Votes on Parts, Baselines and CRs may be deleted.

Command Line Syntax

deletebaselinevote| date=<date> user=<user> <item>

deletecrvote|

deletepartvote

Examples

```
deletebaselinevote date="2008/06/23 11:31:10" user=mary Baseline;3.2
```

Deletes the vote for Baseline;3.2 cast by mary on 23rd June 2008 at 11:21:10.

```
deletepartvote date="2008/06/02 14:30:58" user=fred /Pro-
      duct/Documentation/UserGuide.doc;003
```

Deletes the vote for /Product/Documentation/UserGuide.doc;003 cast by fred on 2nd June 2008 at 14:30:58.

```
deletecrvote date="2008/03/10 11:15:31" user=mary RFC00010
```

Deletes the vote for RFC00010 cast by mary on 10th March 2008 at 11:15:31.

Differencing

Function

Accessed from **Part | Diffs** or from **File | Diffs** or via the toolbar. Computes the differences between different versions of a part or between any two files.

Class

Part, File

Options

Part
Specifies the part you wish to compare

Second Part
Specifies the part that is to be compared with **Part**. This is optional and if unspecified then 2 versions of **Part** will be compared. If specified then a version of **Part** and a version of **Second Part** will be compared.

Version 1
Specifies the version of a part to compared against.

Version 2
Specifies the version of a part to compare **Version 1** with.

Ignore Blanks
This option causes all "blanks" to be ignored during the comparison, see Differencing Options.

Strip New Lines
This option causes empty lines to be ignored during the comparison, see Differencing Options.

Related Commands

Merging.

Concepts

Part (CheckingFiles In and Out of ***AllChange***).

Description

The differencing facilities in ***AllChange*** allow different versions of a part, different parts or two files to be compared and the differences to be displayed.

The functionality is implemented by an external differencing tool called **VisDiffs**. Results are presented as a scrollable table of lines with colours used to indicate each difference and match.

Facilities to locate each difference and match are provided as are options to allow the user to tailor the presentation of the results. Each option can be set from the **VisDiffs** menus (**View** and Option).

They may also be specified as command argument to the **Visdiffs** tool.

Ultimately the **VisDiffs** tool differences 2 files - these may be created by ***AllChange*** containing the contents of the 2 part versions specified, or they may be the 2 files selected in the file browser or explorer (if just one file is selected which does not correspond to a part then **VisDiffs** will be invoked for the one file and will prompt for selection of the second.

The files to be compared may also be specified using the **File | Open** dialog. If too few files are specified on the command line the dialog is automatically displayed. The dialog allows 2 files to be specified. It cannot be accepted (with 'Open') until enough items have been specified. When the first file has been selected

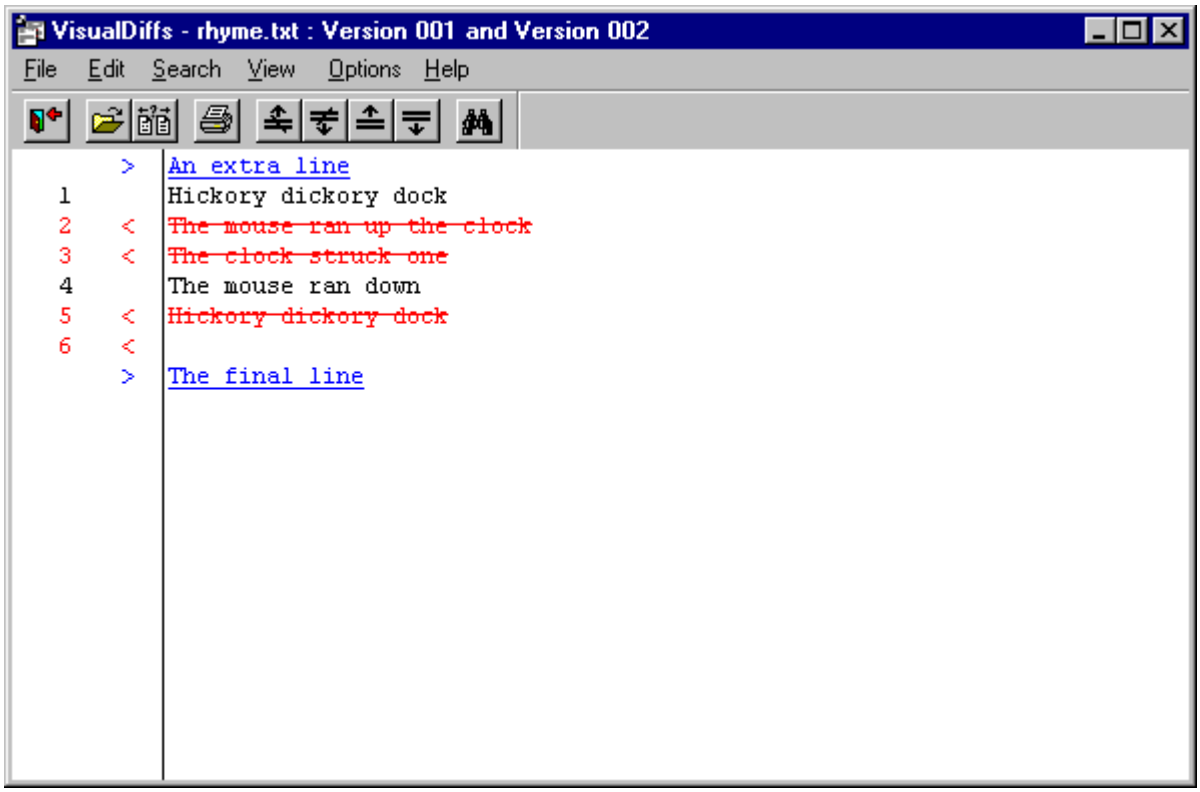
(by double clicking or 'Open') its path is displayed in 'Left File' and another file or release(s) may be selected.

The files are split into lines and compared on a line-by-line basis. The lines which need to be added or deleted from *file1* in order to derive *file2* are then displayed. Note that even if just a single word, or a single character, differs in corresponding lines in the two files the complete line will be shown as having been altered.

The first file or version specified is known as the **left file** and the second as the **right file**.

Figure 41.1 shows what the display looks like.

Figure 41.1: Visdiffs display



By default, as in Figure 41.1, lines which are unchanged are shown in **black**, new lines are shown in **blue** and lines which have been removed are shown in **red**. Lines which have been *changed* are shown as the new line in **blue**, then the old line in **red**.

The **VisDiffs** tools provides access to various facilities to tailor the display using the **View** menu and to modify the differencing parameters on the **Options** menu.

In addition various search facilities are provided on the **Search** menu to help you to locate the area of the file which is special interest.

You may also recalculate the differences, or compare 2 different files using options on the **File** menu.

The Toolbar

The Toolbar allows the most frequently used commands to be accessed directly without having to navigate through the selection menus. Each button on the toolbar executes a command directly.

The toolbar provides the following functions (from left to right):

Exit	Exit **Visdiffs**.
Open	Open files
Redo Diffs	Re-calculate differences
Print	Print
Prev Diff	Find previous difference
Next Diff	Find next difference
Prev Match	Find previous match
Next Match	Find next match
Find Text	Find text

Searching

Facilities are provided to search for the **next/ previous** occurrence of a **difference** or **match**. These area available from the **Search** menu, or from the toolbar.

Next searches forward from the currently selected line and **Previous** backwards. If there is no currently selected line the search starts from the top line on view.

When **View | Lines in Either file** is selected, searches will treat a deletion followed by an insertion is as a single change — this is the default case.

There is also a facility to search for arbitrary text, this is available as **Find** on the **Search** menu.

Enter the text you wish to find in the **Find what** edit control and use the **Direction** radio buttons to determine whether the search is to proceed from the currently selected line *upwards* or *downwards*. The search stops at the last line when searching down and the first line when searching up.

By default the search is case insensitive, if you wish it to be case sensitive then select **Match Case**.

Select the **Find Next** button to search for the required text and **Cancel** to stop searching and close the dialog.

View Options

The **View** menu provides access to various settings which affect the display of the results of the comparison.

Lines in Either File
: Lines which appear in either file will be shown i.e. both differences and matches — this is the default

Lines in One File Only
: Only lines which are unique to either the left file *or* the right file will be shown i.e. only the differences

Lines in Both Files Only
: Only lines which are common to both files will be shown i.e. only the matches

Lines in Left File Only
: Only lines which are in the left file will be shown

Lines in Right File Only
Only lines which are in the right file will be shown

Left Line Numbers
Causes the lines numbers for the left file to be shown

Right Line Numbers
Causes the lines numbers for the right file to be shown

No Line Numbers
Causes no lines numbers to be shown for either file

Location
Causes the *file location character* to be displayed for each line. For lines which are only present in the left file the *file location character* is a <. For lines which are only present in the right file the *file location character* is a >. Lines which are the same in both files are left unmarked.

Strikeout and Underline
Controls whether strikeout and underline are used to indicate whether lines have been added to the left file or deleted from it. If disabled then this will simply be indicated with the used of colour. It is enabled by default.

Differencing Options

The **Options** menu provides access to options which affect how the comparison is computed.

Ignore Blanks
This option instructs **Visdiffs** to ignore all "blanks" when comparing the two files. Without this option, these two lines would be counted as different:

```
the cat sat on the mat
the  cat sat on the mat
```

To be more specific, this option treats all non-ASCII and control characters, as well as spaces and tabs, as blanks. It strips both leading and trailing blanks from lines and then changes all groups of one or more blanks into a single space before comparing lines. Furthermore, if more than one third of the characters in a line are non-ASCII or non-whitespace control characters than that line is treated as though it were empty.

This option should be used if changes in spacing are of no interest.

When changed from the **Option** menu the differences will be recalculated.

This option may also be specified on invoking the differencing tool from ACEwhen differencing part versions.

Strip Newlines
By default empty lines (i.e. lines consisting of just a newline) are significant in the calculation of differences. This option causes empty lines to be ignored.

When changed from the **Option** menu the differences will be recalculated.

This option may also be specified on invoking the differencing tool from ACE when differencing part versions.

Save settings on exit
This option causes all options set to be saved when **Visdiffs** is exited. The settings are saved in the registry.

Any settings saved are always restored when **Visdiffs** starts. In order to reset all options to the default setting the **Visdiffs**section may be emptied using an text editor.

The current directory is saved and restored with other settings.

If a full path is specified for the **left file** on the command line this will set the initial current directory.

Command Line Options

VisDiffs maybe invoked from the command line as follows:

visdiffs [*options*] [*file1*] [*file2*]

where *options* may be:

- **-b** Blanks to be ignored.
- **-c** View location.
- **-k** Use strikeout and underline.
- **-l** View line numbers. This should be followed by R or N to indicate **N**o line numbers, or **R**ight line numbers.
- **-n** Newlines to be stripped.
- **-t** Set window title. This should be followed by the title text
- **-v** View lines. This should be followed by **E**ither file, **O**ne file, **B**oth files, **L**eft file or **R**ight file.
- **-x** Save settings on exit. The state of this option is always saved on exit unless this option is followed with a letter **D** on the command line. In addition the **D** parameter prevents any settings being saved when **Visdiffs** is exited.
- **-f** Exports the resulting differences to a file instead of presenting them interactively. This option should be followed by the export filename. If this has the extension `.rtf` then the results will be exported as RTF, otherwise they will be plain text.

Most options may be disabled by following the option letter with a – character.

Notes

The algorithm used by **Visdiffs** is sophisticated and usually discovers a smallest set of sufficient differences.

Visdiffs splits very large files into more manageable chunks which are then compared against each other. Even so it may require a fair amount of memory to operate successfully. Notes discusses how this may be altered.

Visdiffs cannot produce a reasonable set of differences between wordprocessor files.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**File_Diffs**)

call(**Diffs**)

Display Version Tree

Function

Accessed from **Part | Display Version Tree**. Displays a tree view of the versions of a part

Class

Part

Options

Part

Specifies the parts whose versions which are to be displayed

Concepts

Parts (Creating and Managing Parts).

Description

Display Version Tree shows the versions of a selected part in a tree view. This is particularly useful when branches are used to display the branching structure.

ACCEL Syntax

This facility is implemented via ACCEL:

call(VersionTree)

call(LowVersionTree, *component / version*, *allowselect*)

Edit

Function

Accessed from **Part | Edit** or **File | Edit** or via the toolbar or context menu invokes your editor with the file corresponding to a part as a parameter.

Class

Part

Options

Parts

Specifies the parts that are to be edited.

Related Commands

Attach Workspace, **Check out**.

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Edit invokes the *edit* action for the file/ part selected if there is one, otherwise invokes the *default* action.

If a part is selected which is not checked out to the current workspacethen the appropriate versionof the part is extracted to a temporary file before invoking the editor. The file will be removed on exiting ACE.

This is implemented as an ACCEL function which may be tailored to site requirements.

ACCEL Syntax

call(**edit_sel_parts**)

call(**edit_sel_files**)

Examples

```
edit file.doc
```

will invoke your editor on the workfilecorresponding to the part `file.doc` which must be checked out to the current workspace.

Edit Part Read Permissions

Function

Accessed from **Part |** Edit Part Read Permissions. Allows the permissions for reading/viewing parts in the parts tree to be specified

Class

Part

Options

Part

The part for which the permissions are being specified

Show Permissions for all Parts

If selected the permissions for all parts will be shown and may be edited. If not selected inherited permissions for the specified part are shown but may not be edited.

Recursive

If **Part** is a subsystem then if **Recursive** is selected the permission applies to the subsystem and any parts below it, unless parts below the subsystem have their own permissions.

Class

The class of part for which this permission applies. If **Equal to** is selected the permission applies to all parts of the class. If **Not Equal to** is selected the permission applies to all parts which do not have the specified class.

User/Role

Specifies the roles/users of the specified **User Type** who have permission to read/view the specified parts. If **Any Role** is selected then all users/roles may view the parts. The User

Concepts

Parts (Creating and Managing Parts)

Description

Edit Part Read Permissions allows the read permissions for parts to be edited. This **is — by default — an Administrator only function and will therefore only be available to*AllChange*** users who are defined as Administrators.

Read permissions may be set on any part to determine which users and/or roles are allowed to view parts.

These may be on a per-class basis. If the part is a subsystem, then the permission may be set to be recursive. This means that the permission applies to the subsystem and any parts below it, unless parts below the subsystem have their own permissions for a specified class. This mechanism means that for a subsystem, users with one role can read parts of one class, while users with another role can read parts of another class.

In the **Edit Part Read Permissions** dialog, permissions settings which are inherited from selected part are shown in grey, and may not be edited. To view all explicitly set part read permissions select **Show permissions for all parts**. This allows any of the permissions settings to be edited.

Part permissions are saved in a configuration file named `partperm.ac`, in the project directory. While editing the permissions on a part, no other user can also edit permissions as the file is locked. If the file is already locked when trying to edit permissions, the user is presented with a message informing them of the user who is currently editing the permissions.

ACCEL Syntax

This facility is implemented via ACCEL:

call(EditSelectedPartReadPermissions)

`edit_part_perms(`*partname*`)`

Eval

Function

Accessed from **Misc | Eval...** or via the toolbar. Evaluates an ACCELexpression.

Class

Info

Options

Expression

This may be any ACCEL expression. In ACCthis must be the last argument on the command line; the expression extends to the end of the line and should *not* be quoted.

Related Commands

List, Report.

Concepts

ACCEL

Description

Eval evaluates an expression. The expression may be any valid ACCEL expression (see ACCEL) with the following restrictions:

- Plain field references may not be used, since there are no database records currently active (qualified references may be used)
- Only those variables whose value is always valid may be used.

Eval may be used to evaluate an expression to obtain a result, or may be used simply to perform actions, where the result is not of interest.

If the **Eval** is not being called from a script file (including the startup file) — i.e. it is being used interactively — it prints the result of the evaluated expression. If the expression returns a true/ false value then **Eval** will print `true` if it evaluated to true, and nothing if it evaluated to false. If the expression returns a string, this will be printed.

Command Line Syntax

eval *expression*

Examples

```
eval role
eval workspace
eval cwp
eval cwd
```

prints the user's current role, current workspace, current working part and current directory respectively. There would be no point putting these in a script or startup file since nothing would be printed and there are no side-effects.

```
eval has_perm('program.cob', 'developer')
```

prints `true` if the user has (at least) `developer` permission on `program.cob` in the current working part.

```
eval setvar(editor, 'myeditor')
```

sets the ACCEL variable `editor` to `myeditor` which will then be used as the editor to be invoked when required (useful in system or individual start-up files). The result returned is of no interest.

```
eval command('vcinfo '.part_filename('/src/module.c'))
```

runs the **Vcinfo** tool on the file corresponding to the part `/src/module.c`.

Export to MS Project

Function

Accessed from **CR | Export to MS Project** . Allows information about the selected CR's to be exported to MS Project as a first cut Gantt chart.

Class

CR

Related Commands

Report, **Report Wizard**.

Concepts

CRs (Creating and Managing Change Requests)

Description

The integration with MS Project allows information about ***AllChange*** CRs to be exported to Microsoft Project: this allows the progress of CRs comprising a project to be monitored with MS Project features like Gantt charts. The integration uses the ***AllChange*** OLE Automation capabilities to communicate with MS Project. See the ***AllChange*** Administrator Manual for details of enabling and tailoring this feature.

The **Export to MS Project** function will invoke MS Project and populate it with information about any CRs selected in the CR Browser *plus* all CRs that the selected CRs affect, and so on recursively. Each CR produces a *task* in MS Project; CRs which affect other CRs produce *summary tasks*. This generates a hierarchy of tasks, which is what MS Project typically works with.

The most suitable arrangement for exporting to MS Project is thus obtained if you create one (or more) top-level task(s) in ***AllChange***; each of these may have further tasks among their CRs affected. A presupplied CR class , **task** could be used. At the bottom level each task should have one or more non-task CRs which it affects; these must not affect any other CRs. A sample hierarchy might look like:

```
TSK-00001
    TSK-00002
        CR-00001
        TSK-00003
            CR-00002
        CR-00003
    TSK-00004
        CR-00004
        CR-00005
```

Each CR exported from ***AllChange*** automatically sets the following MS Project attributes:

Task name to the CR summary

WBS code to the CR number

Priority to the CR's **Priority** field if a CR arbitrary field exists with this name.

Other fields are exported but may be changed as desired/ needed.

Notes

The default is the **Description**section of the CR Text.

Resources
The default is the current **Assignee**.

Task Actual Start Date
The default is the date when the CR entered the *InWork*status . A task that has not yet even entered its start status is set to have a duration of 0 days, which means it will not have a visible task bar.

Task Actual Finish Date
The default is the date when the CR entered the *Complete* status. A task that has not yet reached its finish status is set to finish *today*.

Task Anticipated Start Date
The default is the date when the CR entered the status *InWork*. A task with no anticipated start date has this set to the actual start date.

Task Anticipated Finish Date
The default is **cr_Date_Due** arbitrary field if this is used. A task with no anticipated finish date has this set to the actual finish date.

The **Task Actual Start Date** and the **Task Actual Finish Date** are used by MS Project to display a bar representing the duration of the task.

The **Task Anticipated Start Date** and the **Task Anticipated Finish Date** are used by MS Project to display a bar representing the anticipated duration of the task, this MS Project refers to as a *baseline* , though this is not connected to an ***AllChange*** baseline). If you maintain information in CRs about their anticipated duration this can be represented by a baseline bar in the Gantt chart, and compared with the actual duration.

Thus in the default configuration it can be seen that a task's actual duration stretches from status `InWork` to `Complete` and its anticipated duration stretches from status `InWork` to the date stored in the arbitrary field `cr_Date_Due`. These are suitable for use with the supplied CR default life-cycle and arbitrary field definitions, but may be changed to reflect a site's tailoring of these, see the ***AllChange*** Administrator Manual.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Export_to_MS_Project**)

call(**MS_Project**, *crs*)

Find Parts

Function

Accessed from **Part | Find**. Finds parts according to search criteria and presents a list of parts found which can then be used to perform actions on.

Class

Info

Options

Named
Specify the name of the parts that are to be found. This may contain wild_cards. If the name does not contain a wild_card then parts whose name contains that specified are found (i.e. it is assumed to be a search for "**name**").

Look In
Specifies the subsystem that is to be searched.

Recursive
Specifies whether to look down the entire subsystem tree

Versions
Specifies which versions (if any) are to be included in the search. Note that this is ignored if a value is specified for **Containing**.

Only list versions in results
Specifies whether only versions are of interest in the results

Condition
Specifies an ACCELcondition that all parts found must match.

Containing
Specifies all parts found must *contain* the text specified. The **Scan** facility is invoked for this part of the criteria. Note that using this could be very slow if examining a large parts tree.

Match Case
Specifies whether the scan should be case-sensitive.

Related Commands

Report, **ScanBrowsing Parts ,**.

Concepts

Reports and Queries, Creating and Managing Parts

Description

Find Part will search the parts database for parts which match the criteria specified. This includes the ability to scan the contents of the parts for a string as well as the database meta-data about the parts. It should be noted that scanning on a large parts tree could take a significant amount of time.

Any parts found which match the criteria are displayed in the list and may then be used to perform operations. This is useful, for example, if you only wish to check out for edit those parts which are in a particular status .

ACCEL Function

ShowWindow(Find)

Fix Field Values

Function

Accessed from **Misc | Administrator Tools| Fix Field Values**. Checks arbitrary field data for valid values and allows those with invalid values to be reset.

Options

Database
The database for the arbitrary fields to be checked.

Related Commands

Clear Field, Move Field

Description

Fix Field Values checks all **Database** items for valid data for each arbitrary field. This supports Check Box and non-editable Drop Down List fields. If an invalid value is found, the user is prompted for a new valid value. If any data is found in fields which are not defined, the user is prompted to blank it out. This is useful when the drop down list of values for a field changes and when field definitions have been removed and data not cleared.

This function is implemented as an ACCEL script in the function file admintools.ac. By default this is disabled and available only to administrators. See the Administrator manual for how to enable the admin tools.

Command Line Syntax

call(FixFieldValues)

Generate URL for selection

Function

Accessed from **Misc | Generate URL for selection** or via context menus for list. Generates the URL for the currently selected item.

Class

Misc

Options

None

Related Commands

none

Concepts

Not Applicable

Description

Generate URL for selection or via context menus for list. Generates the URL for the currently selected item showing the result in a dialog and in the output window. This can then be copied to the clipboard for pasting into other applications, or creating a desktop shortcut etc.

This is implemented as an ACCEL function defined in utility.ac which in turn calls a function **MakeACInterfaceURL** and **MakeWebInterfaceURL** which generate the URLs for ACE and the AllChange Web Interface for a specified item. MakeACInterfaceURL and MakeWebInterfaceURL may be used in custom accel code and mail subject or body for mailing within life cycles as predefined actions.

See also URL Protocol in the **Administrator Manual**.

ACCEL Syntax

call(GenerateURLForSelectedItem)

call(MakeACInterfaceURL, *command*, *type*, *item*, *extrainfo*)

call(MakeWebInterfaceURL, *command*, *type*, *item*, *extrainfo*)

where **command** is view or cast_vote, if unspecified the view is the default; **type** is cr, part, baseline or issue; *item* is the item to be viewed, and *extrainfo* is of the form *param=value[&param2=value2.....]* and param may be `workspace` when type is issue, or `newwindow=true` to force a new window.

Examples

```
call(MakeACInterfaceURL, 'view', 'cr', cr_number, '')
```

Will generate the URL to view the CR identified by the current CR record. This might produce `all-change://ACDevStandard?a=v&t=cr&i=SCR00011` if the current CR is SCR00011

Get Baseline to Directory

Function

Accessed from **Part | Get Baseline to Directory**. Allows versions of components in baselines to be extracted to any named directory without being logged in the check-outs database.

Class

Baseline.

Options

Baseline

This should specify the baseline to be extracted.

Directory

This should specify the directory that it is desired to extract the required baseline to.

Version : Default | Registered | Top

This should specify which version for each component of a design baseline should be extracted.

Related Commands

Check Out.

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Get Baseline to Directory allows the parts in a baseline to be extracted to any directory without being logged as a check-out and therefore without requiring the directory to be a workspace directory.

The files that are extracted are read-only and should be deleted when no longer required.

The baseline specified may name:

- A release baseline , in which case the versions defined in the baseline will be extracted
- A design baseline, in which the specified version (Default, Registered or Top) of the component defined in the baseline will be extracted.
- A meta baseline, in which case each baseline in the meta baseline will be extracted recursively

The files will be extracted to a directory hierarchy calculated using the baseline toppart as equivalent to the directory specified.

Any components or versions which are obsolete will *not* be extracted unless the Show Obsolete flag is set.

If a workfile already exists in the directory you will be prompted as to whether to overwrite the existing file.

If the directory is a workspace directory and the component is already checked out to that workspace then the checked out version may not be overwritten as this can interfere with AllChange's notion of what version is in a workspace.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Get_Version_to_Directory**)

call(get_baselines_to_selected_directory, baseline, *directory, what-version*)

Get Version to Directory

Function

Accessed from **Part | Get Version to Directory**. Allows versions of components to be extracted to any named directory without being logged in the check-outs database.

Class

Part.

Options

Part

This should specify the part to be extracted. A specific version may be specified, or a component in which case the specified **Version** will be extracted, or a subsystem in which case the specified **Version** of all components in the subsystem will be extracted.

Directory

This should specify the directory that it is desired to extract the required versions to.

Version: Default | Registered | Top

This should specify which version for each component should be extracted.

Related Commands

Check Out.

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Get Version to Directory allows parts to be extracted to any directory without being logged as a check-out and therefore without requiring the directory to be a workspace directory.

The files that are extracted are read-only and should be deleted when no longer required.

The parts specified may name:

- A version, in which case that version will be extracted
- A component, in which case the specified version (Default, Registered or Top) will be extracted.
- A subsystem, in which case the specified version of all components in the subsystem will be extracted to a matching hierarchy rooted from the destination directory. Any components or versions which are obsolete will *not* be extracted unless the Show Obsolete flag is set.

If a workfile already exists in the directory you will be prompted as to whether to overwrite the existing file.

If the directory is a workspace directory and the component is already checked out read only to that workspace then the checked out version may not be overwritten as this can interfere with AllChange's notion of what version is in a workspace.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Get_Version_to_Directory**)

call(get_parts_to_selected_directory, *version*, *directory, what-version*)

Import Baseline

Function

Accessed from **Baseline | Archive Baselines | Import**. Allows Baselines to be restored/ imported from an archive file.

Class

Baseline.

Options

Baselines
Specifies the Baselines that are to be imported.

Related Commands

Archive Baselines, **Delete Baselines**, **Creating Baselines**, **Modifying Baselines**,

Concepts

Baselines (Creating and Managing Baselines)

Description

Import Baseline allows Baselines which have been archived to an external file (`blarch.exp`) to be imported back into the database. Note that direct access to this file is required even if running client/server.

A dialog will be presented showing a list of Baselines in the archive for selection for importing.

On successful completion of the import you will be given the option to remove the imported baselines from the archive. Once removed from the archive they cannot be re-imported at a later date.

This is — by default — an Administrator only function. Furthermore, by default the Baseline archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

This is implemented as an ACCEL script in the function file `archfunc.ac`. Baselines are recreated with their original number and all their original information.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Import_Baselines**)

call(**ImportBaselines**, *archive-file*, *Baselines*)

Import CR

Function

Accessed from **CR | Archive CRs | Import**. Allows CRs to be restored/ imported from an archive file.

Class

CR.

Options

CRs
Specifies the CRs that are to be imported.

Related Commands

Archive CRs, **Delete CRs**, **Creating CRs**, **Modifying CRs**,

Concepts

CRs (Creating and Managing Change Requests)

Description

Import CR allows CRs which have been archived to an external file (`crarch.exp`) to be imported back into the database. Note that direct access to this file is required even if running client/server.

A dialog will be presented showing a list of CRs in the archive for selection for importing.

On successful completion of the import you will be given the option to remove the imported CRs from the archive. Once removed from the archive they cannot be re-imported at a later date.

This is — by default — an Administrator only function. Furthermore, by default the CR archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

This is implemented as an ACCELscript in the function file `archfunc.ac`. CRs are recreated with their original number and all their original information.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Import_CRs**)

call(**ImportCRs**, *archive-file*, *crs*)

Import CRs from CSV File

Function

Accessed from **CR | Import CRs from CSV File**. Allows CRs to be imported from a text file with comma delimited fields (a .csv file)

Class

CR

Options

File Name
The full path to the .csv file

Concepts

CRs (Creating and Managing CRs)

Description

Import CRs from CSV File allows CRs to be imported from a comma delimited text file (a .csv file). This facility is only available if enabled by the ***AllChange*** administrator. **Import CRs from CSV File** is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators

The first line in the CSV file should contain the names of the headings for each column separated by a comma (,).

You will be prompted for the class of CRs that are being imported so the class should not be specified in the CSV file and if it is it will be ignored.

Next, the user is asked whether they wish to map fields. If 'No' is selected, then the fields by using the values of the headings as the field names. For example, the headings for a CR import could be:

```
summary, ref, Locked, assignee
```

Each of these columns would be mapped to:

```
cr_summary cr_ref cr_Locked cr_assignee
```

If the user chooses to map the fields, then a dialog is displayed which allows the user to specify which column in the CSV file maps to which ***AllChange*** field. The CSV column is selected, using the name specified in the headings, and the corresponding ***AllChange*** field may be selected from a drop-list. Any heading names that match with field names will automatically default to map to the corresponding ***AllChange*** fields.

If a column's field is selected as **<None>** in the ***AllChange* Field** list, then that column will be skipped when importing.

Any new line characters found in a cell will disrupt the import, as a new line signifies a new row/record. You should ensure that new lines are only at the end of each row.

ACCEL Syntax

This facility is implemented via ACCEL:

call(ImportCSVCRs)

Import Part

Function

Accessed from **Part | Archive Parts | Import**. Allows Parts to be restored/ imported from an archive file.

Class

Part.

Options

Parts

Specifies the Parts that are to be imported.

Related Commands

Archive Parts, **Delete Parts**, **Creating Parts**, **Modifying Parts**,

Concepts

Parts (Creating and Managing Parts)

Description

Import Part allows Parts which have been archived to an external file (`paarch.exp`) to be imported back into the database. Note that direct access to this file is required even if running client/server.

A dialog will be presented showing a list of parts in the archive for selection for importing. if a subsystem is selected it will import the subsystem and all children recursively. If importing a subsystem then the parent part must already exist. This means that if /Product/Subsystem is archived and deleted that /Product/Subsystem/Subsys cannot be imported without /Product/Subsystem being imported.

On successful completion of the import you will be given the option to remove the imported parts from the archive. Once removed from the archive they cannot be re-imported at a later date.

This is — by default — an Administrator only function. Furthermore, by default the Part archiving facility is disabled regardless of user, see the ***AllChange*** *Administrator Manual* for how to enable the archiving facility.

This is implemented as an ACCELscript in the function file `archfunc.ac`. Parts are recreated with their original name and all their original versions.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Import_Parts**)

call(**ImportParts**, *archive-file*, *parts*)

Import Parts from CSV File

Function

Accessed from **Part | Import Parts from CSV File**. Allows parts to be imported from a text file with comma delimited fields (a .csv file)

Class

Part

Options

File Name
The full path to the .csv file

Concepts

Parts (Creating and Managing Parts)

Description

Import parts from CSV File allows parts to be imported from a comma delimited text file (a .csv file). This facility is only available if enabled by the ***AllChange*** administrator. **Import Parts from CSV File** is — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators

The first line in the CSV file should contain the names of the headings for each column separated by a comma (,).

For importing parts, the first column will be taken as the part name and the second as the subsystem to place the part in, regardless of what the heading names are. Parts will be imported rooted at the selected subsystem in the parts browser. If no subsystem is selected, you will be prompted for what subsystem to root the import to.

You will be prompted for the class of parts that are being imported so the class should not be specified in the CSV file and if it is it will be ignored.

Next, the user is asked whether they wish to map fields. If 'No' is selected, then the fields by using the values of the headings as the field names. For example, the headings for a part import could be:

```
name, subsystem, description
```

Each of these columns would be mapped to:

`pa_name` <*import-root-part*>`/subsystem pa_description`

If the user chooses to map the fields, then a dialog is displayed which allows the user to specify which column in the CSV file maps to which ***AllChange*** field. The CSV column is selected, using the name

specified in the headings, and the corresponding ***AllChange*** field may be selected from a drop-list. Any heading names that match with field names will automatically default to map to the corresponding ***All-Change*** fields.

If a column's field is selected as **<None>** in the ***AllChange*** **Field** list, then that column will be skipped when importing.

Any new line characters found in a cell will disrupt the import, as a new line signifies a new row/record. You should ensure that new lines are only at the end of each row.

ACCEL Syntax

This facility is implemented via ACCEL:

call(ImportCSVparts)

Import Parts from Subversion

Accessed from **Part | Import Parts from Subversion**. Allows parts to be imported from a Subversion repository together with their version history.

Class

Part

Options

Repository:

The URL to the Subversion repository

Concepts

Parts (Creating and Managing Parts)

Description

Import Parts from Subversion allows parts to be imported from a Subversion repository. This facility is only available if enabled by the ***AllChange*** administrator. **Import Parts fromSubversion is** — by default — an Administrator only function and will therefore only be available to ***AllChange*** users who are defined as Administrators.

You must be attached to a workspace and files will be extracted from the subversion repository into the workspace and then checked into ***AllChange***.

This is designed to be an initial import facility and once imported a part may not be imported again (unless the part is first deleted).

The URL to the subversion repository must be specified, e.g. svn://svnserver/mainproject. It is possible to specify the root of the repository or at any level within the repository. Files will be extracted from Sub-version to the workspace directory relative to the Subversion URL specified.

A browser will be shown showing the contents of the specified repository, select the files and/or folders to be imported into ***AllChange***.

By default the repository browser will fill the folder structure on expansion of each folder. This makes for a faster browser but it is not possible to see what folders have sub-folders until they are navigated into. To get all of the folder structure populated at the start, set the **SubversionFullTree configuration option** on the Features tab in Acconfig.

It will then prompt for the class to use for the new parts together with any compulsory arbitrary fields (with the exception of the comment). The class and arbitrary field values specified will be used for all of the parts/versions imported.

The comment field is filled in automatically from the Subversion information. By default the version arbitrary field called **Comment** (if defined) will be used to store the comment. This can be overridden by setting the configuration option **SubversionCommentField** on the Features tab in Acconfig. If the **SubversionCommentField** is not set or the **Comment** arbitrary field does not exist, the subversion comment will not be imported.

Each revision of the Subversion files will be imported as ***AllChange*** versions. The comment and the date of each revision is retained in ***AllChange***.

Note that in order for the date to be retained correctly the **Putaway storefile time** configuration option must be set, a warning will be given if this not the case. This must be set by your ***AllChange*** administrator.

This function is implemented as an ACCEL function in the function file svnfunc.ac.

ACCEL Syntax

This facility is implemented via ACCEL:

call(ImportFilesFromSubversion)

Insert Into Text

Function

Accessed from **CR | Insert Into Text, Baseline | Insert Into Text** or **Part | Insert Into Text**. Allows Text together with a date/time/author stamp to be appended to a CR, Baseline or Part Text Section.

Class

CR, Baseline, Part

Concepts

CRs (Creating and Managing CRs), Baselines (Creating and Managing Baselines), Parts (Creating and Managing Parts).

Description

Insert Into Text allows text to be appended to a Text Section together with a date/time/author stamp.

This is the only method of modifying the text for items which are of a class which is defined with the **Protect Text** attribute.

The item to be modified must be selected in either a browser or viewer.

The section that is to be appended to may be selected from a drop down list.

ACCEL Syntax | Command Line Syntax

This facility is implemented via ACCEL:

call(AppendCRTextPost), call(AppendBaselineTextPost), call(AppendPartTextPost)

call(InsertIntoItemText, *type*, *item, section, text*)

> where *type* is one of "Cr", "Baseline" or "Part", and *item* is the CR/Baseline/Part to append to, *section* is the name of the section to append to or create, and text is the actual *text* for the section.

Issue

Function

Accessed from **Part | Issue...** or via the toolbar. Checks out a copy of a part to the current workspace . Only available to ***AllChange*** administrators.

Class

Part

Options

Parts
Specifies the parts that are to be checked out.

Comment
Allows a comment to be specified giving the reason for the check out.

For Edit
Checks out the part for the purposes of editing, and will reserve a new version for the part.

New Branch
Will create a new branch with the specified name and reserve a new version along that branch. This is only allowed with the **For Edit** option.

Ignore if Already Checked out
If **Ignore if Already Checked out** is selected then no error will be generated for any parts specified which are already checked out to the current workspace.

optimistic
Causes parts to be checked out for edit using optimistic locking

`-noget`
This option is only available from the command line and is not presented as an option in ACE. This is only allowed with the `-edit` option and causes the workfile not to be created. This is useful when the workfile already exists (e.g. supplied by a third party).

Related Commands

Check Out, **Check In**, **Attach Workspace**, **Return Part**, **Promote Object**, **Alter Check-out**, **Alter Version**, **List**, **Report**.

Concepts

Part (CheckingFiles In and Out of ***AllChange***),

Description

Issue is the underlying ***AllChange*** command which is used by the more sophisticated **Check Out**function.

Issue Part allows a copy of a version of a component type part to be checked out to the user in the current workspace. The user must currently be attached to a workspace. It logs the check-out in the check-outs database until such time as that part is checked in (**return**ed).

If the part is not specified with a version-id the default version of the part is checked out, otherwise the version specified is checked out.

Specifying a subsystem among the parts to be checked out causes the **issue** to descend the parts tree recursively downward from that point checking out the default version of all components encountered. Note that only a version can be checked out and logged in the check-outs database; subsystems and components are not themselves logged as checked out. This makes checking out all components in a hierarchy very easy.

It is also permissible to specify `!!`*baseline-name* among the parts to check out. This checks out all versions in the baseline *baseline-name* (if the baseline is a meta-baseline it is followed down recursively). This is preferable to using *baseline-name*`!!*;*` which can run out of space expanding the wildcards.

If the part does not have the **no_file** flag set then a copy of the contents of the version is created in a workfile in the workspace.

If **For Edit** is not specified then the part is checked out for read purposes and may not be changed. This might be used, for example, for general reference, distribution, building or release purposes.

If the **For Edit** option is specified then the part is checked out for the purposes of making a change. If pessimistic locking is used (the default case) a new version is reserved for when that change has been made and the part is **return**ed (see Return Part). The new version will have a number 1 greater than the one checked out. If the **New Branch** option is specified the new version will be the first one on the branch specified instead. If optimistic locking is used (the **Optimistic** option is specified) then no specific new version is reserved and the new version is determined at return time instead. Optimistic locking used in combination with **New Branch** will cause the new version on return to be placed on the specified branch.

The command will be executed for each part specified. Only one version of a given part may be checked out to a particular workspace at any time; it must be checked in before a different version may be checked out to that workspace.

The file created as a result of an issue command (if any) will have a *time stamp* of either the time that the version was stored or the time that the file had when the version was stored depending on the **Putaway store file time** configuration option, see ***AllChange*** Administrator Manual.

Command Line Syntax

issue [`-edit` [`-branch` *branch*] [`-noget`]] [`-comment` *comment*] [`-notissued`] [-optimistic] *parts*

Examples

```
issue buildfile
```

checks out the default version of `buildfile` to the user in the current workspace, creating a corresponding workfile in the current directory and logging the check out in the check-outsdatabase. `buildfile` must not already be checked out to this workspace.

```
issue /product
```

checks out the default versions of all components anywhere within the `/product` hierarchy to the user in the current workspace, creating corresponding workfiles and logging the check outs of the versions (only) in the check-outs database.

```
issue -edit /product/source/*.bas
```

checks out the default version of all components ending in `.bas` which are children of `/product/source` to the current workspace with the intention of editing them. If the version of `file1.bas` checked out is `file1.bas;009` then `file1.bas;010` is reserved for a later check in (**return**).

```
issue -edit -branch exp1 -comment 'Experimental version'
module.c;!release_1
```

checks out for edit whatever version of *current-working-part*`/module1.c` appears in baseline `release_1`. The new version reserved will be `module1.c;exp1.001`. The comment is stored in the check-outs database (the quotes are necessary in ACC— but not ACE — since the string contains a space).

```
issue !!release_1
```

checks out all versions referenced by baseline `release_1`.

Keywords

Function

Keywords shows Version Control keywords embedded in files.

Class

File.

Related Commands

Check Out, **Stamping Word Documents**

Concepts

Files (File Operations)

Description

Keywords examines *any* type of file looking for VC keywords. Every time it finds a VC keyword/ letter it displays it in the output window, together with the substitution text following the keyword. See Stamping Text Files for a description of VC keywords.

The purpose of **Keywords** is to identify what versions of parts were used to construct the file being examined. For example, an executable file may be the result of compiling several distinct source modules. If the user includes VC keywords in the source in such a way that they will appear in the corresponding object file, e.g. in a literal string, and this in turn is included in the final executable then running **Keywords** on the latter will display each component module's keywords. Suitable keyword for this purpose are `$ACpart` and `$ACversion$`. Hence in C each source module might contain:

```
static char vcid[] = "$ACpart:$ $ACversion:$";
```

Notes

The **Keywords** function invokes an underlying VC tool called **vcident**.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**File_Keywords**)

command(vcident *files*)

Merging

Function

Accessed from **Part | Merge** or via the toolbar. Applies the changes made between 2 specified versions of a part into the version currently checked out for edit.

Class

Part

Options

Version 1
Specifies the version of a part to compared against.

Version 2
Specifies the version of a part to compare **Version 1** with.

Related Commands

Differencing.

Concepts

Part (CheckingFiles In and Out of ***AllChange***).

Description

The Merge facilities in ***AllChange*** allow the changes between 2 versions to be applied to the workfile for the version currently checked out for edit.

The functionality is implemented by an external merge tool called **VisMerge** which takes 3 files and calculates and interactively displays the results of applying to *leftfile* the changes that lead from *middlefile* to *rightfile*. The 3 files are supplied to the tool from ***AllChange*** as follows:

leftfile	is the workfile of the version checked out for edit
middlefile	is version 1
rightfile	is version2

Results are presented as a scrollable table of lines with colours used to indicate merged text and leftfile clash text, rightfile clash text and resolved text.

Figure 53.4 shows what the display looks like and shows the result of merging two independent sets of changes to the rhyme shown in figure 53.1.

Figure 53.1: Middle File

```
Hickory dickory dock
The mouse ran up the clock
The clock struck one
The mouse ran down
Hickory dickory dock
```

This is presented to the merge tool as `file2`. `file1` contains a change to line 3 and removes the last line as shown in figure 53.2.

Figure 53.2: Left File

```
Hickory dickory dock
The mouse ran up the clock
The clock struck two
The mouse ran down
```

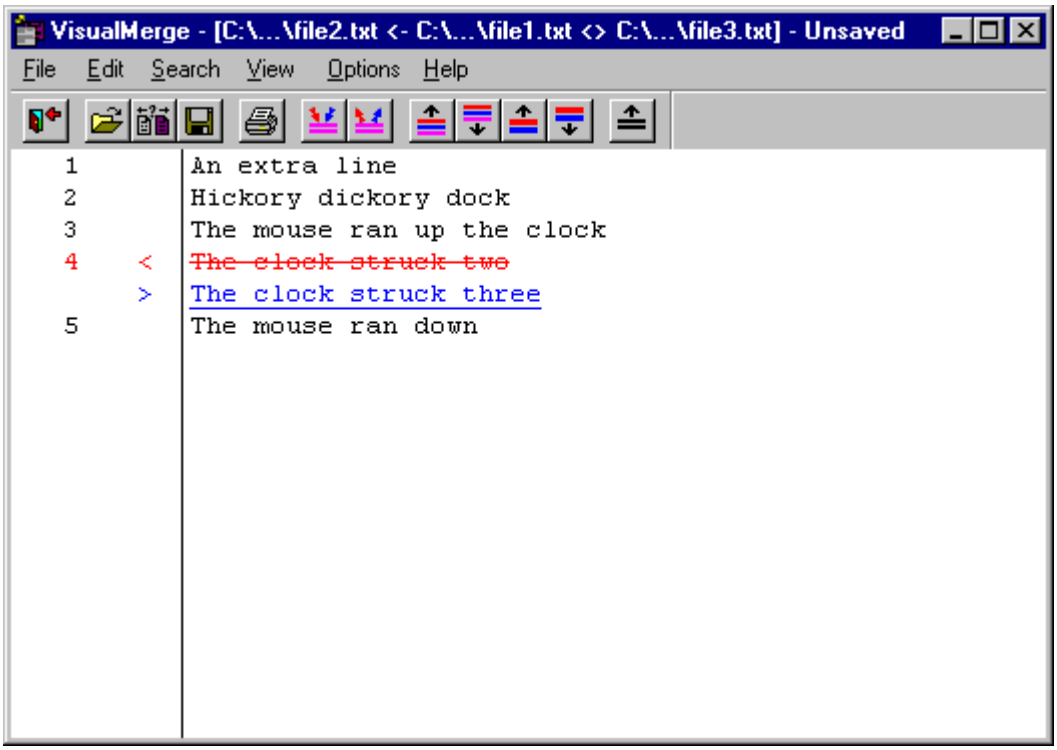
`file3` contains a different change to `file2`, namely the addition of a line at the beginning and a different change to line 3 as shown in figure 53.3

Figure 53.3: Right File

```
An extra line
Hickory dickory dock
The mouse ran up the clock
The clock struck three
The mouse ran down
Hickory dickory dock
```

Since both `Left File` and `Right File` changed line 3, this causes a *clash* as shown in figure 53.4.

Figure 53.4: Vismerge display



A *clash* occurs whenever both *leftfile* and *rightfile* have changes in the same section . The user is required to make a decision as to what changes are required in order to resolve the clash. This may be done by modifying the text as required and the merge tool allows this to be performed in the clash resolution editor.

The results of the merge may be saved in text format if all clashes are resolved; otherwise the results may be saved in an intermediate merge format which may be read in by **Vismerge** at a later time.

Facilities to locate each clash and match are provided on the **Search** menu and options to allow the user to tailor the presentation of the results are provided on the **View** menu.

Files are split into lines, compared on a line-by-line basis and the user is shown for each clash the lines from *leftfile* and those from *rightfile*. Note that even if just a single word, or a single character, differs in corresponding lines in the two files **Vismerge** shows the complete line as a clash.

The Toolbar

The Toolbar allows the most frequently used commands to be accessed directly without having to navigate through the selection menus. Each button on the toolbar executes a command directly.

The toolbar provides the following functions (from left to right):

Exit	Exit **Vismerge**.
Open Files	Open files and/or releases.
Redo Merge	Re-perform merge.
Save Results	Save results in text or merge format.
Print	Print.
Prev Clash	Find previous clash.
Next Clash	Find next clash.
Prev Match	Find previous match.

Next Match	Find next match.
Find Text	Find text.

Specifying Input Files

The files to be merged may be specified in **Vismerge** using the **File | Open** dialog. Either a single merge format file with a `.mrg` extension or 3 text files may be specified.

The dialog allows 1 merge format file or 3 text files to be specified. Opening a merge format file allows you to continue resolving clashes which resulted from a previous merge of 3 files.

The **Open** dialog cannot be accepted (with **Open**) until enough items have been specified. When the first file has been selected (by double clicking or **Open**) its path is displayed in **Left File** and second file may be selected (if the extension is not `.mrg`). The second file's path is displayed in **Middle File** and a third file may then be selected.

Saving Results

The results of a merge may be saved using the **File | Save** or **File | Save As** menu options.

If there are no unresolved clashes the output is saved as text. If there are any unresolved clashes the output is saved in merge format and the file is given the extension `.mrg`. Saving in merge format allows you to continue resolving clashes at a later date continuing from where you got to when the file was saved. In order to continue with a previous merge you may **Open** a `.mrg` file instead of opening three files.

Searching

Menu (and toolbar) items are provided to search for the **next**/ **previous** occurrence of a **difference**/ **match**. **Next** searches forward from the currently selected line and **previous** backwards. If there is no currently selected line the search starts from the top line on view.

A menu (and toolbar) item is also provided to search for text.

View Options

The **View** menu allows the results of a merge to be displayed in different ways.

View Lines Options

The options in the first group on the **View** menu select which lines of the resulting merge are displayed.

Merged Text and Clashes:	show both merged text and clashes (default)
Clashes Only:	show just the lines which are clashes
Merged Text Only:	shows just the merged text and no clashes
Merged Text and Left Clashes:	shows the merge text and lines from the left file only where there are clashes
Merged Text and Right:	shows the merge text and lines from the left file only where there are clashes

The options in the second group on the **View** menu affect how resolved clashes are shown.

Resolved or Else Clashes:	shows the resolution if a clash has been resolved, otherwise the clash is shown
Resolved and Clashes:	shows the resolution if a clash has been resolved together with the clash
Only Resolved:	shows only resolved clashes
No Resolved:	shows only clashes even if they have been resolved

View Line Numbers Options

The options in the third group on the **View** menu select how line numbers are displayed.

Left Line Numbers:	shows line numbers for the left file
Right Line Numbers:	shows line numbers for the right file
Resolved Line Numbers:	shows line numbers for the resolved output file
No Line Numbers:	shows no line numbers (default)

View Location Option

This option causes the file location character to be displayed for each line. Lines from the left file only are marked with a <, lines in the right file with a >, lines that have been resolved by the user with a `1` and lines which were automatically merged are left unmarked.

Use Strikeout and Underline Option

This option controls whether strikeout and underline are used to indicate the location of lines (it is enabled by default). Clash lines that come from the left file are shown with strikeout. Clash lines that come from the right file are shown with underline. Resolved lines are shown with strikeout and underline.

Merging Options

The various options affecting the **Vismerge** tool are available on the **Options** menu.

Ignore Blanks

This options causes blanks and whitespace in the files to be ignored when computing the differences between the files in order to calculate the merged result.

Save Settings on Exit

This option causes all options set to be saved when **Vismerge** is exited. The settings are saved in the registry.

Any settings saved are always restored when **Vismerge** starts.

Resolving Clashes

A clash is resolved (or edited) by double clicking (or selecting a line and using `Enter`) on a line in one of the 3 sections in a clash group (left, right or resolved). The resulting dialog contains the text of the section selected which may then be edited; if the clash has already been edited this is the resolved text; otherwise it is the left or right clash according to which was selected. The left or right clash text may be inserted using the **Ins Left** or **Ins Right** buttons;this might be used, for example, if the left clash text has been copied (because it was selected) and you want to paste in the right clash text as well for editing.

A previously resolved clash may be unresolved (i.e. whatever text was typed in as the resolution is discarded and the section reverts to an unresolved clash) by using **Edit | Unresolve** (or the toolbar button).

Command Line Options

Vismerge may be invoked from the command line as follows:

vismerge [*options*] [*file1*] [*file2*] [*file3*]

where *options* may be:

- **c** View location. This option may be disabled by following the option letter with a - (minus) character.
- **k** Use strikeout and underline. This option may be disabled by following the option letter with a - (minus) character.
- **l** View line numbers. This option must be followed by one of the letters **L**, **R** or **N**.
- **o** Output filename. This should be followed by the name of the file to contain the result of the merge.
- **t** Set window title.
- **v** View lines. This option must be followed by one of the letters **T**, **O**, **M**, **L** or **R** and **E**, **V**, **Y** or **S**.
- **x** Save settings on exit. The state of this option is always saved on exit unless this option is followed with a letter **D** on the command line. In addition the **D** parameter prevents any settings being saved when **Visdiffs** is exited.
- **-f** Exports the resulting differences to a file instead of presenting them interactively. This option should be followed by the export filename. If this has the extension `.rtf` then the results will be exported as RTF, otherwise they will be plain text.

Exit Codes

Mergeuses exit code 0 to indicate that there were no clashes while merging the files, exit code 1 if there were clashes and exit code 2 for problems (e.g. file not found).

Notes

The algorithm used by **VisMerge** is sophisticated and usually combines the files in precisely the manner desired. However, the output should be checked to ensure that nothing unexpected has occurred: it might be a good idea to use the **VisDiffs** tool to compare the new version with one of the originals to verify what precisely has been done.

VisMerge splits very large files into more manageable chunks which are then compared against each other. Even so it may require a fair amount of memory to operate successfully.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Merge**)

call(**File_Merge**)

Monitor Event

Function

Accessed from **Monitor | Add Monitor...** or via the toolbar. Allows a monitor to be placed on an item for a particular kind of event.

Class

Monitor

Options

Item Type
Specifies the type of item to be monitored.

Event
Specifies the event to be monitored. The available events will vary depending on the **Item Type**.

	use		

cr	cr	statuscr	
baseline	baseline	statusbaseline	
file	promote	release	
part	part	status	newversion
	use		

Any

If selected causes the event to be placed on *any* item of the type specified

Item

Specifies the objects that are to be monitored for the event. The items will be **Parts**, **Baselines** or **Files** according to the **Type** selected.

Arb1

This will usually be used to specify the specific event required for one of the general events (part, baseline and cr). Allows information to be assigned to the arbitrary field. If used this field should have been assigned a meaningful field name by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the menu.

Related Commands

Delete Monitor.

Concepts

Monitors.

Description

Monitor Event allows the addition and removal of monitors on items for a particular event. Each time the event occurs on that item, the placer of the monitor will be informed (mailed).

A monitor will be placed on the specified event for each item specified. The items to be monitored will be those specified unless **Any** is selected. For all parts and files the items may be specified using wildcards and/ or fetch files (operating system wildcards for files and part wildcards for parts — see Part Paths). The wildcards will be expanded, and default versions for parts specified with a partial version-id resolved, prior to placing the monitors (as opposed to when the monitored event takes place). A monitor placed on a component will cause the user to be informed any time the event occurs on that component or any of its versions; a monitor placed on a version will be activated only when the event occurs on that version.

Information about all monitors which have been placed is stored in the monitors database and may be examined using the **List** or **Report** commands.

Monitors can be used, for example, to keep track of files promoted to a pool which may affect your own work, or to keep track of any new versions of a part which are created.

Command Line Syntax

monitor `event=`*event* `user=`*user* [*arb1*=*value*] *items*

Examples

```
monitor event=baseline Release2
```

will mean that the user is informed any time a change is made to Release2 baseline that has been configured to be monitored.

```
monitor event=newversion *.c *.h;
```

will mean that the user is informed any time somebody creates a new version of any `.c` components or the current version of any `.h` components in the current working part. The `.c` and `.h` components are all those that are in the current working part at the time the **monitor** command is invoked, as is the default version .

```
monitor event=statuscr CR00050 CR00051
```

monitors any changes to the status of CRs 50 and 51.

```
monitor promote c:\pool1\*.obj
```

monitors all existing `.obj` files in pool directory `c:\pool1` to see whether a new version is promoted.

Move Field

Function

Accessed from **Misc | Administrator Tools | Move Field**. Allows data in an arbitrary field to be moved from one arbitrary field to another.

Options

Database
The database for the arbitrary field to be moved.

Class
The class for the item whose arbitrary field is to be moved.

From Field Number
The number of the arbitrary field data to be moved. This may be a number between 1 and 40 for all databases except CR, and 1 and 100 for the CRs database.

To Field Name
The name of the arbitrary field to which the data is to be moved.

Related Commands

Clear Field, Fix Field Values

Description

Move Field allows data in an arbitrary field to be moved from one arbitrary field to another.. This is useful if an arbitrary field definition is changed from one arbitrary field to another. Any data existing in the original field should be moved to the new field and the original field data should be set to blank (to ensure that if that field is redefined at a later date it does not inherit old invalid data). Move Field does both of these actions.

Move Field should be used after the field definition has been changed. The **From Field Number** is then the number of the arbitrary field originally used to hold the data, and the **To Field Name** is the name of the field (as newly defined).

This function is implemented as an ACCEL script in the function file admintools.ac. By default this is disabled and available only to administrators. See the Administrator manual for how to enable the admin tools.

Command Line Syntax

call(MoveFields)

Move Check-outs to Branch

Function

Accessed from **Workspace | Move Check-outs to Branch**. Allows parts checked out for edit to change the reserved version to a branch or from a branch to the trunk.

Class

Part

Options

Part
The parts which are to be affected by the move.

Branch
The branch to which the check out for edit is to be moved. Specify blank/empty to move from a branch to the trunk.

Concepts

Parts (Checking Files In and Out)

Description

Move Check-outs to Branch allows parts checked out for edit to a branch to be moved to the trunk, or on the trunk to a branch. This allows, for example, a check out for edit on an experimental branch to be moved to the main line when the experiment is successful and to be incorporated into the main line of development. It also allows a check for edit on the main line of development to be diverted to a branch should, for example, an emergency fix be required on the main line.

If a newer version than the version that was checked out exists then a warning is issued that a merge is required and the move is not possible.

If a branch which already exists is specified then an error will be issued.

ACCEL Syntax

This facility is implemented via ACCEL:

call(MoveIssuesToBranch)

Move Part

Function

Accessed from **Part | Move Part** or a context menu or the toolbar. Allows a part to be moved (either renamed within the same subsystem or moved to a different subsystem).

Class

Part

Options

Old Part
Specifies the parts that are to be moved

New Part
Specifies the new name and/ or path required. If a plain name is specified then the part will be renamed in the same subsystem. If a full path is specified then the part will be moved to the specified subsystem and given the name specified.

Change References

This allows any links/ references to the part being moved in baselines, CRs or other parts to be modified to refer to the new part.

Baselines: if selected will cause any references to the part being moved in baselines to be changed to refer to the new part.

CRs: if selected will cause any references to the part being moved in crs as parts affected or versionssolved to be changed to refer to the new part.

Parts (parts affected): if selected will cause any references to the part being moved as a partaffected of another part, to be changed to refer to the new part.

Parts (uses): if selected will cause any usage relationships which refer to the part being moved to be changed to refer to the new part. *Note that this could be very slow, only select it if usage relationships are in use*

It may not always be desirable to modify all links as this may be regarded as historical information which is required to be maintained.

The following options only apply if moving a subsystem and are checked by default if a subsystem is selected.

All Baseline Top Parts (subsystem name change): If selected, all baselines will be looked at to see if their top part contains the old subsystem that has been moved.

All CR Top Parts (subsystem name change): If selected, all CRs will be looked at to see if their top part contains the old subsystem that has been moved.

Without these options, only baselines and CRs that have parts affected will be altered. With theses options, all baselines and CRs will be changed, regardless of wether they have parts affected or not.

A warning will be given for any parts that have been adjusted and no longer fall under the baseline or cr top part. The (new) part will remain on the CR or baseline, but will not be under the baseline's or CR's top part. These items will need further attention by the user.

Old Part

This allows the action taken on the *Old Part* to be specified. It may be either **Delete**d so that the result is a *move*, or it may be simply made **Obsolete**. If **Mark as Obsolete** is selected then the **Old Part** and all its descendants will be marked as obsolete. This is useful if the original naming/ organisation of the parts is to be retained for historical purposes.

Related Commands

Rename Part, **Delete Part**, **Copy Part**,

Concepts

Parts (Creating and Managing Parts).

Description

Move Part allows a part to be moved (either renamed within the same subsystem or moved to a different subsystem).

Links/ references to the part moved may be selectively modified to the new part or retained for historical purposes. Similarly the original part may be either deleted or marked as obsolete.

If any component moved is of a class which requires a CR then this will be prompted for and all components moved associated with the CR as a part affected and all versions of the component associated with the CR as versions solved. If the **Old Part** is made obsolete then this will also be associated with the CR.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**MovePart**)

Allows a part to be moved (either renamed within the same subsystem or moved to a different subsystem).

New CR

Function

Accessed from **CR | New...** or via the toolbar. Allows a new change request (CR) to be created.

Class

CR

Options

[General]

Class

May be any of the defined classes for CRs and is used to determine the CRs life-cycle .

ID

Used for the user-supplied section , if any, of the new CR number to be generated. This option may be compulsory, optional or forbidden depending on the CR numbering format for the new CR (which in turn depends on the CRs class).

Summary

May be used to summarise the reason for the CR.

Top Part

Names the topmost part affected by this CR.

Reference

May be used for a reference.

Arb1...Arb100

Allows information to be assigned to the arbitrary fields 1 to 100. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the menu.

`status=`*status*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the status of the CR to be set to *status* with no conditions or actions being invoked.

`originator=`*originator*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the originator of the CR to be set to *originator* rather than using the user ID of the current user.

`assignee=`*assignee*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the assignee of the CR to be set to *assignee*. Note that no actions associated with assigning a CR will take place (e.g. the assignee will not be emailed).

`number=`*number*

This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change*** *Administrator Manual*. This allows the number of the CR to be set to *number*. *Number* must

not already exist; the auto-incrementing number, if any, usually used when creating a CR is not altered.

`date=`*date*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date that the CR was created to be set to *date*.

`text=`*filename*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the CR to be taken from the filename specified.

[Text]

Specifies the actual text of the change request. The ***AllChange*** Administrator will probably have set the system to use a template for the CR text.

[Items Affected]

Parts
May be a list of the specific parts that are affected by this CR.

Baselines
May be a list of baselines that are affected by this CR.

CRs
May be a list of (other) CRs that are affected by this CR.

Attachments
May be a list of files which are to be attached to this CR.

Create as link
If selected will create any attachments specified as *links*, otherwise they will be created as copies. Note that if link attachments have been disabled in the configuration options this option will be disabled.

Related Commands

Change CR Status, **Alter CR**, **Assign CR**, **Rename CR**, **CR Text**.

Concepts

CRs (Creating and Managing Change Requests).

Description

New CR creates a new Change Request (CR).

Each CR is given a "number'' to identify it; this is used as the reference to the CR for other CR commands, reporting etc. CR numbering is discussed in Creating and Managing Change Requests; what numbering scheme is used is site-defined and depends on the new CR's class. The default numbering scheme is (an auto-incrementing) 5 digit number. If the class of CR to be created has a user-supplied section in its number the **ID** option should be used to supply that part of the number (if the whole of the CR number is user-supplied then the **ID** option should be used to supply the whole identifier); if it has no user-supplied section then the **ID** option should not be used at all.

There is also a reference field which may be used as a site specific identification for the CR. This field is set by the **Reference** option; it may then be used in conditions for selecting CRs for those commands where a condition may be specified; it has no significance to the system.

Any files which are to be attached to the CR may be specified, these will then be copied to a secure area and associated with the CR *unless* the **Create as link** option is specified in which case the attachment will be created as a link.

Any parts affected by the CR may be specified; relative part names and wild cards may be used to specify the parts. This field may then be used, for example, to cause the CR to *trigger* the life-cycles of the parts affected by the CR when the CR is actioned. Any baselines affected by the CR may also be specified: the full baseline names should be given. Any other CRs affected by the CR may also be specified: this may be used to group related CRs together into "tasks".

The information provided as options to the command is stored with the CR together with various other information such as the date, the user etc (see Creating and Managing Change Requests).

CRs have a life-cycle associated with them which depends on their class just like parts — see Creating and Managing Change Requests; Each CR has a current status which may be changed as the CR progresses through its life-cycle (see Status CR). On creation of a new CR the status of the CR is set to the initial status of the CR cycle.

All information associated with a CR may be accessed for reporting/ querying purposes using the **List CR** and **Report CR** commands.

The actual text of the change request, which may include any other fields of information required together with a full description of the change requested, may be edited, retrieved and stored — see CR Text.

CRs may also be assigned to users — see Assign CR.

Command Line Syntax

newcr [id=*id*] [toppart=*part*] [ref=*string*] [class=*string*] [summary=*string*] [partsaffected=*part-list*] [blinesaffected=*baseline-list*] [crsaffected=*cr-list*] [[-linkfiles] filesaffected=*file-list*] [*arb1*...*arb100*=*value*] [assignee=*assignee*] [status=*status*] [originator=*originator*] [date=*date*] [number=*number*] [text=*file-name*]

Examples

```
newcr
```

creates a new CR in the CRs database.

```
newcr ref=ABC123 class=CR_bug 'summary=Program crashes'
partsaffected=header.h
```

creates a new CR with a site-specific reference of ABC123, a class of CR_bug, a summary of Program crashes (the quotes are necessary in ACC — but not ACE — because the string contains a space) and affecting part header.h in the current working part.

```
newcr class=task 'crsaffected=00003 CR00007 BUG-00010'
```

creates a new CR of class of task which affects CRs 00003, CR00007 and BUG-00010 (the quotes are necessary in ACC — but not ACE— because the string contains a space).

```
newcr class=fault id=Ext/12345-ABC
```

creates a new CR of class fault with the string Ext/12345-ABC being used to generate the new CR's "number". Depending on the CR numbering format associated with class fault this may form the whole of the CR's number or they may be further fixed characters prepended and/ or appended to it.

New Instance

Function

Accessed from **Part | Instance | New Instance**. Allows new instances of a version of a part to be created

Class

Part

Options

Version
Specifies the version for which the new instances are to be created.

Quantity
The number of instances to be created.

Arb1...Arb40
Allows information to be assigned to the arbitrary fields. If used these should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

Related Commands

Alter Instance, Delete Instance

Concepts

Parts (Creating and Managing Parts)

Description

New Instance allows new instances of a version of a part to be created. New Instances created will be automatically numbered. The number of digits used for the instance id may be modified by the ***AllChange*** administrator in the Configuration Options.

The user who creates the instances and the date/time the instances are created are automatically recorded in the instances.

Command Line Syntax

newinstance `-quantity` *quantity* [*arb1..arb40=value*] *version*

Examples

```
newinstance -quantity 10 Serial_Number="123456"
      "/Car/Accessories/CD Player;002"
```

Creates 10 new instances of version 002 of /Car/Accessories/CD Player with Serial Number 123456

New Report

Function

Accessed from **Report | New**. Allows a new ACREPORT to be created.

Class

Info

Options

Report Type
This may be either **Simple Table** or **Generic**.

Database
This specifies which database is the primary target of the report

Populate from Browser
This is only available for tabular reports and allows an initial report layout to be created based on the browser columns selected in ACE for the database.

Concepts

Description

New Report allows a new **ACREPORT** to be created using a report create creation wizard.

You are presented with a series of screens asking you to specify the type of report that you wish to create and the database that the report is to provide information from.

This is a part of the functionality of the **ACREPORT** report generator which is fully documented in **ACREPORT**

New Version

Function

Accessed from **Part | New Version ...** or via the toolbar. Allows a new version of a component type part to be created.

Class

Part

Options

Parts
Specifies the parts from which a new version is to be created. These must refer to versions unless the initial version is being created.

No Action
Specifies that no file management commands should be issued (i.e. no actions on external physical files).

No Initial File
May only be used if the new version created is the first version of a component. Without this option the contents of the corresponding workfile in the current workspace are stored as the initial contents of the first version. Specifying this option causes an empty version to be stored instead.

Optimistic
When used in combination with **No Version** causes the new version to be created as per optimistic locking on a checkout for edit.

Baseline
Specifies the baseline (if any) which should be updated to contain the new version created. Note that if no new version is created due to there being no differences, then the baseline will be updated to the version from which the new version was to be created.

Branch
Specifies that a branch with the specified name should be created for the new version.

Varb1...Varb10
Allows information to be assigned to the arbitrary fields. If used these fields should have been assigned meaningful field names by the ***AllChange*** Administrator; the assigned field name may be used to specify the arbitrary fields in ACCand will be shown in ACEin the dialog.

No Flags
Specifies that the version created should have no flags set (i.e. it is a full new version which is under version control).

No Version
Specifies that the version created should have the **NoVersion** flag set (like **check out for edit**).

`-duplicate`
Specifies that the version created should have the **DuplicateVersion** flag set (like **status** when

moving into a status which creates a new version). This argument is only available from the command line and is not presented as an option in ACE.

`status=`*`status`*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the status of the new version created to be set to *status* with no conditions or actions being invoked, instead of the initial status in the life-cycle being used.

`symbname=`*`symbname`*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the symbolic name of the new version created to be set to *symbname*, instead of it being calculated automatically.

`user=`*`user`*
This argument is only available from the command line and is not presented as an option in ACE . This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user who created the new version created to be set to *user*, instead of the user id of the person invoking the command.

`date=`*`date`*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the date of the new version created to be set to *date*, instead of the system date being used.

`-noget`
This option is only available from the command line and is not presented as an option in ACE. This causes the workfile not to be extracted. This is useful when the workfile already exists (e.g. supplied by a third party).

`-newversionid` *`versionid`*
This option is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the version to be created to be specified (and should be used with caution).

`ver_text=`*`filename`*
This argument is only available from the command line and is not presented as an option in ACE. This allows the text for the version to be taken from the filename specified.

Related Commands

Add Part, **Check Out**.

Concepts

Parts (Creating and Managing Parts, Checking Files In and Out of AllChange) .

Description

New Version allows a new version of a component type part to be created. A corresponding version will be created in the external version control file (unless **No Action** is specified) storing the contents of the corresponding workfile.

This command should not normally be used, since new versions will usually be created via the **Check Out** and **Check In**dialogs. To prevent its accidental use the user must have **dbadmin** permission on the part.

This command is useful for creating the first version of a component which was previously added using the argument stating that no initial version was to be created. It may also be used to create a new version of a component (including storing it in the version control file) which has been obtained without following the usual **check out for edit**/**check in** cycle, e.g. when new versions of files are simply supplied by the outside world.

Command Line Syntax

newversion [`-noaction`] [`-branch`*branch*] [`-baseline` *baselinename*] [`-comment` *comment*] [`-noinitial`] [`-noversion` | `-duplicate`] [`-noget`] [*varb1*...*varb15*=*value*] [`status=`*status*] [`user=`*user*] [`symbname=`*symbname*] [`date=`*date*] [`-newversionid` *versionid*] [`ver_text=`*filename*] *parts*

Examples

```
newversion / subsystem /component
```

creates the initial version of `/subsystem/component`; the component had previously been added without any first version.

```
newversion -noaction /subsystem/component;004
```

creates a new version of `/subsystem/component` to follow version 004. No operating system commands are issued.

```
newversion /subsystem/component;004
```

creates a new version of `/subsystem/component` to follow version 004. The contents of the corresponding workfile are stored in the version history file.

Open File

Function

Accessed from **File | Open** or context menu invokes the *open* action for the selected file.

Class

File

Options

Files

Specifies the files to be opened.

Related Commands

Edit.

Concepts

Part (Checking Files In and Out of ***AllChange***)

Description

Open invokes the *default* action for the files selected. This may be the **open** action but does not have to be.

This is implemented as an ACCEL function which may be tailored to site requirements.

ACCEL Syntax

call(**open_sel_files**)

Open Report

Function

Accessed from **Report | Open**. Allows an ACREPORT format file to be opened for modification purposes or for running the report.

Class

Info

Options

Report Format File
This specifies the report format file to be opened

Concepts

Reports and Queries

Description

Open Report allows an **ACREPORT** to be opened.

Once opened the report may be modified or it may simply be executed.

This is a part of the functionality of the **ACREPORT** report generator which is fully documented in **ACREPORT**

OS Command

Function

Accessed from **Misc | OS Command...** or via the toolbar. Allows an operating system command to be issued from within the ***AllChange*** system.

Class

Misc

Options

Command
Specifies the command to be issued.

Files
Specifies the files on which the command will operate.

Paged
This should be selected only if the external command is not interactive, This will allow ACE to capture the output of such a program and display it in the Output window.

Description

OS Command allows any arbitrary operating system command may be issued from within the ***AllChange*** command line interface or from ACE without exiting the system.

If an interactive operating system command is invoked with the `Paged` field set results are extremely unpredictable. The most likely result is that you will fail to get any output from the external command and it will then wait for some input from the keyboard.

Command Line Syntax

command *os-command*

Examples

```
command dir
```

lists the contents of the current directory .

Part Text

Function

Allows the text of a part to be edited, imported and exported to file.

Class

Part

Related Commands

Add Part.

Concepts

Parts (Creating and Managing Parts).

Description

Part Text allows the text of the specified part to be extracted to a file, edited and returned to the system. Three ***AllChange*** commands are available for editing, importing and exporting the part Text to file.

These are only available via the ***AllChange*** command line

The text of a part may be any arbitrary text. When the part was created, the ***AllChange*** Administrator will probably have set the system so as to create a template for the text. This should be completed as required by your site.

`editpatext` invokes your editor on the text file corresponding to the specified part.

`getpatext` will retrieve the text of the part from the system and place it in an external file.

Unless otherwise specified, the name of the file is generated from the part name. The file name is the name of the part with any punctuation characters removed, e.g. the text for part `/product/sub1/part.txt;001` would be placed in a file named `productsub1parttxt001`. The rules are explained in detail in the ***AllChange*** Administrator Manual.

Unless otherwise specified the file will be created in the current directory, and may then be examined/ altered using any tools available.

The part text may be returned to the part database when changes have been made using `putpatext`.

If the Part text was only retrieved in order to examine it, when finished with it the file may simply be removed.

`putpatext` updates the text of a part to that from a file.

It takes the text of each part specified from a file and replaces the existing part text, if any, with this. The filename used will be computed in the same way as `getpatext` unless otherwise specified.

Command Line Syntax

editpatext part-*names*

getpatext [`-file` *filename*] *part-names*

putpatext [`-file` *filename*] *part-names*

Examples

```
editpatext /product/sub1/part.txt;001 /document.doc
```
extracts, invokes your editor on, and then replaces the text of the specified components or versions.

```
getcrtext /product/sub1/part.txt;001 /document.doc
```
extracts the text of parts /product/sub1/part.txt;001 and /document.doc from the part database, placing them in files `productsub1parttxt001` and `documentdoc` respectively.

```
putcrtext /product/sub1/part.txt;001 /document.doc
```
stores the text of parts /product/sub1/part.txt;001 and /document.doc into the part database, taking them from files `productsub1parttxt001 and documentdoc` respectively.

Promote

Function

Accessed from **File | Promote...** or via the toolbar. Allows objects to be promoted from a workspace to a pool .

Class

File

Options

Files
Specifies files that exist in the workspace which are to be promoted (but see **Dependents**), and may be files that have corresponding parts; they may also be files such as derived objects (object and executable files) which may not have associated parts. Operating system wildcards and/ or fetch files may be used for specifying the objects.

Pool
Specifies the pool to which the objects are to be promoted.

Dependents
Specifies that all dependents of each object should be promoted.

Only Derived
Only promotes derived objects.

Allow Edits
Allows objects checked out for edit, or derived from checked out for edit, to be promoted to a pool. Note, however, that promoting objects which are checked out for edit is not compatible with the **build** command. This option implies **No Return**.

No Return
Prevents objects which are checked out read only from being checked in.

No Build
Prevents **promote** checking that the objects promoted are up to date.

No Build Threads
Allows objects which have no BT to be promoted. Any other options requiring the use of a BT (e.g. **Dependents**, **Only Derived**) will have no effect for objects for which there is no BT.

Related Commands

Attach Workspace, **Build**.

Concepts

Files (File Operations)

Description

Promote Object moves objects from a workspace to a pool. The objects will be moved to the pool together with their associated build threads (see Build) and will be set to have *read-only* access.

By default (i.e. unless modified by command line arguments) the following rules apply:

- objects which are derived from parts checked out for edit may not be promoted.
- objects which have been *checked out* (read only) are *checked in* before they are promoted.
- only objects which are built up to date may be promoted so that a consistent set of items are promoted to the pool.
- objects which do not have a BT may not be promoted.

Pools may be used for sharing of items amongst several workspaces and are used by build (e.g. object and header files).

Care should be taken if the **Allow Edits** option is used, as this will allow the promotion of objects for which no version has been stored, or objects derived from parts for which a version has not been stored. Such objects can therefore not be guaranteed to be reproducible.

Command Line Syntax

promote [`-depends`] [`-edit` [`-comment` *comment*]] [`-noreturn`] [`-derived`] [`-allowedits`] [`-nobuild`] [`-nobts`] ([`-pool`] *pool*) *objects*

Examples

```
promote -pool headers defs.h
```

promotes the workfile`defs.h` from the current directory in the current workspace to the directory associated with the pool `headers`. `defs.h` must be checked out (not for edit) to the current user and workspace; it will be checked in (so it is no longer checked out), and the file will be moved to the pool's directory (so it is no longer in the current directory). The BT for `defs.h` — which was placed in the current directory when it was checked out — will be promoted to the pool too.

```
promote -pool pool1 -depends -noreturn *.exe
```

promotes all `.exe` workfiles, together with all their dependents (object and source files), and all associated BTs to `pool1`. The `-noreturn` causes the files to be copied, rather than moved, to the pool's directory and source files which are checked out are not checked in. **promote** will check that the files are built correctly.

```
promote -pool pool1 -depends -derived -nobuild *.exe
```

promotes all `.exe` workfiles, together with all their *derived* dependents (object but not source files), to `pool1`. **promote** does not bother to check that the files are built correctly.

Quit

Function

Accessed from **File | Exit** or via the toolbar or via the windowing system's "close" button. Quits ***AllChange***.

Class

File.

Description

Quit exits ***AllChange***. It may be used in scripts.

Command Line Syntax

quit

Examples

```
quit
```

exits ***AllChange***.

Re-Read Configuration

Function

Accessed from **Misc | Re-Read Configuration** allows certain configuration files to be read when changes have been made by the ***AllChange*** administrator

Class

General

Options

configuration file
Specifies the configuration file to be re-read.

Description

This reads in the selected configuration file allowing changes to be accessed without restarting ***All-Change***.

Configuration files supported include:

- Workspaces
- Roles
- Pools
- Command Access
- Report Summaries

ACCEL Syntax

This facility is implemented via ACCEL:

call(ReadConfig)

read_roles()

read_rolemap()

read_workspaces()

read_pools()

read_report_summaries()

read_lockouts()

Read Dev Functions

Function

Allows administrators to re-read function definitions in specified ACCEL function definition files.

Class

General.

Related Commands

Read Functions,

Description

Read Dev Functions re-reads the function definition files selected from a prompt. This is useful to allow any modifications to the functions to take effect without exiting ACE and starting it again.

This function is implemented as an ACCEL function and therefore may be tailored to site specific requirements. It is defined in `command.ac`.

This function is not available from the menus as standard, but may be added to **Menu Item** using ACCONFIG. The function to call is **ReadDevFunc** and should be added with an **Action** of **CallUserFunc**, see the ***AllChange****Administrator Manual* for details.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**ReadDevFunc**)

Read Functions

Function

Accessed from **Function | Read Functions**. Allows administrators to re-read the function definitions in the standard ACCELfunction definition files.

Class

General.

Related Commands

Read Dev Functions,

Description

Read Functions re-reads the function definition files `cmdfunc.ac` and `projfunc.ac` for the current ***AllChange***project. This is useful to allow any modifications to the functions to take effect without exiting ACE and starting it again.

This function is implemented as an ACCEL function and therefore may be tailored to site specific requirements. It is defined in `commands.ac`.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**ReadFunctions**)

Release

Function

Accessed from **File | Release...** or via the toolbar. Copies releasable objects from a poolto a release directory.

Class

File

Options

Files
Specifies the objects that are to be released. The objects are filenames that will be found in the pool. Operating system wildcards and/ or fetch files may be used for specifying the objects.

Baseline
Specifies the name of the baseline in which the objects' components must appear.

From Pool
Specifies the name of the pool from which the objects are to be copied.

To Directory
Specifies the directory to which the objects are to be copied.

Include Build Threads
Causes the objects' build thread to be copied as well as the object.

Related Commands

Promote Object.

Concepts

Files (File Operations)

Description

Release Objects copies releasable objects from a pool to a release directory.

If a baseline is specified then each object released will be checked to ensure that it is composed of elements which have been baselined to ensure that the release is repeatable. This facility is only available if BTs are being used.

The BTs associated with each object released may also be copied to the release directory if BTs are enabled. This can be useful in providing a document for each object as to how that object was built and of what it is composed.

Command Line Syntax

release [`-bts`] ([`-baseline` *bname]*) ([`-from`] *pool*) ([`-to`] *release-dir*) *objects*

Examples

```
release -baseline release_1 -from pool1 -to K:\RELEASE
        P:\POOL\*.EXE
```

releases all `.EXE` files in `P:\POOL`, which is the directory associated with pool `pool1`, by copying them to the release directory `K:\RELEASE`. The BTs which reside in the pool with these objects are first checked to ensure they confirm that the `.EXE`s are correctly built from source files which appear in baseline `release_1`.

Rename Baseline

Function

Accessed from **Baseline | Rename...** or via the toolbar. Allows the name of a baseline to be changed.

Class

Baseline

Options

Old Name
Specifies the name of the baseline to be renamed.

New Name
Specifies the new name for the baseline.

Related Commands

Add Baseline, **Delete Baseline**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Rename Baseline changes the name of the baseline specified in **Old Name** to the name given in the **New Name**.

When a baseline is renamed the parts/ baseline in the baseline are retained on the newly named baseline. In addition any attachments are retained in the newly named baseline together with the status log.

Furthermore, any CRs or meta-baselines which refer to the baseline will now refer to the newly named baseline.

Command Line Syntax

renamebaseline ([`-from`] *frombline*) ([`-to`] *newname*)

Examples

```
renamebaseline Current Release6_4
```

changes the name of baseline `Current` to `Release6_4`.

Rename Branch

Function

Accessed from **Part | Rename Branch**. Allows a branch on all versions of a component or all versions of components in a subsystem to be renamed.

Class

Part

Options

Part
The path of a component, or subsystem which is to be searched for components, with versions on the branch to be renamed

Old Branch Name
The name of the branch which is to be renamed

New Branch Name
The new name for the branch

Just show proposed rename operations
If this is checked then the rename will not take place but information as to all proposed renames will be shown in the output window

Concepts

Parts (Creating and Managing Parts),(Checking Files In and Out)

Description

Rename Branch allows all versions on a branch of a component, or all components within a subsystem, to be renamed.

ACCEL Syntax

This facility is implemented via ACCEL:

call(RenameBranch)

Rename Part

Function

Accessed from **Part | Rename...** Allows the name of a component part to be changed.

Class

Part

Options

Old Name
Specifies the pathname of the part to be renamed.

New Name
Specifies the new name for the part. This must be just the name of the part and not a full path name.

Related Commands

Add Part, **Copy Part**, **Delete Part**.

Concepts

Parts (Creating and Managing Parts).

Description

Rename Part changes the name of the componentspecified in **Old Name** to the name given in the **New Name**.

The new name must *not* be a path, but just the new name for the part.

Only component type parts may be renamed. Unless a component has an explicit locationthe associated VC file is renamed to correspond to the part's new name.

When a component is renamed information held on the component referring to other parts as parts affected is retained in the newly named component together with the statuslog and versions of the component.

Furthermore, any CRs which refer to the component or any versions of it as parts affected by the CRor versions solved by the CR will now refer to the newly named component/ versions.

Note, however, that any baselines which include the component or any versions of it will *not* be modified to reflect the new name. This historical information is retained in the baseline should it desirable to revert to the state at the time the baseline was populated.

If it is desired to rename a subsystemtype part, then either **Move Part** must be used or the part must be copied to the new path name for the subsystem and the old subsystem deleted.

Rename Part does not require a CR to make this change regardless of the class of the part being renamed.

Rename Part is and administrator only facility and requires dbadmin permission by default.

In the normal case Move Part should be used to rename or move a component or subsystem, this will change references to baselines if required and is also subject to a Change Request to authorise the change depending on the class of the part moved.

Command Line Syntax

rename ([`-from`] *frompart*) ([`-to`] *newname*)

Examples

```
rename /product1/misspelt spelt
```

renames the part `/product1/misspelt` — which must be of type component — to `/product1/spelt`. Unless the part has an explicit location the VC file is renamed too.

Rename CR

Function

Accessed from **CR | Rename** or via the toolbar or context menu. Allows a CR to be renamed.

Class

CR

Options

Old name
This is the CR that is to be renamed. It will be taken from the currently selected CR

New name
This specifies the new name required for the CR.

Related Commands

New CR, **Alter CR**.

Concepts

CRs (Creating and Managing Change Requests)

Description

Rename CR allows a CRs name/ number to be changed. The new name specified *must* be unique.

Note that CRs may be renamed to any unique identifier, there is no check that the new name conforms to the numbering scheme defined for that class of CR, nor does it have any effect on the auto incrementing numbers.

When a CR is renamed the parts, CRs, baselines and attachments for the CR are retained in the newly named CR together with the status log.

Furthermore, any other CRs which refer to the renamed CR will now refer to the newly named CR.

This is implemented as an ACCELfunction and so may be tailored as required.

ACCEL Syntax

This function is implemented via ACCEL:

call(**AceRenameCR**)

call(**RenameCR**, *oldcrnumber*, *newcrnumber*)

Rename File

Function

Accessed from **File | Rename** or context menu. Allows a file to be renamed.

Class

File

Options

Old Name
Specifies the file to be renamed

New name
Specifies the new name for the file

Description

Rename allows a workfile to be renamed. Note that this does *not* rename any corresponding part .

ACCEL Syntax

call(**AceRenameFile**)

Report

Function

Accessed from **Reports | Report** or via the toolbar. Allows user defined reports to be generated.

Class

Info

Options

items
Specifies the items that are to be candidates for inclusion in the report. `any` may be used which causes all items in the database to be candidates. Certain databases permit a range of items to be specified in the form: *item*`-->`*item* (there must be no space on either side of the `-->`). The items specified, should be appropriate to the database being listed, e.g. parts for the parts database, baseline names for the baselines database etc.

All
Specifies that all versions of component type parts are to be candidates for inclusion in the report (by default no versions are included). May only be used when reporting on the parts database.

Default
Specifies that only the default version of component type parts are to be candidates for inclusion in the report.

Edit
Specifies that reserved versions of component type parts are to be candidates for inclusion in the report. May only be used when reporting on the parts database.

Registered
Specifies that the registered version of component type parts are to be candidates for inclusion in the report.

Top
Specifies that the top version of component type parts are to be candidates for inclusion in the report.

Include Statuses
Specifies that the statuses for each cycle are to be candidates for inclusion in the report. May only be used when reporting on the cycles database.

Follow Uses
Specifies then the report will follow any uses type parts encountered to the part used when reporting. May only be used when reporting on the parts database.

Recursive
Specifies that all children of each specified item are to be candidates for inclusion in the report. May only be used when reporting on the parts, baselines or check-outs databases. When reporting on the parts database this is also required by the **All**, **Default** or **Edit** options to specify that versions are candidates.

Show last
Allows the number of baseline versions which are to be reported on to be specified. i.e. report on the last *n* versions of each baseline included in the report.

Condition
Allows a condition to be specified which must be satisfied by each item which is a candidate for inclusion in the report. The condition may be any valid ACCELexpression. The item being examined is the current record of its database, so plain field references should be used for its fields. In ACCthis must be the last argument on the command line; the condition extends to the end of the line and should *not* be quoted.

Format
Specifies the name of the file containing the desired specification (both content and layout) of the report. This option is compulsory. For details on the report format file see the ***AllChange*** Administrator Manual.

Index
The index by which the data items are accessed to produce the report affects the sort order of the report and (in combination with **Value**) the items reported on. It may also have a significant performance impact when used in combination with a **Value** to limit the items reported on compared to a similar result being achieved using the **Cond**ition and the default index.

The **(Default)** index is the primary index for the database. The indexes available will include any fields defined for the database.

Value
The **Value** may be used in conjunction with any non-default index and specifies the index value the items must have in order to be included in the report.

The values that may be selected for a specified index will include the special values:

- **(Any)** to indicate all values for the index are required
- **(Empty)** to indicate values where the field indexed has no value

Output File
Specifies the name of the file to which the output of the report should be sent. It is only valid for text bases report formats. If this option is not specified, the output will be sent to the screen. If specified as `socket`<*socketnum*> then output will be sent to internal socket *socketnum*.

`-append`
This option is only available from the command line and it not presented as an option in ACE. Used in combination with the **Output File** option, it causes the output file to be appended to instead of being overwritten. This is only valid for text based reports.

Page Length
Specifies the page length of the report. The length should be specified as the number of lines required per page. If no page length is specified, it is taken to be unlimited. This is only valid for text based reports.

Printer
Send output of report to printer.

-noallowabort
This option is only available from the command line and is not presented as an option in ACE. This prevents the abort dialog from being displayed thus ensuring that the report may not be aborted; this may be useful in ACCEL scripts, such as when running a report on behalf of another program. This option is only valid for text based reports

Related Commands

Report Wizard, **List**, **Find**, **Eval**.

Concepts

Reports (Reports and Queries, ACREPORT, ACCEL).

Description

Report produces user defined reports on the various databases maintained by ***AllChange***; the content and layout of these reports can be specified by the user in external files. To obtain a quick summary of information on a database **List**, or **Find Part** for parts, can be used instead.

Before using **Report**, a *formatfile* must be set up to describe the content and layout of the report. It also defines the database(s) that are to be reported on.

The **Report** command requires the format file name and the items to be reported on as options. There are various other options which may be specified on invocation of the **Report** command. These options may all be defined in the report format file and have default values if not defined. If any of these options are specified from the report dialog or on the command line this overrides any specification in the format file.

The items that are to be *candidates* for inclusion in the report are specified on the command line. These may be modified by the **All**, **Edit**, **Default**, **Registered**, **Top**, **Recursive** and **Show last** options. Furthermore, each candidate for inclusion in the report must comply with any condition specified in the **Condition** option before it will be included in the report. The items must be appropriate to the database being reported on.

The search for candidates can be limited to a range of items when reporting on the check-out, monitor , cr or baseline databases: this is achieved by specifying *item*-->*item* as an item, e.g.:

```
report crful CR01000-->CR01500
```

outputs information on CRs CR01000 to CR01500 (inclusive) only. If the lower bound is omitted the search starts from the beginning of the database; if the upper bound is omitted the search continues to the end of the database, e.g.:

```
report crful -->CR00020
report crful CR10000-->
```

Where a database holds a large number of items this facility will (drastically) reduce the search time compared to searching the whole database.

A summary of the options whose availability depends on each database, together with the type of item required, is shown below:

Report options for databases

Database	Options	Items
part	**All Edit Default Registered Top Recursive Follow Uses**	parts
check-outs	**Recursive**	parts
baseline	**Show last Recursive**	baseline names

monitor		monitored items
cr		CR numbers
status logs		parts/ baseline names/ CR nums
workspace		workspace names
pools		pool names
class		class names
role		role names
cycle	**Include Statuses**	cycle names
command		command definitions titles
none		none

Some useful format files supplied with ***AllChange*** which will be in the ***AllChange*** system directory include:

`blineful`	Provides a full report on baselines including all information from the baseline header and details of each part baselined.
`crful`	Provides a full report on CRs detailing all information held about a CR.
`coful`	Provides a full report on check-outs giving all details held for each part checked out.
`partful`	Provides a full report on parts detailing all information held on each part and additional details about versions from the version history files.
`partiss`	Lists only versions of parts and details of who they are checked out to.
`gistats`	Shows general information about the ***AllChange*** system.
`gibt`	"Prettyprints" complex BT (Build Thread) files.
`crg*`	Draws graphs from CR statistics.
`crxl*`	Exports CR statistics to Microsoft Excel.

Other format files supplied are used by the **List** command but may also be used as useful summary formats with **Report**.

The output from a report is `ASCII` text and may therefore be viewed or edited in a standard editor, sent to a printer etc.

Reporting on Parts

A report on the parts database will include those parts specified from the command dialog (or in the format file). Normally the body of the report will be reproduced once for each named part. Specifying **All** will add an entry for each version of the named parts. Similarly, the **Default**, **Registered**, **Top** and **Edit** options include the default, registered, top and edit versions respectively. Specifying **Recursive** will cause all the children of each named part to be described.

Reporting on Check-outs

If any partnames are specified from the command dialog then all check-outs of those parts will be included in the report. If no partnames are given then all check-outs will be included in the report. A range of partnames is accepted.

If the **Recursive** option is specified then check-outs of all children of the specified parts will be included in the report.

Reporting on Baselines

If particular baseline names are specified then the report will be restricted to those baselines. If no baseline is specified then all baselines will be include in the report. A range of baseline names is accepted.

If **Show last** specifies the number of baseline versions to be included then only the last *how-many* will be included (where *how-many* is the number specified).

Show last may only be used for the baseline headers database on the default index.

If **Recursive** is used then if the baseline is a meta baseline, then all the child baselines will be visited in turn

Reporting on Monitors

Monitors to report on are specified in terms of the item monitored. If no items are specified then all monitors will be included in the report. A range of items is accepted.

Reporting on Change Requests

The numbers of those change requests of interest may be specified from the command dialog (full CR numbers must be specified, e.g. `CR00017` may not be abbreviated to `CR17`). If no change request numbers are given then all change requests will be included. A range of CR numbers is accepted.

Reporting on Pools

Items specified from the command dialog will be taken to be the names of pools to be reported on. If no pool names are specified then all pools will be included in the report.

Reporting on Classes

This will report on all classes whose names are specified from the command dialog. If no class names are given then all classes will be included in the report.

Reporting on Workspaces

This will report on all workspaces defined to the ***AllChange*** system matching those workspace names given from the command dialog. If no workspace names are specified then all workspaces will be included.

Reporting on Roles

This will report on the specified roles. If no roles are specified then all roles defined to the ***AllChange*** system will be included.

Reporting on Cycles

Without the **Include Statuses** option, information on each cycle named on the command line is provided, the body of the report produced once for each cycle. If no cycle names are given then all cycles defined to the ***AllChange*** system will be included in the report.

If the **Include Statuses** option is specified then the body of the report is produced once for each status of each specified cycle and information on specific statuses will be made available.

Reporting on Command Definitions

Those command definitions whose command names are specified on the command line will be included in the report. If no commands are named then all command definitions will be included.

Reporting on No Database

It is possible to write report format files which output general information without being centred on any particular database. Normally a report format file names the database for which it is intended and **report** iterates through this database looking for items. Specifying `none` for the database name means that no databases are searched for anything.

This facility may be used to create general reports and statistics about the system.

Command Line Syntax

report [`-recursive`] [`-all` | `-def` | `-reg` | `-top` | `-edit`] [`-latest` *how-many*] [`-uses`] [`-recurseup`] [`-noallowabort`] [`-indexedby` *index* [`-indexitem` *index-item*]] [[`-append`] `-output`*outputfile* | `printer`] [`-page`*length*] [`-format`] *formatfiledb-items* [`-cond` *condition*]

Report BTs

Function

Accessed from **File | Report BTs**. Allows a report to be produced on a Build Thread for a file showing how that file was *built* (or constructed).

Class

File.

Options

BTs
Specifies the files (or the BT files) for which the BT (Build Thread) is to be reported on.

To File
If specified defines the file to which the report should be sent

Page Length
Specifies the number of lines to be allowed per page of the report. If unspecified then no pagination will occur.

To Printer
If specified the report will be sent direct to a printer

Related Commands

Build,

Concepts

Files (File Operations)

Description

Report BTs produces a formatted report of the Build Thread (BT) corresponding to a file.

If there is no BT for the file specified an error will be produced as the output of the report.

Only files which are generated by ***AllChange*** and the ***AllChange*** **Build** tool will have BTs.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Report_BTs**)

Report Wizard

Function

Accessed from *database-menu*|**Report Wizard** or from **Reports | Report | Report Wizards** or via the toolbar. Provides a step by step guide to generating user defined reports.

Class

Info.

Related Commands

Report,

Concepts

Reports (Reports and Queries)

Description

Report Wizard is available on each of the **Part** , **Baseline** and **CR**menus. This will guide you through the process of generating a report from within ACEon each of the corresponding databases.

You will need to specify:

- What items are to be reported on
- Any conditions to be satisfied by these items in order to be included in the report
- The Index that is to be used for the report
- The category (or type) of report that you wish to generate
- The specific report format of the selected category that you wish to generate
- Where the report should be displayed

This information will be prompted for in a series of dialogs.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Part_Report_Wizard**)

call(**Baseline_Report_Wizard**)

call(**CR_Report_Wizard**)

Return Part

Function

Accessed from **Part | Return...** or via the toolbar. Allows a part which has been previously checked out to be returned, from the workspace back to the system.

Class

Part

Options

Parts

Specifies the parts that are to be returned.

Checked out Read-only
Defines what action to take on any parts which are checked out read-only:

- **Keep** specifies that they should be left checked out.
- **Return** specifies that they should be checked in.

Checked out for Edit
Defines what action to take on any parts which are checked out for edit:

- **Keep** specifies that they should be left checked out.
- **Return** specifies that they should be checked in from edit creating a new version of that part and storing the contents of the workfile in that version.
- **Discard** specifies that they should be discarded (i.e. a new version should not be created and the version that was reserved by the check out of the part is thrown away). This is used to change your mind about the previous check out for edit if you do not wish to make a change after all.

Ignore if not Checked out
Specifies that any parts which are not checked out to the current workspace should be ignored. If this is not set then an error will be generated for any parts which are not checked out.

Comment
Specifies a comment to be passed on to the version control tools as the comment against the new version created. This option may only be specified with the **From Edit** option.

Baseline
Specifies the name of a baseline into which the part returned is to be added/ replaced.

`-keep`
This option is only available from the command line and is not presented as an option in ACE. Returns the part, but keeps the workfile in the workspace. Care should be taken if this option is used directly, as the fact that a local workfile exists will not be logged in the check-outs database.

`-ignorereadonly`
This option is only available from the command line and is not presented as an option in ACE. Specifies that any parts which are checked out read only to the current workspace should be ignored.

`-ignoreedit`
This option is only available from the command line and is not presented as an option in ACE. Specifies that any parts which are checked out for edit to the current workspace should be ignored.

Related Commands

Attach Workspace, **Check Out**, **Check In**, **Issue Part**, **Alter Baseline**, **Delete Version**, **Alter Version**.

Concepts

Part (Checking Files In and Out of ***AllChange***),

Description

Return Part allows a part that has previously been checked out to be checked in. The user must currently be attached to a workspace.

Specifying a subsystem among the parts to be returned causes the **Return Part** to descend the parts tree recursively downward from that point returning whatever version is checked out of all components encountered. This makes checking in all components in a hierarchy very easy.

It is also permissible to specify `!!`*baseline-name* among the parts to return. This returns all versions in the baseline *baseline-name* (if the baseline is a meta-baseline it is followed down recursively). This is preferable to using *baseline-name*`!!*;*` which can run out of space expanding the wildcards.

If returning a part which was checked out for read only purposes then the log of the check-out is simply removed from the check-outs database and the working copy of the file is deleted from the workspace.

However, if returning from edit:

- If the part was checked out for edit with pessimistic locking then the new version, reserved during the corresponding check-out for edit, is made into a full version of the part, storing the contents of the new version under version control
- If the part was checked out for edit with optimistic locking the new version to be created is calculated at this time. If any new versions have been checked in since the workfile was checked out then an error will be issued informing that a merge is required before the part may be checked in.

If the **Baseline** option is specified, then the part returned will be added into the specified baseline if it is not already present, and will replace an existing entry if one is already present in the baseline. This facility may be useful for maintaining incremental baselines.

Parts must have been previously checked out to the current workspace. It may or may not also be a requirement that the user performing the return be the same user as the one who executed the earlier check-out: the ***AllChange*** Administrator may have configured the system to allow any user (attached to the workspace) to return a component checked out to any other user, or to permit only the person who checked out to return, or something between the two. Furthermore they may be a difference between the requirements for a plain return and a return from edit.

Command Line Syntax

return [`-readonly`] [`-edit` [`-comment` *comment*]] [`-unedit`] [`-keep`] [`-ignore-readonly`] [`-ignoreedit`] [`-issued`] [`-baseline` *baseline-name*] *parts*

Examples

```
return buildfile
```

returns whatever version of `buildfile` is presently checked out (for read-only) to the current workspace, deleting the corresponding workfile and removing the log of the check-out from the check-outs database.

```
return -edit -readonly -issued /product
```

returns whatever version of every component within the `/product` hierarchy is presently checked out to the current workspace, deleting the corresponding workfiles and removing the logs of the check-outs from the check-outs database. Each component which is checked out for read-only will be returned as such, any component checked out for edit will have the new version stored under version control and any parts which are not checked out will be ignored.

```
return -edit -comment 'Experimental version' /product/src/*.bas
```

returns all components ending in `.bas` which are children of `/product/source` from edit. These components must currently be checked out for edit to the current workspace. (It may or may not be a requirement that they be checked out to the current user.) The new version which was reserved for each component at the check-out stage is now made permanent with its contents being taken from the corresponding workfile by the version control tools. The comment (which requires quoting because of the space under ACCbut not ACE) is passed on to the tool.

```
return -unedit file
```

returns `file`, which must be checked out for edit, deleting the workfile and the log in the check-outs database, and removing the reserved new version from the parts database.

```
return !!release_1
```

returns all versions referenced by baseline `release_1`.

Role

Function

Accessed from **Misc | Set Role ...** or via the toolbar. Allows the user to set his current role .

Class

Misc

Options

-role

Specifies the name of the role required. If it is not specified or if `None` is used the current user role is set to *no role*.

Concepts

User Roles (Access Control and User Roles)

Description

Role allows the user to set his current role to one which is valid for that user.

In general a user does not necessarily need to have a current role. A user will have permission to perform an operation if he is a valid user for the required role for the particular part on which the operation is being performed.

It is useful to set the current role if the user may have more than one role and wishes to restrict his activities to those permitted by only one of these roles. Alternatively if the user needs to have the **dbsuperuser** or **dbadmin** permission, his current role must be set to whichever is required.

Command Line Syntax

role [`-role`] *rolename*

Examples

```
role developer
```

sets the user's role to `developer` which will then be used when determining required permission for all future commands. The user must have `developer` role for at least one part in the parts database.

Seebuild

Function

Accessed from **File | Build | Seebuild** displays the dependency relationships described in a buildfile.

Class

File

Options

Target

This is specifies which target object's dependencies you wish to examine. This is optional.

Buildfile

This specifies the name of the buildfile to be examined. This option is optional and if unspecified will default to `buildfil` in the current directory.

Show which files need building

This option specifies that files are to be examined to see if they need to be rebuilt and highlight those that are *out-of-date*.

Show why files need building

This option specifies that SeeBuild should highlight the files which cause a *target-file* to need to be rebuilt.

Related Commands

Build,

Concepts

Files (File Operations)

Description

Seebuild is implemented as an external tool which is invoked from ACE. It provides a display showing the dependency relationships described in a **Build** buildfile and allows the user to interactively investigate them. It is very useful for finding answers to questions like:

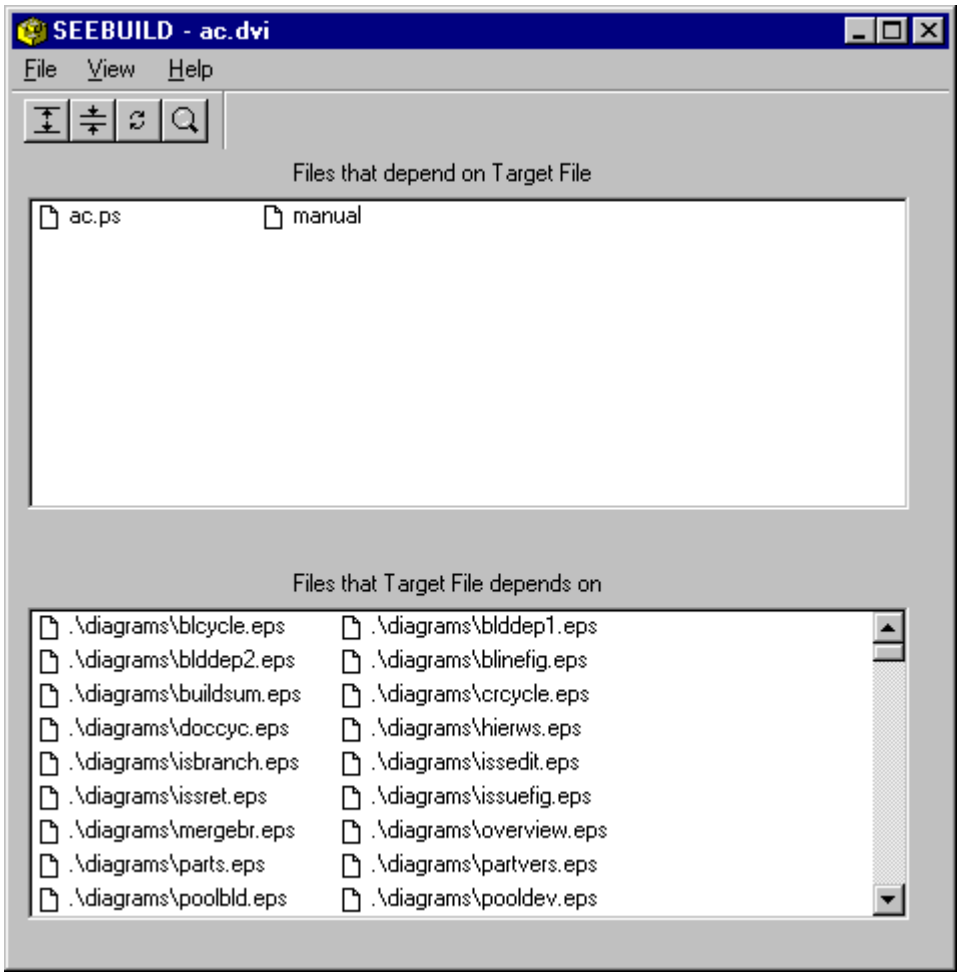
"what targets will have to be rebuilt if this file is altered"

or

"what are all the files ultimately used in building this target".

Figure 75.1 shows what a **Seebuild** display looks like.

Figure 75.1: Seebuild display

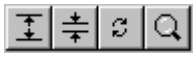


Seebuild always has a single file, the *target file*, under investigation; its name is shown in the title bar of the **Seebuild** window. If a filename has been specified as an argument to **Seebuild** this is the initial target file; otherwise it is the first target in the buildfile.

Two listboxes are displayed in the **Seebuild** window: the top list contains the names of all files which *depend on* the target file; the bottom list shows all files which the target file *depends on*. For example, if an object file were the target file then the top list might show all the executable files made from the object file and the bottom list might show all the source and header files from which the object file is compiled.

The Toolbar

The Toolbar allows the most frequently used commands to be accessed directly without having to navigate through the selection menus. Each button on the toolbar either executes a command directly or shows the relevant command dialog.



The toolbar provides the following functions (from left to right):

Out	ZoomOut level of indirection
In	ZoomIn level of indirection
Previous	Toggle between previous and current target
Next	Show "Change Target" dialog

Changing the Current Target File

To change the target file a filename from either of the listboxes maybe selected by double clicking on it. Alternatively if you know the required filename you can enter this from **File | Change Target**.

You can toggle between the previous and current target files by either selecting **File | Previous Target** or from the Toolbar.

Changing the Level of Indirection

Normally the listboxes show those files that have a direct dependency relationship with the file under examination. You may, however, need to determine which files will ultimately be affected by making a change to the target file or those files which if changed could ultimately affect the target file.

Selecting **View | ZoomOut** increases the level of indirection between the target file and the files shown in the listboxes; i.e. at level 1 the top list would show those files that depend on at least one file *that in turn* depends on the target file, while the bottom list would show all the files that the target file immediately dependents depend on. If there are no files at the next level of indirection selecting **ZoomOut** will leave the level at the current (highest) level.

Selecting **View | ZoomIn** decreases the level of indirection by one level.

If the current level of indirection is greater than zero it is indicated by a number enclosed by brackets shown at the end of the target file name.

Seebuild Options

Various options affect this information **Seebuild** displays. These settings may be altered from the **View | Options** dialog

Check allows you select to show **What** or **Why** files need rebuilding.

If **What** is selected then the modification times (or ***AllChange*** Build Threads) of the files named in the buildfile are compared. Any files that are found to be out-of-date, and need to be rebuilt, in either of the list-boxes will be marked. **N.B.** If a file does not exist it will be marked as needing to be rebuilt.

The items in the lists will be shown with an Icon to the left of the file name to denote various information about the item. The icons used are shown in the table below:

Icon	Description
	File is up to date
	File is out-of-date (needs building)

If **Why** then **Seebuild** will check whether the current **target file** needs to be rebuilt and if it does to look at the reasons why. Dependents displayed in the bottom listbox *causing* the target to be rebuilt either directly (because they are newer) or indirectly (because they in turn will be rebuilt) will be marked with an appropriate symbol. While in the *top* listbox any of the files displayed which will need to be rebuilt *because of* the current target will be marked.

The items in the lists will be shown with an Icon to the left of the file name to denote various information about the item. The icons used are shown in the table below:

Icon	Description
	File is not causing the targets build
	File needs to be rebuilt
	File is new (will cause the rebuild)

If **Show marked files only** is selected then the display will be limited to show only those files that have been marked by one of the **What** or Why options. This can be useful in pinpointing pertinent information.

Examples

The command

```
seebuild file1.o
```

might produce a display similar to the one in Figure 75.1 for the buildfile shown in Figure 75.2.

Figure 75.2: buildfile viewed by Seebuild

```
all : program1 program2 program3
program1 : program1.obj file1.obj file2.obj
        $(CC) -o $@ $^ $(LDFLAGS)
program2 : program2.obj file1.obj
        $(CC) -o $@ $^ $(LDFLAGS)
```

```
program3 : program3.obj file2.obj
        $(CC) -o $@ $^ $(LDFLAGS)
program1.obj : general.h program1.h file1.h
      file2.h
program2.obj : general.h program2.h file1.h
program3.obj : general.h program3.h
file1.obj : general.h file1.h
file2.obj : file2.h
```

Double-clicking on `program1` in the bottom listbox would make it the **target file** and show its dependents and dependencies. Alternatively you could have entered the target filename (`program1`) into the Change Target dialog (available from the File pulldown menu or the Toolbar). Using either method provides an easy way to *view* the effects of a specified buildfile.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Seebuild**)

call(**SeebuildSelectedFiles**, *files*)

Send Mail to User

Function

Accessed from **Function | Send Mail to User** or via the toolbar. Allows a mail message to be sent to a user.

Class

General.

Options

To
Specifies the users and/or groups to send the message to

CC
Specifies the users and/or groups to send the messages to as copy recipients

BCC
Specifies the users and/or groups to send the messages to as blind copy recipients

Subject
Specifies the subject of the message

Include selected item details
Allows details about any selected item to be appended to the message text.

Message
Specifies the text of the message

Related Commands

Send Mail to AC Users,

Description

Send Mail to User allows a mail message to be sent to any user.

If a plain user name is specified as **To, CC or BCC** then the user name is taken to be a user logon id and is a registered ***AllChange*** user or is a defined group; the mail name will then be taken from the user registration information. Note that users who do not have a **Mail name** specified in the user registration will not be mailed. If a **To, CC or BCC** name specified contains an `@` then the name is taken to be a complete mail address and is used without further mapping.

The user selections lists for To, CC and BCC will exclude and users who do not have a **Mail name** specified in the user registration.

If **Include selected item details** is selected then information about any selected item is appended to the message. This information is formatted by a function named **MakeItemMailMessage** in `utility.ac`, and may be modified if required, for instance to include additional fields.

It is designed to allow the user to send mail to another user concerning the currently selected item:

- If a version is currently selected it defaults to sending to the user who created the version
- If a baseline is currently selected it defaults to sending to the baseline creator
- If a CR is currently selected it defaults to sending to the CR assignee, or to the CR originator if the current user is the CR's assignee or it is unassigned.

If no mail system has been specified then this is not available from the **Function** menu.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Send_Mail_To_User**)

call(**SendMail**, *to*, *cc*, *bcc*, *subject*, *message*, *force-mail-to-self*)

Send Mail to AllChange Users

Function

Accessed from **Function | Send Mail to *AllChange* Users**. Allows a mail message to be sent to all registered ***AllChange*** users.

Class

General.

Options

Subject
Specifies the subject of the message

Message
Specifies the text of the message

Related Commands

Send Mail to User,

Description

Send Mail to *AllChange* Users allows a mail message to be sent to all registered ***AllChange*** users using the mailing system specified in the configuration options.

If no mail system has been specified then this is not available from the **Function** menu.

Note that users who do not have a **Mail name** specified in the user registration will be excluded from the mail.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Send_Mail_To_AC_Users**)

call(**SendMail**, *to*, *cc*, *bcc*, *subject*, *message*, *force-mail-to-self*)

Server Password

Function

Accessed from **Function | Server Password**. Allows users to change their AllChange password and administrators to add new users/passwords and delete users

Class

General

Options

Username
Specifies the login ID of the user

New password
Specifies the new password for the user

Confirm password
Specifies the password a second time to confirm it.

Add
Adds a new user and password. Only administrators may do this.

Delete
Removes the user from the password definitions. Only administrators may do this.

Change
Updates an existing entry to a new password - normal users may only perform this operation.

Concepts

Security and Access Issues

Description

Server Password allows a user to change their ***AllChange*** logon password. In addition ***AllChange*** administrators may add new logon users and remove existing ones.

If no password is specified for **Add** or **Alter** then the user must still logon but does not require a password.

Command Line Syntax

```
serverpassword [-add | -delete | -alter ] <username> [<password>]
```

Set No Keywords Substitution

Function

Accessed from **Misc | Administrator Tools | Set No Keywords Substitution**. Allows keyword substitution to be switched off on parts.

Options

Part
The top of the part tree to be searched for parts matching the **Pattern**.

Pattern
The pattern that parts must match for the function to act on them

Description

Set No Keywords Substitution descends the Parts Tree from **Part** explicitly setting the "no keyword substitutions" flag on parts (VC files) matching the **Pattern**.

This might be particularly useful to ensure that, say, all "*.bin" components' VC files definitely do not do any keyword substitutions --- this is usually achieved by adding the ".bin" file suffix to the **Binary Suffixes** Configuration Option, but that list is only used at the moment a part (and hence VC file) is first created: if a suffix is subsequently decided to be binary, the **Set No Keywords Substitution** function can ensure all existing "*.bin" are updated correspondingly.

This function is implemented as an ACCEL script in the function file admintools.ac. By default this is disabled and available only to administrators. See the Administrator manual for how to enable the admin tools.

Command Line Syntax

call(SetPartsNoKeywordSubstitutionsFlag)

Set System Flag

Function

Accessed from **Misc | Set System Flag...** or via the toolbar. Allows system flags to be set.

Class

Misc

Options

Actions
: If switched off this causes all further ***AllChange*** commands to be executed without issuing the appropriate command definition or life-cycle actions. The default is that actions are switched on.

Conditions
: If switched off this causes all further ***AllChange*** commands to be executed without checking the appropriate command definition or life-cycle conditions. The default is that conditions are switched on.

Echo
: If switched on this causes the following ***AllChange*** operations to send output to the Command Output window:

  - All ***AllChange*** commands executed by ACEare echoed.
  - Any operating system command lines starting with the @ (at) character are echoed prior to being issued.
  - All lines in script files are echoed prior to being executed. This may be placed at the start of script files to see what is going on.

  This facility is very useful for debugging. The default is that echo is switched off.

Echo Command
: If switched on all ***AllChange*** commands executed by ACE (only) are echoed. Switching the Echo flag on/ off also switches this flag on/ off.

Trace
: If switched on trace information showing what ***AllChange*** is doing will be output. In ACE this will appear in the Command Output window at the end of each command, or it will appear immediately in

the ***AllChange*** Debug Messages window if this has been opened by running ACDBGWIN. This facility is very useful for debugging.

Stop on Errors, Stop on Warnings
In ACE, if switched off these will cause fatal error messages or non-fatal warning messages respectively to be sent to the Output Window instead of opening an error dialog and waiting for a response. These may be used when a series of commands are executed and you do not wish the operation to be suspended when a warning/ error occurs, e.g. during batch processing. The default is that warnings/ errors go to an error dialog.

showobsolete=on|off
This allows the Show Obsolete flag to be set. The default is off. This is not presented in ACE and is provided for use in scripts if needed.

Description

Set System Flag allows certain system flags to be set and unset in order to alter the normal behaviour of the system. This is intended to be used by the System Administrator.

Switching off **Actions** or **Conditions** may be useful when it is necessary to update the ***AllChange*** database, but you do not require any external files (e.g. VC files) to be updated. *This will only normally be necessary after an error and should be used with great caution.*

Command Line Syntax

set [actions=on|off] [conditions=on|off] [echo=on|off] [echocommand=on|off] [stoperror=on|off] [stopwarn=on|off] [trace=on|off]

Examples

```
set actions=off
```

switches off the **actions** system flag, so that all future ***AllChange*** commands will not cause any operating system commands to be issued.

```
set actions=on
```

sets the **actions** flag back on, restoring the normal issuing of operating system commands.

Status Baseline

Function

Accessed from **Baseline | Status ...** or via the toolbar. Allows the status of baselines to be changed.

Class

Baseline

Options

Baseline
Specifies the names of the baselines whose status is to be changed.

New Status
Specifies the new status required.

Related Commands

Add Baseline, **Alter Baseline**, **Delete Baseline**,

Concepts

Baselines (Creating and Managing Baselines)

Description

Change Baseline Status changes the status of baselines in their life-cycle .

The life-cycle of a baseline is determined by its class .

The current status of the baseline will be changed to the new status specified if the following conditions are met:

- the new status is a legal status in the baseline's life-cycle
- the *exit condition* from the current status of the baseline is met
- the *entry condition*of the new status is met

On exit from the current status, the *exit actions* for the current status are performed, and on entry to the new status, the *entry actions* for the new status are performed.

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on.

The baseline life-cycle definition is stored in the cycles database. The statuses for the life-cycle and associated entry and exit conditions and actions may be accessed using the **Report** or **List** commands.

Command Line Syntax

statusbaseline ([`-status`] *newstatus*) *baselinenames*

Examples

```
statusbaseline Released b1
```

moves the status of baseline `b1` onto `Released`, which must be a valid progression in its life-cycle. Any actions associated with this progression are executed.

Status CR

Function

Accessed from **CR | Status ...** or via the toolbar. Allows the status of a CR to be changed.

Class

CR

Options

CR Nums
Specifies the numbers of the CRs whose status is to be changed.

New Status
Specifies the new status required.

Other
This option simply enables information to be passed on to the life-cycle . It may be used, for example, to specify a comment which may be passed on to a part **status** change which may in turn be passed on to a **check out**command invoked by the life-cycle actions.

Related Commands

New CR, **Alter CR**, **Assign CR**.

Concepts

CRs (Creating and Managing Change Requests).

Description

Change CR Status changes the status of a CR in its life-cycle.

The life-cycle of a CR is determined by its class .

The current status of the CR will be changed to the new status specified if the following conditions are met:

- the new status is a legal status in the CR's life-cycle
- the *exit condition* from the current status of the CR is met
- the *entry condition*of the new status is met

On exit from the current status, the *exit actions* for the current status are performed, and on entry to the new status, the *entry actions* for the new status are performed.

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on.

The CR life-cycle definition is stored in the cycles database. The statuses for the life-cycle and associated entry and exit conditions and actions may be accessed using the **report** or **list** commands.

Command Line Syntax

statuscr [`-other` *string*] ([`-status`] *newstatus*) *cr-numbers*

Examples

```
statuscr action CR00005 CR00055 CR00555
```

moves the status of CR numbers CR00005, CR00055 and CR00555 onto `action`, which must be a valid progression in their life-cycle. Any actions associated with this progression are executed.

Status Part

Function

Accessed from **Part | Status ...** or via the toolbar. Changes the current status of a part .

Class

Part

Options

Parts
: Specifies the parts for which the status is to be changed. If a component type part is specified without a version then the default version 's status is changed, unless the **Part** option is used.

New Status
: Specifies the new status for the part.

Part
: Specifies that the status of a component part itself, rather than its default version, is to be changed.

Branch
: If this option is used and the new status creates a new version of the component (as defined in the life-cycle) then the branch specified will be created as the new version. (i.e. this option forces the new version to be a branch).

Other
: This option simply enables information to be passed on to the life-cycle. It may be used, for example,

to specify the CRs against which a change is being made when the status change performs a **check out for edit**. The label should indicate the required input (i.e. in the above case it should appear as **CRs** rather than **Other**.

-optimistic
This option is only available from the command line and is not presented as an option in ACE. This option if specified for a new status which creates a new version should be passed on to any check-out for edit command performed during the status actions and allows optimistic locking to be implemented in part life cycles.

Related Commands

Alter Part.

Concepts

Parts (Creating and Managing Parts).

Description

Change Part Status alters the status of each part/ version specified to the new status providing the following conditions are met:

- the new status is a legal status for the life-cycle of the part
- the *exit condition* from the current status of the part is met
- the *entry condition*of the newstatus is met

Uses type parts do not have a status and therefore the status command is not appropriate for these types of parts.

On exit from the current status, the *exit actions* for the current status are performed, and on entry to the new status, the *entry actions* for the new status are performed.

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on.

Details of each life-cycle defined to the system are stored in the cycles database. The statuses for each life-cycle and associated entry and exit conditions and actions may be accessed using the **report** or **list** commands.

The life-cycle for a part is determined by its class , as defined in the classes database.

Any type of part may have a life-cycle and therefore a current status. For component type parts, each version of the part has its own current status, allowing each version to move through its life-cycle separately for the purposes of parallel development. Specifying a component type part without an explicit version-id causes the default version's status to be changed, unless the **Part** option is used to change the status of the component itself. The life-cycle of versions is determined by the life-cycle of the component part.

Command Line Syntax

status [`-part`] [`-branch` *branchname*] [`-other` *string*] ([`-status`] *newstatus*) [-optimistic] *parts*

Examples

```
status Development /product/component
```

moves the status of the default version of `/product/component` onto `Development`, which must be a valid progression in its life-cycle. Any actions associated with this progression are executed. This has the same effect as if `/product/component;` had been specified.

```
status Editing *.h;003
```

moves the status of version 003 of all `.h` components in the current working part onto `Editing`.

```
status -part Terminated component
```

moves the status of the part `component` itself (not any of its versions) onto `Terminated`.

Update

Function

Accessed from **Part | Update** or **File | Update**. Allows a checked out part to be updated to the latest (registered) version.

Class

Part, File

Options

Parts

This should specify one or more Subsystems or Components to be updated. If a **Subsystem** is specified then the part tree from the subsystem will be descended with each component being updated.

Files

This should specify one or more files or directories to be updated. If a directory is specified then the corresponding subsystem in the parts tree is used.

Related Commands

Check Out, **Update Workspace**

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Update allows selected parts or files to be updated to the latest (registered) version.

Parts checked out for edit with optimistic locking are checked for merge requirements and the merge tool invoked if a merge is required, otherwise updated to the latest version.

If used with a directory, the corresponding subsystem for the directory is calculated and this is used.

Command Line Syntax

This facility is implemented via ACCEL:

call(**Update_Parts**)

call(**UpdateWorkspaceParts**, *parts*, false)

Examples

Update Baseline

Function

Accessed from **Baseline | Alter | Update** or via the Update button on the baseline detail viewer. Allows the details in a baseline to be updated.

Class

Baseline

Options

Baseline
Specifies the baseline to be updated

Parts
Specifies the parts to be candidates for inclusion/ update in the baseline.

Versions
Specifies which versions (if any) should be submitted for addition to/update in the baseline:

All
Causes all versions of a component type part to be submitted for addition to/ update in the baseline. This should only be used with a condition which will filter out all but 1 version..

Default
Causes only the default version of a component type part to be added to or updated in the baseline.

Registered
Causes only the registered or top version of a component type part to be added to or updated in the baseline, ignoring any checked out version.

Top
Causes only the top version of a component type part to be added to or updated in the baseline, ignoring any checked out or registered version.

Instances
Specifies which instances should be included in the baseline. Only valid for Instance baselines.

All
All instances should be included in the baseline

Top
Only the top (latest) instance is included in the baseline

Condition
Specifies a condition which any parts to be added to/ updated in the baseline must satisfy.

`-recursive`
This argument is only available from the command line and is not presented as an option in ACE. This specifies whether any parts specified should be recursed down the tree. This is used by default in the ACE **Update Baseline** dialog.

`-delete`
This argument is only available from the command line and is not presented as an option in ACE. This specifies whether items should be deleted from the baseline.

`detail=`*parts*/*blines*
This argument is only available from the command line and is not presented as an option in ACE. This allows items to be specified to be added to, updated or deleted from the baselines.

`detailclass=` *class*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change** Administrator Manual*. This allows the **class** logged against a baseline detail to be modified. This may only be used with the **detail=** argument.

`detailstatus=` *status*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous'' and can only be used in certain circumstances — see the ***All-Change** Administrator Manual*. This allows the **status** logged against a baseline detail to be modified. This may only be used with the **detail=** argument.

Related Commands

Alter Baseline, **Add Baseline**, **Delete Baseline**, **Change Baseline Status**, **Check Out Part**.

Concepts

Baselines (Creating and Managing Baselines)

Description

Update Baseline allows the details of a part or instance baseline to be modified.

This may be useful for implementing an incremental baseline which is added to as necessary. The baseline may then be *locked* when it is required to freeze the baseline.

In order to alter the details of a baseline (and implement an incremental baseline) an initial baseline must have already been created, even if it does not contain any parts. This may be achieved using the **Add Baseline** command.

For each part specified if the item already exists in the baseline then it will be replaced with the item specified and its current details. If the item specified does not exist in the baseline it is added to the baseline. If a component version is specified then the first occurrence of a version of the part in the baseline will be replaced with the new version specified. In this way the version baselined may be changed.

For release baselines the **Default**, **Registered** and **Top** version selection is used to add/ replace the default, registered or top version respectively of each component in the hierarchy.

For instance baselines the **All** or **Top** instance selection is used to add/replace all instances or just the top instance.

If the baseline is a design baseline then no versions may be selected and each part in the hierarchy will be added/ replaced in the baseline .

Each part added/ updated must satisfy any **Condition** which is specified.

Any baseline detail items which are obsolete will *not* be added to the baseline unless the Show Obsolete flag is set.

Note that version/ parts are only added or updated in the baseline when using the recursive arguments, never deleted (even if a part no longer exists).

If the **Delete** option is specified, then the parts specified will be deleted from the baseline.

The addition/ replacement of parts in a baseline may be achieved by the **Check Out Part** command (see Check Out), if this is required.

Command Line Syntax

alterbaseline [[`-delete`] `detail=`*parts*/*baselines*] [`-recursive`] [`-def`|`-reg`|`-top`] [`-allinstances`|`-topinstance`] [`detailclass=`*class*] [`detail-status=`*status*] *baseline-names* [`-cond` *condition*]

Examples

```
alterbaseline detail=file.pas b1
```

alters baseline `b1` (which must already exist) to include the details of part `file.pas` which must be a child of the current working part, replacing any existing entry for that part in the baseline.

```
alterbaseline detail=/product/src/*.pas; b1
```

alters baseline `b1` to include the details of the default version of all `.pas` components in `/product/src`. If a version of any of these parts already appears in the baseline it will be replaced by the current default version.

```
alterbaseline -delete detail=b1!/product/src/*.pas;* b1
```

alters baseline `b1` by deleting the details of any versions of all `.pas` components which were in `/product/src` when the baseline was taken.

```
alterbaseline -def detail=/ subsystem defbline
```

updates the release **defbline** to contain the current *default* version for all the components in `/subsystem`.

```
alterbaseline -recursive detail=/subsystem desbline
-cond pa_type==component
```

updates the design baseline **desbline** to contain the components in the `/subsystem` tree.

Update Workspace

Function

Accessed from **Workspace | Update Workspace**. Allows a workspace check-outs to be updated.

Class

Part

Options

All Check-outs

Updates the workspace to the current registered version of any parts checked out read only to it.

All Parts

Updates the workspace to the current registered version of all of the parts in the workspace subsystem regardless of whether they are currently checked out or not. This means that for those that are checked out read only they will be updated to the latest registered version and any which are not currently checked out will be checked out. Parts which are checked out for edit to the workspace are not changed.

In Baseline

Updates the workspace to contain only those parts specified in the baseline . For a design baseline the registered version will be used and for a release baseline the version specified in the baseline is used. Meta-baselines are supported causing recursion through each sub baseline.

Check In Parts Not In Baseline

This option is only valid with the **In Baseline** option. If selected this will force any part which is currently checkd out to the workspace but which is not in the specified baseline to be checked in (i.e. removed from the workspace). This ensures that the workspace contains only parts which occur in the baseline and no others. Any parts which are checked in will be reported in the output window.

Confirm Parts to Update

If selected then a dialog will be displayed listing all the parts that ***AllChange*** intends to update. You can cancel the update before it starts or deselect parts that you don't want to update.

Check In Obsolete Parts

If selected then any parts that are checked out in the current workspace and are obsolete will be checked in. When updating from a baseline obsolete parts in the baseline will *not* be checked back in. Only obsolete parts which do not exist in the baseline are checked in. If this option is selected, then the Check-outs For Edit option is available. The option specifies how to deal with parts checked out for edit which are obsolete. If this option is set to Ignore, then the parts are not returned. If Keep Changes is selected, then the part's changes are checked in, to create a new version if applicable. If the option is Discard then the part is returned, and any changes thrown away.

The number of obsolete parts checked in is reported in the command output window.

Related Commands

Check Out,

Concepts

Part (CheckingFiles In and Out of ***AllChange***)

Description

Update Workspace allows a workspace to be updated. This means the parts which are checked out to the workspace are updated according to the options selected.

For any parts checked out for edit with optimistic locking, if no changes have been made in the workspace and a newer version than the one checked out is available then the workspace may be updated to the newer version. If changes have been made in the workspace then the part will be offered to be merged with the newer version.

The function may also be invoked automatically on attaching to a workspace if the workspace has the **Autoupdate** flag selected, see the ***AllChange*** *Administrator Manual* for details.

This is implemented as a site modifiable ACCELfunction.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Update_Workspace**)

call(**UpdateWorkspaceParts**, *parts*, *promptlist?*)

call(**UpdateWorkspaceIssues**, *promptlist?*)

call(**UpdateWorkspaceBaseline**, *baseline*, *return-parts?*, *promptlist?*)

Use Part

Function

Accessed from **Part | Use...** or via the toolbar. **Use Part** allows a part to be specified as using another part.

Class

Part

Options

Parts
Specifies the existing part that is to be *used*.

NewPart
Specifies the new part that is to use the existing part.

Relative part
Specifies to store the part used as relative to the new usespart

Absolute part
Specifies to store the part used as an absolute path. If the part used is specified as a relative path then it will be made absolute by prepending the current working part.

Related Commands

Add Part, **Delete Part**, **Copy Part**.

Concepts

Parts (Creating and Managing Parts).

Description

Use Part allows one part to use another part, allowing the sharing of systems or subsystems to be represented.

The new part will be created to use the existing part. The new part will be of type *uses*, and is treated specially.

Various commands have a **uses** option which specifies that when a part of type uses is referenced the part that is used should actually be referenced. If the option is not specified then the uses type part itself is referenced.

Where a **uses** option is not available, the uses part will be referenced.

Command Line Syntax

use [-relative | -absolute] ([`-from`] *frompart*) ([`-to`] *newpart*)

Examples

```
use /oldproduct/part /newproduct/part
```

creates a new part, `/newproduct/part`, which uses an existing part, `/oldproduct/part`.

Assume current part is `/project1`. Then:

```
use -relative -from subsys1 -to /project1/subsys2/uses1
```

`/project1/subsys2/uses1` points to `../subsys1`; if, say, the whole of `/project1` is copied to `/project3` the uses will refer to `subsys1` in the new `project3`.

```
use -absolute -from subsys1 -to /project2/subsys2/uses1
```

`/project2/subsys2/uses1` points to `/project1/subsys1`; if, say, the whole of `/project1` is copied to `/project3` the uses in `/project2` will still refer to `subsys1` in the original `project1`.

```
use -from subsys1 -to /project1/subsys2/uses1
```

`/project1/subsys2/uses1` points to `subsys1` (i.e. `/project1/subsys2/subsys1`, not `/project1/subsys1`).

Version Control Command

Function

Accessed from **Misc | VC Command** allows the ***AllChange*** administrator to perform version control commands on the VC files associated with parts.

Class

Part

Options

Command
Specifies the version control command to be invoked.

Options
Specifies any command line options required to be passed to the selected command.

Object
Specifies either the **Parts** or the VC **Files** which are to be operated on.

Concepts

Parts (Creating and Managing Parts), AllChange Version Control Tools

Description

This invokes the version control tool specified as the **Command** with the options on the specified to operate on the VC files or the VC files corresponding to the parts specified.

Note that a part may not be specified if running client/server.

This is useful, for example, for updating a VC file to resolve discrepancies between the part and VC file.

For details of each available version control command, see the ***AllChange*** VC Tools Manual.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**VC_Command**)

call(**VCCommandSelectedFiles**, *parts*, *files*)

View File

Function

Accessed from **Part | View File or File | View.** Allows the content of a part version or file to be displayed using an external viewer application.

Class

File

Options

Part or **File**
The name of the part or file to be viewed

Concepts

Files (File Operations).

Description

View File allows the user to *view* the specified file/part using an external viewer. The viewer that is used may be specified in **Misc | Options** on the **Misc** tab. If no viewer is specified then Quick View will be used if it is installed. If it is not installed then ***AllChange*** will perform the same operation as choosing **Edit** on a file or part.

ACCEL Syntax

This facility is implemented via ACCEL:

call(view_sel_parts)

call(view_sel_files)

call(view_file, *filename*)

Vote Pass Next

Function

Accessed from V**ote | Pass to Next Voter** or via the cast vote dialog. For a serial vote causes the vote to be passed to the next voter/group of voters.

Class

Baseline, CR and Part

Options

Type
The type of the item being voted on. This may be cr, part or baseline

Item
The item being voted on. This may be a part, a CR or a baseline.

`user=`*user*
This argument is only available from the command line and is not presented as an option in ACE. This argument is classed as "dangerous" and can only be used in certain circumstances — see the ***AllChange*** *Administrator Manual*. This allows the user that passed the vote to be set to *user* instead of the user id of the user performing the operation.

Related Commands

Add Vote, Alter Vote, Delete Vote

Concepts

Voting

Description

For a serial vote causes the vote to be passed to the next voter/group of voters as defined in the vote definition. On passing to the next voter, an Initial email is sent to the next set of voters.

Command Line Syntax

votepassnext type=cr|part|baseline [user=<username>] <item>

Examples

```
votepassnext type=cr SCR00010
```

Passes the vote to the next voter and sends the vote initial email to the next voter(s).

Who

Function

Accessed from **Function | Who**. Lists the users that are currently logged onto ***AllChange*** for the current project .

Class

Info

Description

Who examines the ***AllChange*** actions log file (`acacts.log`) and displays a list in the output window of all users currently logged onto ***AllChange***.

If action logging is not enabled, then **Who** will not be able to display this information.

It is possible for **Who** to list users who are no longer logged on (e.g. a user exits ACEuncleanly). For this reason **Who** only lists users who have logged on within the last 7 days.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Who**)

Workspace Registrations

Function

Accessed from **Workspace | Workspace Registrations** . Allows you to define and modify your pool and default versionworkspaceregistrations .

Class

Part

Options

Pools
This specifies the pools that are to be registered to the workspace

Highest on Branch
This specifies the name of the branch that you wish to use as your default/ registered version of parts

Version in Baseline
This specifies the name of the baseline that contains the version of parts that you wish to be your default/ registered version

Other Patterns
This specifies any additional rules for defining the default version which do fit the **Highest on Branch** or Version in Baseline criteria.

Concepts

Parts (Checking Files in and Out of AllChange), Customising your Workspace

Description

Registrations are used to define the default version parts should have whenever they are referenced *while attached to a particular workspace*; they also define the pools that are to be used for build purposes for the workspace.

Workspace Registrations allows you to define the registrations for the current workspace.

Any pools selected in the **Pool** list will be used as pools for build and autoupdate pool purposes.

If **Highest on Branch** specifies the name of a branch, then the highest version on the branch specified will be the default version if the branch exists.

If **Version in Baseline** specified the name of a baseline then the version of the component in the specified baseline (if it exists in the baseline) will be the default version *if the specified branch does not exist (or no branch was specified)*.

Additional rules may be specified in the **Other Patterns**. The rules are applied in the order in which they are encountered.

If no rule matches for the component then the default rules for the default version will be applied.

Each line of the **Other Patterns** should specify a *Default- version-pattern*. This specifies a part — which may be a pattern — and the default version of parts matching that pattern to be used when attached to the workspace and no version is explicitly specified, see Workspace Registrations for further details.

ACCEL Syntax

This facility is implemented via ACCEL:

call(**Workspace_Registrations**)

Glossary

A

ACC
: ACC, AllChange Command Line, provides a command line interface to AllChange.

ACCEL
: ACCEL, AllChange Command Evaluation Language, is used to configure the system to your requirements. Access is controlled according to user definable rules and actions may be tailored, providing an open interface to other tools and allowing commands to trigger others.

ACCONFIG
: ACCONFIG, AllChange Configuration Editor , allows the AllChange Administrator to configure most aspects of AllChange

ACE
: ACE, AllChange Environment, provides a menu-driven interface to AllChange.

attachment
: CRs and baselines may have attachments associated with them. An attachment is an external file which is to accompany the CR/ baseline. This might be a diagram, screenshot, document or whatever.

B

baseline
: A baseline is a snapshot of the current state of a product or subsystem at a point in time. Baselines may be used for reporting/ querying, checking in/ out, releasing against, etc. Note that the term baseline may be mapped to another term (e.g. release)

branch
: Versions may lie on branches. Branches allow alternate versions of the same component to be developed in parallel. Note that the term branch may be mapped to another term

BT
: Build threads (BTs) may be maintained for each object required to construct a system. BTs contain information as to exactly what objects are composed of in terms of the versions of parts used and the translation rules used to create the objects. AllChange build uses these BTs to ensure only those objects not available are rebuilt.

C

check in
: The Check In command takes a workfile in a workspace and stores it as a new version of a part under AllChange control. The workfile is removed and the check-out record is removed. If the workfile was checked outread only, then the workfile is simply removed.

check out
: The Check Out command extracts a copy of a version of a part and places it as a workfile in a workspace, at the same time creating a log in the check-outs database

check-out state
: The check-out state of a part indicates whether it has been checked out.

class
: Classes are used to classify parts, change requests and baselines and to determine their life-cycle. They are identified by a site-supplied name. This can be used to classify the item, e.g. source code, documentation or hardware for a part, bug fix or upgrade request for a change request, test or release for a baseline. Note that this term may be mapped to another nomenclature (e.g. CIType)

command actions
: AllChange commands have user defined actions which are executed when AllChange performs its internal processing. These are defined in the command definitions database.

component
: Component-type parts are at the bottom of the part hierarchy and have no children, but they may have versions. Component type is analogous to a file in an operating system filing system. Note that this term may be mapped to another nomenclature (e.g. Item)

configuration item
: Configuration Items are items which are to be controlled. Each configuration item (CI) declared to AllChange is known as a part

CR
: A CR (Change Request) is a change management record used for recording fault reports, change requests, problem reports etc. Note that the term CR may be mapped to another term such as ACMD (AllChange Management Document), RFC (Request for Change)

cwp
: The current working part (cf. current directory in an operating system)

D

default version
: Whenever a component type part is accessed without a version specifier there is a corresponding default version. There is a standard defaulting mechanism which generally accesses the most recent version. This may be modified for each workspace by use of registrations.

E

entry condition
: The entry condition for a command is tested before internal operations may be performed; if it fails the command will not proceed. This may be used for access permissions (e.g. a user must have a particular role), or other conditions (e.g. a part must have a particular status for the command to be allowed).

I

instance
: A version may have instances associated with it to represent specific occurences of that version (e.g.in manufacturing)

L

life-cycle
: A life-cycle (or simply cycle) is defined as a series of statuses through which a part, CR or baseline passes and may be used to control access to items and implement approval procedures etc.

location
: The location field of a part (in combination with the location of its ancestors) determines the actual operating system location corresponding to the part. If omitted the location will default to the same

name as that of the part, inheriting the rest of the path from the parent part.

M

MCSCCI
: The Microsoft Common Source Code Control Interface

merging
: Merging is the process of amalgamating independent sets of changes to the same file which have been performed in parallel. AllChange provides a tool to help in this process for text files (e.g. source code).

meta-baseline
: A meta-baseline is a baseline containing references to other baselines (whereas a normal baseline contains references to parts/ versions). Note that this term may be mapped to another nomenclature

monitor
: A monitor is a registration to be notified when a particular event occurs on a particular item.

P

part
: Parts are all the configuration items known to AllChange. Parts may be of type component or sub-system (or uses a special case). Note that this term may be mapped to another nomenclature (e.g. CI (Configuration Item))

pool
: Pools are used for sharing objects amongst different users. The objects held in pools are operating system files and the pools themselves are simply operating system directories. Note that this term may be mapped to another nomenclature

Project
: An AllChange project is a set of configuration files and user data. Individual users may be assigned various roles for the products managed by the system. Permission to perform different commands can be made to depend on the role assigned to individual users.

project directory
: The directory containing the AllChange configuration files and project data

R

Register Definition File
: The file in a workspace which defines the registrations for that workspace

registrations
: Registrations are used to define the default version for a part in a particular workspace. They also define the pools that are to be used by the workspace for build purposes.

role
: Individual users may be assigned various roles for the products managed by the system. Permission to perform different commands can be made to depend on the role assigned to individual users.

S

SCCI
: The Source Code Control Interface for Microsoft

section
: A section is a part of an ACReport which determines the data that is to be shown in that part of the report

SSMS
: SQL Server Management Studio. This is a tool supplied by Microsoft with SQL Server allowing low level access to the SQL Server database. Note that it is not supplied as standard with SQL Express but can be downloaded

status
: A status is a stage in a life-cycle through which parts, change requests or baselines may pass. They may be used to control access to items and implement approval procedures etc. Note that this term may be mapped to another nomenclature (e.g. State)

status log
: This is an audit trail of the progression of an object through its life-cycle. Note that this term may be mapped to another nomenclature (e.g. Audit Trail)

subsystem
: Subsystem-type parts may have children. Subsystem type is analogous to a directory in an operating system filing system. Note that this term may be mapped to another nomenclature (e.g. Folder)

system directory
: The directory in which AllChange has been installed

U

uses
: Parts may be of type uses. A uses-type part indicates that this part uses another part (the part used is given in the location for the part)

V

VC file
: The complete version history i.e. the actual contents of every version of a component is physically stored in an external VC (Version Control) file. AllChange retrieves old versions from these files and stores new versions into them when the user checks in/ out components

version
: A part of type component (only) may have versions associated with it. These represent the revisions that the part has been through during its lifetime. The actual contents of the versions controlled by AllChange are physically stored in external VC files. Note that this term may be mapped to another nomenclature (e.g. Revision)

voting
: voting may be used to implement and record approvals at stages in the life-cyle for CRs, parts and baselines

W

workfile
: Files in a workspace are referred to as workfiles, as opposed to version history (VC) files which do not reside in workspaces

workspace
: Workspaces are used to hold local copies of parts for examination, editing and building purposes. The objects held in a workspace are operating system files and the workspaces themselves are

simply operating system directories. Note that this term may be mapped to another nomenclature (e.g. Sandpit)

Index

A

B

C

D

E

N

O

P

Q

R

S

T