

AllChange User Manual

AllChange User Manual

Version 10.0, Sep 2014



Email: support@intasoft.net

Web: <http://www.intasoft.net>

DISCLAIMER

While every effort is made to ensure accuracy, Intasoft Limited cannot be held responsible for errors or omissions, and reserve the right to revise this document without notice.

COPYRIGHT

This document is protected by copyright and may not be reproduced by any method, translated, transmitted, or stored in a retrieval system without prior written permission of Intasoft Limited.

TRADEMARKS

AllChange is a registered trademark of Intasoft Limited.

MS-DOS, SQL Server 2008, SQL Express, Excel, Word, Project, Internet Explorer, Windows and Windows NT are trademarks of Microsoft Corporation.

UNIX is a trademark licensed exclusively by X/Open Co. Ltd.

SoldWorks is a registered trademark of Dassault Systèmes

ITIL ® is a Registered Trade Mark, and a Registered Community Trade Mark of the Office of Government Commerce, and is Registered in the U.S. Patent and Trademark Office

All other trademarks are acknowledged as the property of their respective owners.

© Crown Copyright Office of Government Commerce. Reproduced with the permission of the Controller of HMSO and the Office of Government Commerce.

ACKNOWLEDGEMENTS

Copyright © 1992–2014 by Intasoft Limited.

All rights reserved.

Table Of Contents

Introduction to AllChange	2
Overview of AllChange.....	2
Facilities Provided by AllChange.....	4
Product Design.....	5
Product Development.....	6
Change Management.....	8
User Roles.....	9
Life-Cycles.....	10
Voting/Approvals.....	11
Registrations.....	11
Monitors.....	11
Build.....	12
Baselines.....	12
Release Management.....	13
Reports.....	13
Configurability.....	14
Terminology.....	15
User Interface.....	16
AllChange Documentation	17
About AllChange Documentation.....	17
Document Conventions.....	17
Related Documentation.....	17
About this Manual.....	17
AllChange Programs	18
The Intasoft Group.....	18
ace.....	18
Entering ace.....	18
Exiting ace.....	20
ACC.....	20
Entering ACC.....	20
Exiting ACC.....	21
ACC Shell.....	21
acconfig.....	21
acserver.....	22
Backupac.....	22
Acdbgwin.....	22
Project Settings Editor.....	22

VC programs.....	22
Running AllChange Programs.....	22
Getting Started.....	24
About Getting Started.....	24
Basic Setup Requirements.....	24
The Basic Setup Requirements.....	24
AllChange Projects.....	25
Home Directory.....	25
The Main Window.....	25
About the Main Window.....	25
Browsers.....	31
Viewers.....	33
About the Cycle View.....	35
Cycle Viewer Options.....	36
Getting Help.....	37
Creating and Managing Parts.....	39
About Creating and Managing Parts.....	39
Parts and the Hierarchy.....	39
Browsing the Parts Hierarchy.....	40
About Browsing the Parts Hierarchy.....	40
About Part Paths.....	42
Part Versions.....	43
Instances.....	43
Part Browser Filters.....	44
View Part Information.....	45
About View Part Information.....	45
The Part Tab.....	46
The Text Tab.....	223
The Version Tab.....	51
The Instances Tab.....	53
The Relationships Tab.....	54
The Status Log Tab.....	55
The Check-outs Tab.....	55
The Votes Tab.....	56
Browsing and Viewing Part Relationships.....	57
Part Relationships Browser.....	57
Part Relationships Viewer.....	58
Browsing and Viewing Instances.....	59

Instance Browser.....	59
Instance Viewer.....	60
Creating a New Part.....	61
About Creating a New Part.....	61
Creating New Parts Explicitly.....	61
Updating a Part.....	64
Updating a Part.....	64
Modifying Text.....	66
Importing Existing Files.....	68
About Importing Existing Files.....	68
Importing Files from ace.....	70
Importing files from Interfaces.....	70
Part Life-cycles.....	70
About Part Life-cycles.....	70
Changing Status.....	71
Archiving Parts.....	73
Archiving Versions.....	74
Advanced Use of Parts.....	75
Part Paths.....	75
Versions and Branches.....	76
Fetch Files.....	78
Uses Type Parts.....	78
Checking Files In and Out of AllChange.....	80
About Checking Files In and Out of AllChange.....	80
Valid for Change CR Criteria.....	81
Locking Models.....	81
Workspaces.....	83
About Workspaces.....	83
Checking Out.....	89
About Checking Out.....	89
Check Out Read-Only.....	91
Checking Out for Making a Change.....	92
Checking In.....	98
About Checking In.....	98
Check In Parts.....	104
Check In of Files/ Directory.....	105
Using Life-cycles to Check In and Out.....	105
Checking In and Out from Microsoft Development Environments.....	106

Checking In and Out from Eclipse	107
Checking In and Out from MS Word	108
Checking In and Out from MS Excel	108
Checking In and Out from Real Time Studio	108
Checking In and Out from Dreamweaver	109
Checking In and Out from TestDirector	109
Checking In and Out from Rhapsody	110
Interface for versions of Rhapsody earlier than 3.0	110
Checking In and Out from DOORS	110
Checking In and Out from Explorer	111
Checking In and Out from SolidWorks	112
Finding out What is Checked out	113
Finding out What has Changed	114
What File Contents have Changed	114
Which Files have been Changed	115
Finding Parts	117
Moving Parts from one Workspace to Another	118
Updating Workspaces	118
Extracting Versions Unlogged to a Directory	120
Importing New Versions	121
Managing Files on a Remote Machine	122
Managing Web Development	123
File Operations	125
About File Operations	125
File Browser	125
Editing/Viewing Files	126
Pools	127
Build	128
About Build	128
Build Options	130
Build Threads	132
Using Pools for Building	133
Promoting Files to a Pool	134
Updating Pools Automatically	136
Releasing files from a Pool	136
Comparing files	137
File Stamping	137
About File Stamping	137

Stamping Text Files.....	137
Stamping Word/Excel Documents.....	139
Windows Explorer Interface.....	144
MS Word/Excel Interface.....	145
Rhapsody Interface.....	146
Interface for versions of Rhapsody earlier than 3.0.....	146
DOORS Interface.....	147
Eclipse Interface.....	148
File Decorators.....	148
Moving/Deleting Files.....	149
Synchronisation Support.....	149
Limitations.....	149
SolidWorks Interface.....	149
AllChange Task Pane Tab.....	150
Synchronising Custom Properties.....	150
AllChange Functions.....	151
Building.....	154
About Building.....	154
How Build Works.....	155
The Buildfile.....	156
About The Buildfile.....	156
Special Targets.....	158
Build Macros.....	158
Build Macro Substitution.....	159
Build Macro Expansion.....	159
Internal Macros.....	159
Macro Priorities.....	161
Search Path for Files.....	161
Inference Rules.....	162
Action Line Modifiers.....	162
Creating External Files.....	163
Buildfile Directives.....	164
How to Set Up a Buildfile.....	164
About How to Set Up a Buildfile.....	165
The Build Template.....	166
Search Path for Files.....	169
Updating Buildfiles.....	169
Equivalent Filenames.....	170

Autobuild Examples.....	171
Autobuild Notes.....	173
Build Builtin Rules and Macros.....	173
Build Environment.....	173
Example Buildfile.....	174
About The Build Tool.....	174
About The Autobuild Tool.....	176
Preprocess Tool.....	176
About Preprocess Tool.....	176
Prep Command Line Options.....	177
Prep Directives.....	178
Prep Macros.....	179
Prep Expressions.....	180
Prep Sample input file.....	182
Prep Examples.....	182
ACCEL.....	184
About ACCEL.....	184
ACCEL Condition Editor.....	184
Creating and Managing Change Requests.....	188
About Creating and Managing Change Requests.....	188
Browsing CRs.....	188
View CR Information.....	190
View CR Information.....	190
CR Number.....	191
The General Tab.....	191
The Text Tab.....	223
The Relationships Tab.....	194
The Status Log Tab.....	196
The Votes Tab.....	197
Browsing and Viewing CR Relationships.....	198
CR Relationships Browser.....	198
CR Relationship Viewer.....	199
Creating New Change Requests.....	200
About Creating New Change Requests.....	200
Creating a CR in ACE.....	201
Creating a CR by Email.....	202
Creating a CR using the Web Browser.....	203
Creating CRs from a Word Document.....	203

Updating a Change Request	205
About Updating a Change Request	205
Modifying Text	206
CR Life-cycles	207
Assigning a Change Request	210
Locking CRs	210
Hierarchical CRs	210
Selecting CRs	211
Reading a CR into the Viewer	211
Selecting CRs Shown in the Browser	211
Selecting CRs Assigned to Me	212
Creating and Managing CRs from Doors	212
Integration with Microsoft Project	213
Customising the Arbitrary Fields	214
Archiving of CRs	214
Creating and Managing Baselines	216
About Creating and Managing Baselines	216
Part Baselines	217
Meta Baselines	217
Baseline Browser	218
View Baseline Information	220
About View Baseline Information	220
Header	221
The Text Tab	223
The Details Tab	224
The Relationships Tab	226
The Status Log Tab	228
The Votes Tab	228
Browsing and Viewing Baseline Relationships	229
Baseline Relationships Browser	229
Baseline Relationship Viewer	230
Creating a New Baseline	231
About Creating a New Baseline	231
Creating a Meta Baseline	231
Add Baseline	231
Creating an Empty Baseline	234
Copying a Baseline	234
Updating a Baseline	236

Updating the Header.....	236
Updating the Details.....	237
Automatically Altering Part Baseline Details.....	238
Baseline Life-cycles.....	241
Finding Out the Differences between Baselines.....	244
Checking Out a Baseline.....	244
About Checking Out a Baseline.....	245
Incremental Releases.....	245
Archiving Baselines.....	245
Voting.....	247
About Voting.....	247
Casting a Vote.....	249
Votes Browser.....	250
Vote Viewer.....	251
Monitors.....	253
Monitors.....	253
Reports and Queries.....	255
About Reports and Queries.....	255
Types of Report.....	256
Report Wizard.....	261
Generating AllChange Reports from MS Word.....	265
Report.....	266
About Report.....	266
Common Options.....	267
Limiting the Items Reported On.....	269
Page Breaks.....	269
Reporting on Parts.....	269
Reporting on Instances.....	270
Reporting on Baselines.....	270
Reporting on Check-outs.....	270
Reporting on Change Requests.....	270
Reporting on Votes.....	272
Reporting on Monitors.....	272
Reporting on General Information.....	272
Generating New Reports.....	273
Generating New Reports.....	273
Column Report Wizard.....	274
Access Control and User Roles.....	277

Access Control and User Roles.....	277
Customising your Workspace.....	279
About Customising your Workspace.....	279
Workspace Registrations.....	279
About Workspace Registrations.....	279
Registering to Pools.....	280
Default Version Registration.....	280
Attaching to a Workspace/ Role Automatically.....	280
Customising the User Interface.....	282
About Customising the User Interface.....	282
Defining Browser Contents.....	282
Shortcuts.....	283
Spell Checking.....	286
Views.....	287
Folding View.....	288
Options.....	289
The Environment.....	292
AllChange Web Browser Interface.....	293
About AllChange Web Browser Interface.....	293
AllChange Browser Interface Logon.....	293
Browser GUI.....	295
Browsers:.....	296
Viewers.....	296
Reports.....	299
Web Workspaces.....	300
Options.....	301
Cancel Current Operation.....	302
Glossary.....	303
Index.....	308

Introduction to *AllChange*

Configuration Management (CM) is a very familiar term to the developer, after all, every development project does configuration management and all developers do configuration management ... so what's the problem?

The real question is not about what you do but *how well do you do it?*

Every developer and manager for a project (which may or may not involve software) would probably consider CM to be something along the lines of:

"A set of procedures for tracking and documenting a product throughout its life-cycle, to ensure that all changes are recorded and that the current state of the product is known and reproducible."

and they would, of course, be correct — but that is only one aspect.

There is:

- The identification and secure storage of the items that need to be brought under CM control (i.e. the control of the configuration) together with the actual implementation and enforcement of the CM procedures.
- Building — the process of ensuring the product is *built* correctly.
- Audit — the process of verifying and validating that the configuration is complete and consistent.
- Accounting — the process of keeping records of the other CM activities.

So where does ***AllChange*** fit in to all this? How can it help?

AllChange forms the cornerstone upon which all aspects of Configuration Management are built. In other words:

- If you do not do it very well then ***AllChange*** will help change all that with as little pain as is possible.
- If you do it quite well, but it is a large overhead in manual procedures, then ***AllChange*** will reduce that overhead significantly.
- And if you do it very well already then ***AllChange*** will help you do it even better!

Software is becoming increasingly complex with multiple code streams across multiple development environments becoming more of the norm than the exception.

Consumers rightly demand even better quality and reliability with new product features in even shorter timescales. They are, after all, the driver behind the development process. But such demands require an even greater concentration on control of the development process to achieve such goals.

Overview of *AllChange*

AllChange provides a Configuration Management system second to none. ***AllChange*** provides an *active* supportive environment for your development, but does not impose throttling constraints or unacceptable bottlenecks.

AllChange provides information about the make-up of your product or project and actively helps project managers and developers by:

- Ensuring that your procedures are carried out according to your specification
- Tracking the progress of change
- Securing components of a product against unauthorised access

- Allowing your developers to work together but without getting in each others way
- Making information available to those people that require it

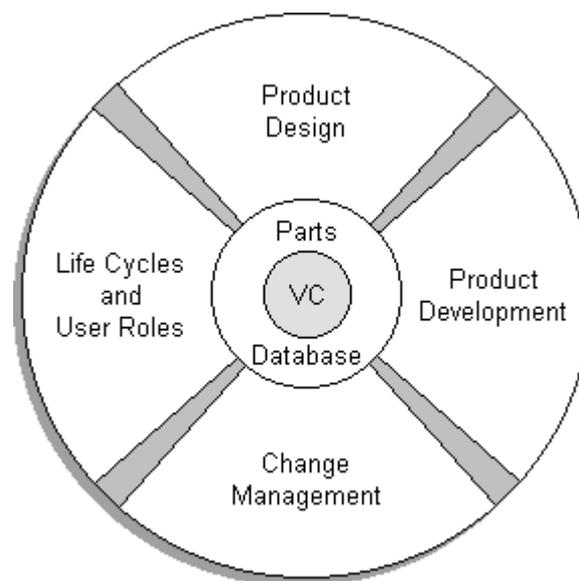
You will have noticed that throughout the above we have referred specifically to "your" procedures, "your" processes and "your" people. This is because a key feature of **AllChange** is that it is *highly configurable* so that it will fit into your current working practices, not fit you around its own.

Let's face it, a major change in your working practices is never easy, even if desirable, and adoption of any software, whatever it promises, should not mandate such a change.

AllChange ensures that you remain in complete control of your own processes.

Taking this a step further, we made a positive decision to allow you to specify the levels of control that suit your specific needs. We recognise that different projects will require different levels of control. For example, a safety critical system is likely to require rigorous control whilst a simple payroll application, in comparison, is likely to require substantially less.

How does **AllChange** do all this?

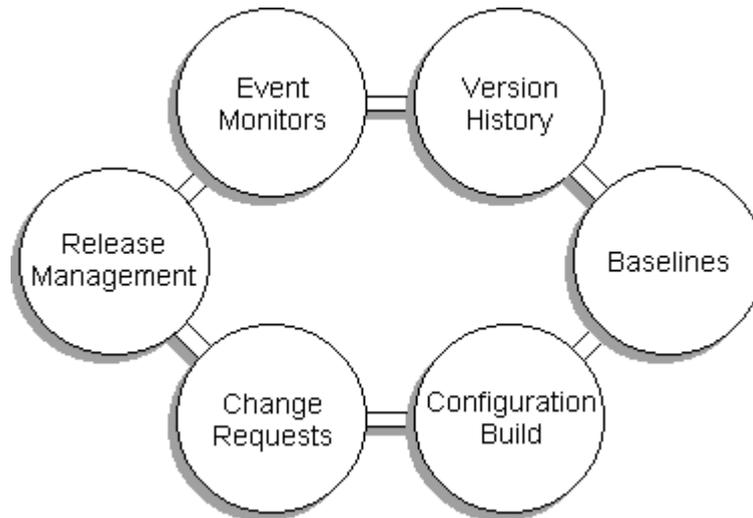


AllChange has at its heart a database holding information about all the configuration items that you want to manage.

These items — including every revision they undergo — can then be tracked through the process of change: from inception, through development and on to implementation and maintenance.

Access to **AllChange** can be controlled and items it holds can be monitored, providing active support throughout the development and maintenance processes.

AllChange contains a number of integrated managers which share information between them to produce a complete Configuration and Change Management environment. This ensures support is provided for the whole development life-cycle, through original design and development, and on to product enhancements and fault correction.



AllChange includes a range of facilities designed to support your development to the fullest including:

- Design decomposition
- Life-cycle support
- Configuration Management
- Product development
- Change Management
- Baselining
- Configuration Building
- Release Management
- Audit/ management trail
- ad hoc query/ reporting of all items in the database

Facilities Provided by AllChange

AllChange uses its database to store and maintain all information you consider to be relevant to the different aspects of product development and the changes that you experience.

- [Design decomposition](#)
AllChange enables you to describe the way you have structured your product by helping you define it in terms of the logical constituents of which it is composed and the relationships that will exist between them.
- [Life-cycle](#) support
AllChange allows you to define the life-cycle of major elements you wish to control. This includes the specific item under control, the change it experiences and the baseline it goes to construct.
- Configuration Management
AllChange supports all of the requirements for a successful Configuration Management process.
- [Product Development](#)
AllChange facilitates the controlled creation and development of actual product items (e.g. software modules) as well as associated design documentation throughout the product life-cycle.
- [Change Management](#)
AllChange enables you to track and control changes to your product items, bringing under one system all the usually disparate documents that instigated the change. This includes all change control

documents whatever your site name may be for them, for example, Change Requests, Bug Reports, Problem Reports, Maintenance Requests, Component Reports etc.

- [Baselines](#)

The **AllChange** baseline mechanism enables you to specify the precise configuration of a product at any one time. This gives vital support to your Release Management process.

- [Monitors](#)

AllChange will utilise your existing email system to automatically inform your people when certain tasks have been completed and notified to **AllChange**.

- [Configuration Building](#)

AllChange is able to construct a specific configuration automatically thereby reducing the possibility of human error in the final application.

- [Release Management](#)

A vital area in today's complex world. The penalties of releasing the wrong version of code to your end users can be catastrophic to your business. **AllChange** enables you to automate key aspects of this task including the distribution of the final product to your test or implementation environments.

- Audit/ management trail

AllChange builds an Audit Trail automatically each time you put an item through a stage in its life-cycle. This ensures you can tell what you did and when.

- [ad hoc query/ reporting](#) of all items in the database

With **AllChange** you can run ad hoc reports direct from the user interface and have these sent to a file, a printer, or direct to your monitor.

Managers depend on accurate information to make their decisions. With **AllChange** you can construct reports specific to an individual manager's needs and restrict access to them if sensitive data is involved.

- Control Web development

With **AllChange** you can control your Web development with facilities for local development areas and deployment to the web server.

- Web Browser Interface

With the **AllChange** Web browser interface you can control multi-site projects across the internet or an intranet.

- Open Interface for Integrations

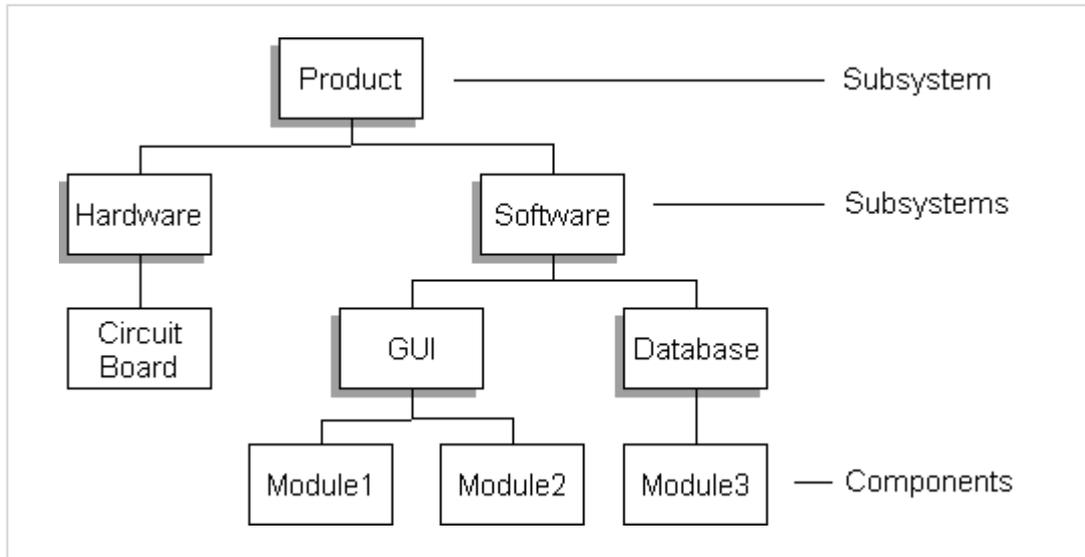
AllChange is an open system with an API which allows two-way integration with other products. You can perform your own integration or use those supplied, which include:

- Windows 95/98 and NT Explorer.
- Microsoft Office tools (e.g. Word, Excel).
- Microsoft Development tools (e.g. Visual Basic, Visual C++).
- Microsoft Project.

Product Design

Your product is likely to be constructed from many individual items. These "Configuration Items" are defined to the system in a hierarchical manner: that is, in the form of a design tree. In the same way that an Operating System structures its files by directories and filenames so **AllChange** structures its Configuration Items by *subsystems* and *components* (jointly referred to as *parts*).

This tree helps you model your product's structure and provides the framework for most of the work you want to do.



AllChange will track the *versions* of all the components, logging the date the change was made, the name of the person who made the change and the reason it was made, in addition to the actual file contents and versions thereof.

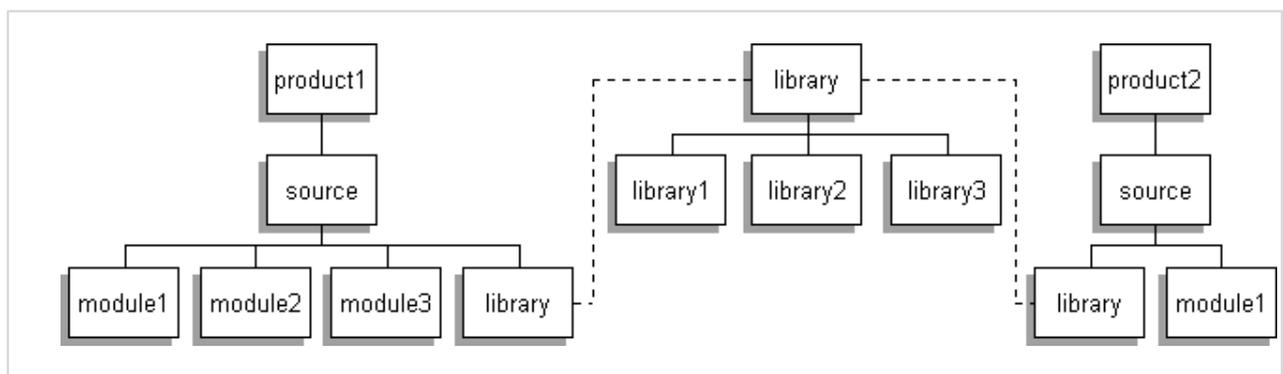
Components may refer to actual on-line files, such as source modules, documentation, design specifications; or to external objects, such as hardware. **AllChange** refers to these with the generic term *parts*.

Any changes that have been made to a file are maintained by the version control facilities, allowing any version to be retrieved, the changes to be examined and, if necessary, the regression to any other previous version of that file that you wish.

Each file (part in **AllChange** terms) has a *class* which determines its *life-cycle*. This helps you classify the different types of item that you want to control (e.g. source code, document, image etc.).

The classes are defined by you to suit your specific needs, as is the life-cycle through which it progresses.

In addition to the hierarchical relationship between parts, the *usage* relationships that may exist between them may also be defined. In the diagram below **Product1** and **Product2** both *use* the **Library**.



Product Development

AllChange has been specifically designed to help you create an environment for product development and maintenance in which productivity, quality and reliability can thrive.

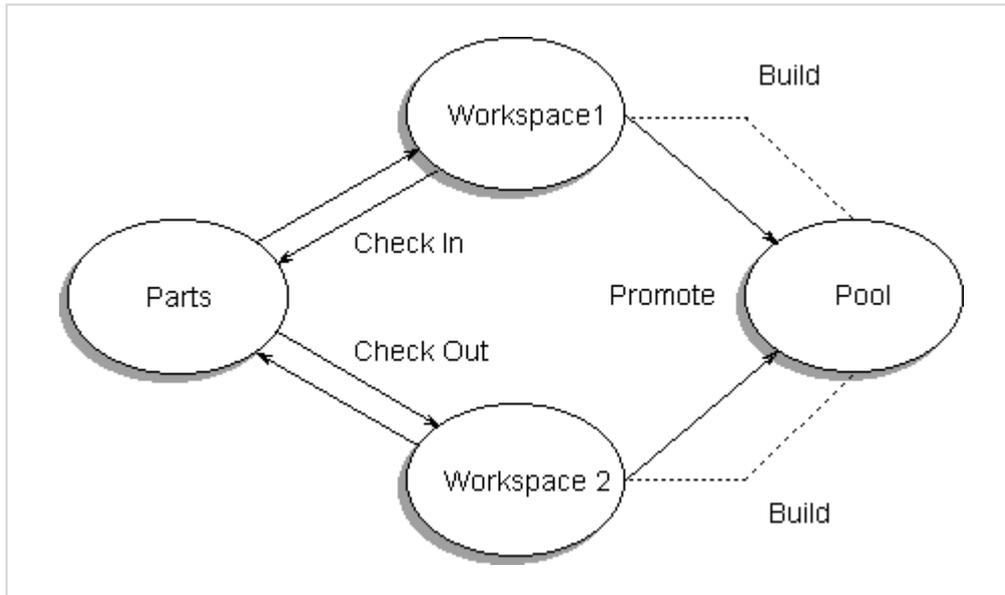
Each item you put under control is stored in a discrete section of the **AllChange** repository known as the *parts database*.

AllChange support for the development environment is founded around this and two other key concepts:

- User Workspaces
- Shared Pools

Each developer has a *workspace* associated with a particular **AllChange** subsystem; developers can also share workspaces. It is usually the directory where they perform all of their development and building work for that particular product.

Pools are created for objects which are to be shared amongst different workspaces (e.g. derived object files, header source files and other such common code). A different pool may be used for each of your configurations.



Whenever a part is needed, either for the purpose of making a change or for examination, a copy of the part is *checked out* to a workspace. **AllChange** maintains a record of which parts have been checked out to each workspace, enabling you to identify exactly who is working on what at any one time.

The developer can then make any required changes to the *checked out* item. If appropriate, he can then rebuild any required objects using a combination of the items in the local workspace and those from the pools as appropriate. This ensures that the developer uses the latest/ consistent versions of items (from the pool) which they are not changing.

When a change is complete the parts are *checked in* and the new version stored. Any objects which are to be made generally available to other users are *promoted* to the appropriate pool.

As an alternative, the changed part can be moved to another workspace and used by another developer/ manager for their specific purposes. This is typically done as part of a quality review process where the changed code has to be approved by another individual before the change can be accepted back into the system.

In further support of an approval type process, any conditions you wish to impose before a promotion is allowed can be defined to meet your specific needs.

The management of workspace and pool contents may be achieved:

- **manually**, i.e. the developers specifically request that parts are checked out to their workspaces and promote files to pools when they wish to publish their changes.
- **automatically**, i.e. **AllChange** may be configured to automatically update workspaces and pools as new versions are created.

Users may register particular versions of parts that they require as their default. This ensures that consistent version sets can be defined and accessed with the minimum of user intervention.

Many popular software tools allow **AllChange's** check out/ check in process to be executed from within their own GUI, thus allowing users to benefit from the change control facilities offered by **AllChange** but without the necessity of learning a new GUI interface.

Integrations supported include:

- Microsoft Word.
- Visual Basic and Visual C++.
- Windows 95/98 and NT Explorer. .

Information updates can be made automatically by the use of *monitors*. Your email system is used to notify a person that a particular event has occurred on an item they have expressed an interest in.

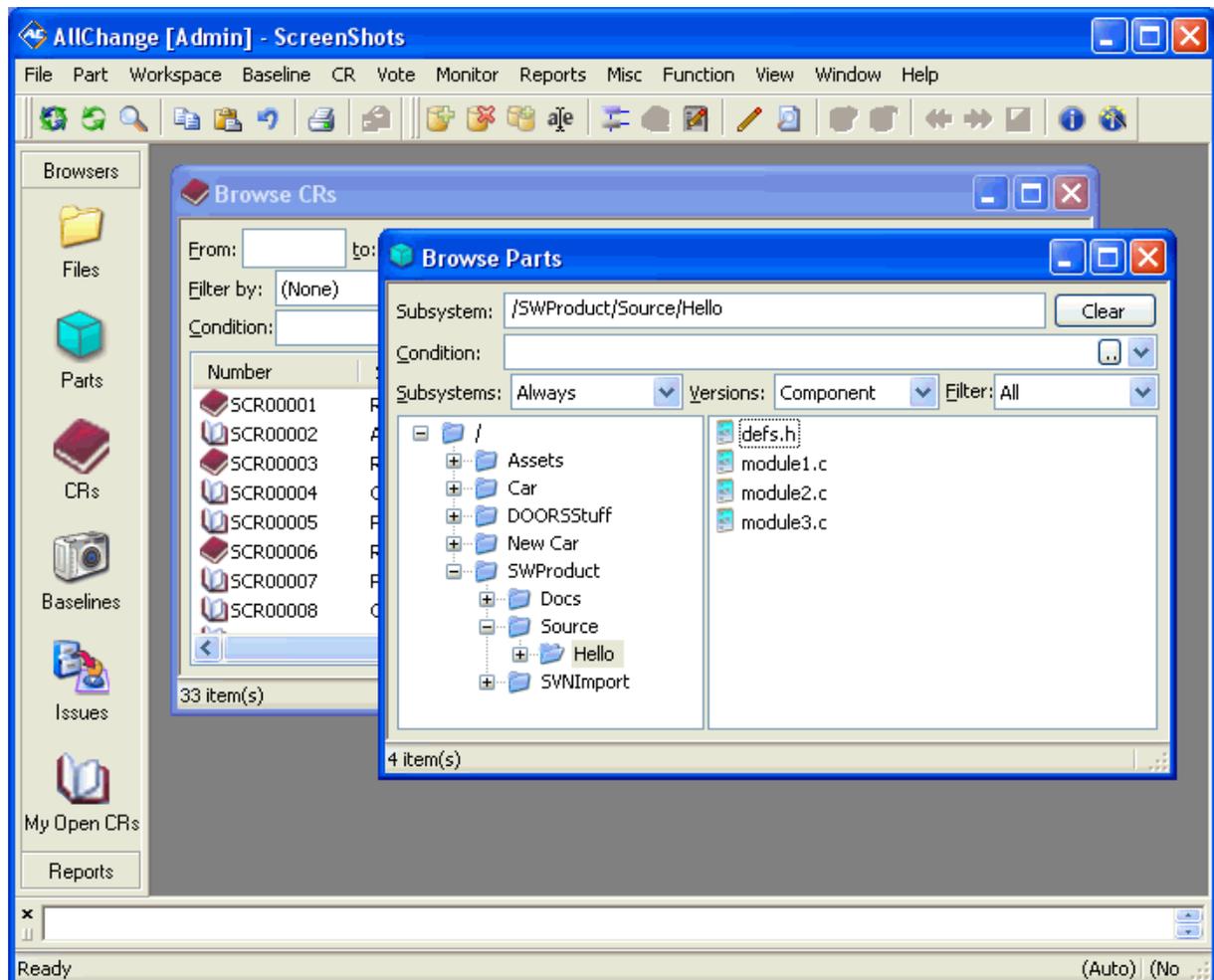
In this way, a manager can be informed the moment a particular change is completed. This is an excellent way of guaranteeing that the manager is notified of any changes made by any individual on a particular item.

It also allows developers to be notified of changes which will have an impact on their development.

AllChange has a very sophisticated *Configuration Building* process which allows the construction of an object by reusing any components that match a specification — see [Build](#) below.

Change Management

Change management allows the creation of fault reports, change requests, problem reports, maintenance requests etc. in the form of *Change Request* documents (CRs). Different "classes" of CR may be used for different types of problems and each type can be cross referenced to any other.



In the same way as other items in **AllChange**, CRs have associated life-cycles, defined by you, which may be used to *trigger actions* or *enforce approval procedures*.

Each class of CR can have its own life-cycle. For example: a bug report may have a different life-cycle and approval procedure from an upgrade request.

CRs may be "numbered" (i.e. identified) according to your own site standards and automatically assigned a unique identifier. Again, different classes of CR may have different numbering schemes, or may be entirely as you specify.

CRs may be associated with the parts contained within the parts database and may be used to control changes to these items. They may also be associated with the particular baseline which caused the CR to be raised and, naturally, the baseline in which the CR was completed.

CRs may be *assigned* to developers for action and by using the monitor function developers and managers can be kept informed about its progress. **AllChange** can also use your email system to notify the intended recipient that the CR has been assigned to them and include summary details if you wish it to do so.

As mentioned above, CRs can be cross referenced to other CRs of a different class. In this way a hierarchy of CRs can be recorded. For example: *Incidents* may be defined which result in *Problems* which in turn may result in other CRs (or Requests for Change — RFCs) to be generated which can be assigned to individuals to make the required changes.

You can build rules which allow you to ensure that the Problem Report may not be closed until all the RFCs associated with it have also been closed.

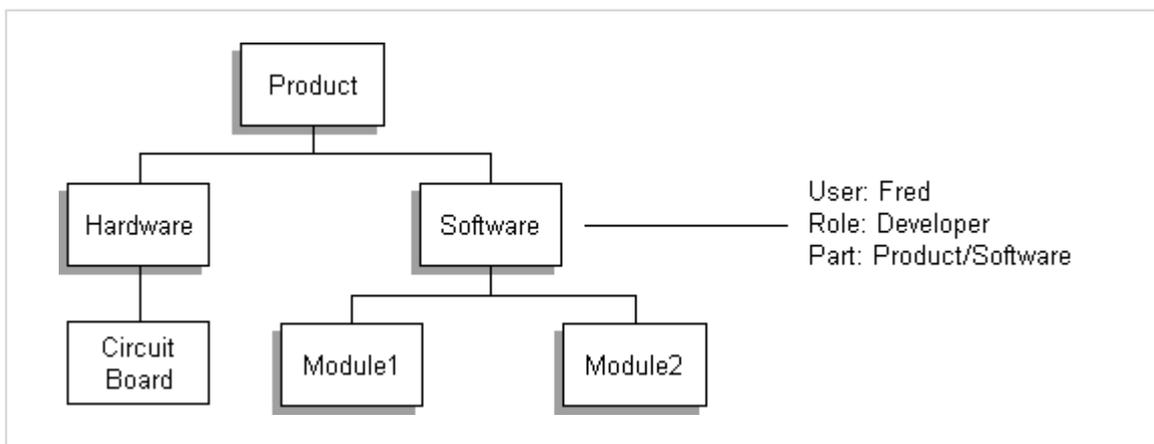
CRs may be submitted by E-mail or over the internet/ intranet using the Web browser interface.

User Roles

AllChange uses the concept of a user *role* to implement access controls and procedure enforcement. **AllChange** recognises that an individual may fulfil more than one role at any one time and user roles accommodate this.

Roles are defined and assigned to users as a part of the **AllChange** system setup and administration.

Your developers can be assigned various roles for different parts in the database. Development Team Leaders can, for example, have their own role which gives them greater access to the range of **AllChange** functions than, say, an Analyst.



For example, "Fred" may be a developer for Product Software but not be allowed to update the database design documentation. "Tom" as the database administrator may not be allowed to change a particular part or parts and so on. The rules you construct can be as simple or complex as you care to make them.

It should be noted at this point that access to **AllChange** functions can be controlled down to the specific command, role and object that is to be accessed if it is useful for you to do so.

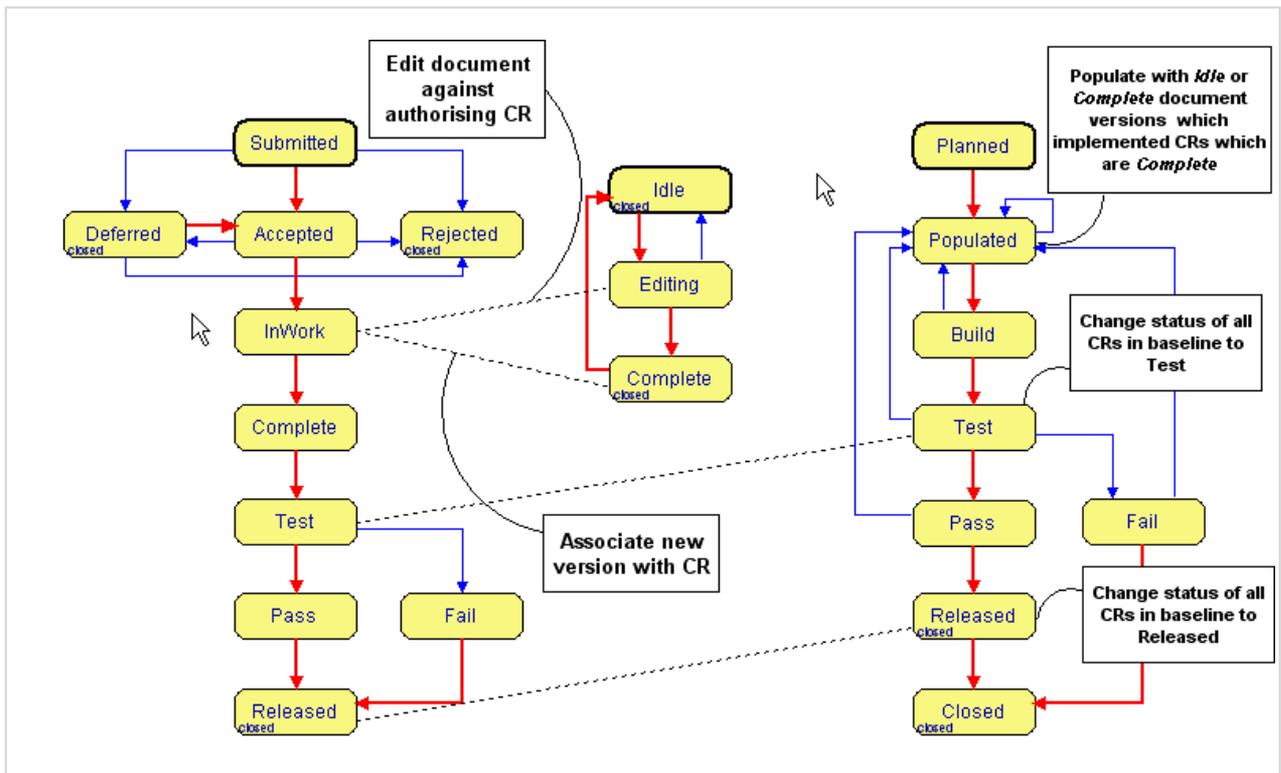
Another example could be: Developers may be allowed to **check out** and **check in** parts, but only Project Managers may **add**, **delete** and **rename** parts.

Furthermore, roles may be associated with life-cycle statuses (see [Life-cycles](#)) for procedure enforcement.

Life-Cycles

Life-cycles are used to implement the procedures you wish your developers to follow in the development of parts and for dealing with change requests (CRs). They may also be associated with baselines in order to specify release/test procedures.

You define what these life-cycles should be according to your site procedures by specifying a sequence of statuses, together with entry/exit conditions and actions. This means, for example, that life-cycles may have different user roles associated with each status which may, in turn, be used to trigger actions, e.g. mail relevant personnel and/or cause a part to be checked out to a developer's workspace.



By using life-cycles many actions can be totally automated. For example: the life-cycle for a baseline can be used to automate the release process for a product: on entering the status of *Build* the parts defined in the baseline can be checked out to a workspace area ready to build the release and when the status of *Test* is entered the status of all the CRs which have been released in the baseline can also be progressed to *Test* to indicate that they are now ready for testing by the testers.

In this way **AllChange** can enforce and actively participate in the procedures specified by the life-cycle, as well as acting as an information resource as to the current status of a particular item.

All parts may have an associated life-cycle according to their class. Different classes of part may thus have different life-cycles, so that source files need not have the same life-cycle as documentation or hardware items.

CRs have an associated life-cycle according to the class of CR, thus allowing different classes of CR to have different life-cycles (e.g. Fault Reports may have a different life-cycle from Upgrade Requests).

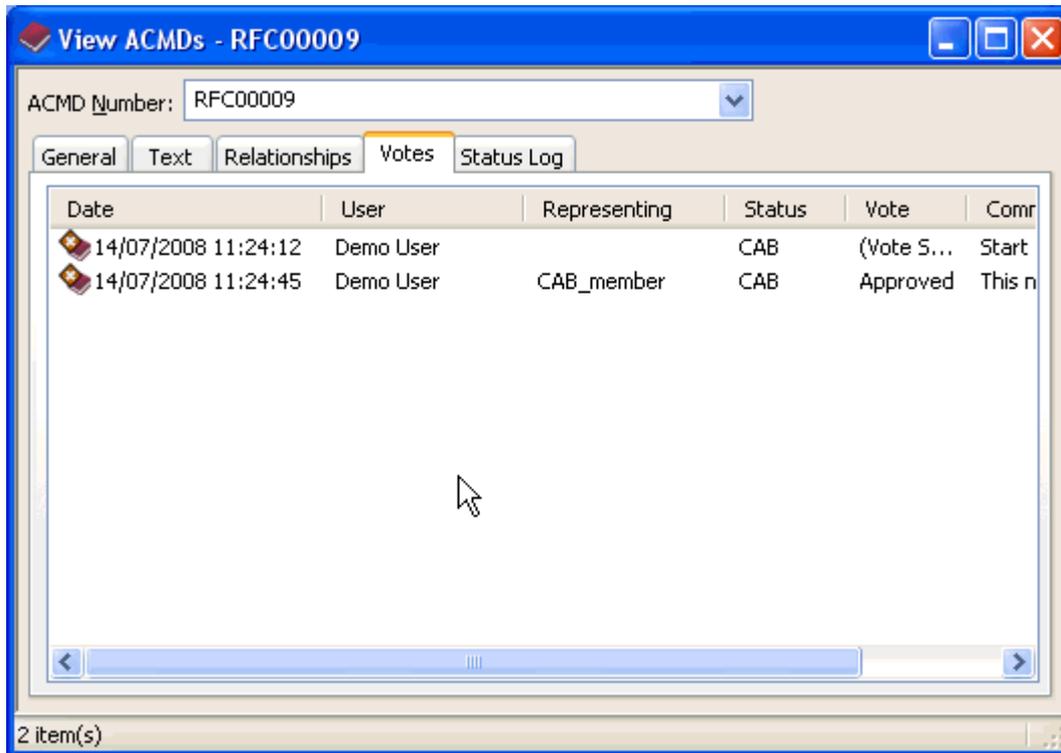
Baselines too may have an associated life-cycle to reflect the state of the baseline.

All items which have associated life-cycles may also have the *status history* of that item automatically logged. This provides for a complete audit trail on any item.

Voting/Approvals

AllChange has a facility to allow decisions to be made based on a consensus opinion at defined points in a life-cycle, this is known as voting. Voting may thus be used to implement and record approvals.

Voting is supported for CRs, parts and baselines and each of these may be enabled/disabled as required.



Different types of vote are supported including unanimous, more than half, advisory only and serial.

Registrations

Registrations provide a mechanism to ensure that the correct/ required version of parts are obtained whenever a developer uses a particular workspace. This ensures that a consistent set of parts is used and reduces the scope for human error in obtaining versions from the wrong stream of development.

For example, one developer may wish to register to the version of a part that is the top version on a particular branch if that branch exists, otherwise to the top version on the main line of development (otherwise known as the trunk). Another developer, in the meantime, may require the version contained in a particular baseline which is a totally different version to the one being worked on by the other developer.

A complete set of rules may be defined for registrations for any part, and each developer may have their own registrations.

As you can see, registrations provide a facility to isolate users from the changes made by others on the team.

Monitors

AllChange has *monitors* which can be placed on a range of actions.

The owner of the monitor is informed using the host mailing system that the event monitored has taken place. This feature is important in **AllChange's** *active participation* in the configuration management and software development process. It is not up to a person to constantly make enquiries to determine whether certain actions have taken place but rather **AllChange** will pass that information on automatically.

This means that users can, for example, monitor the creation of any new versions of a particular part, or promotion of certain objects which may affect their own development currently in progress.

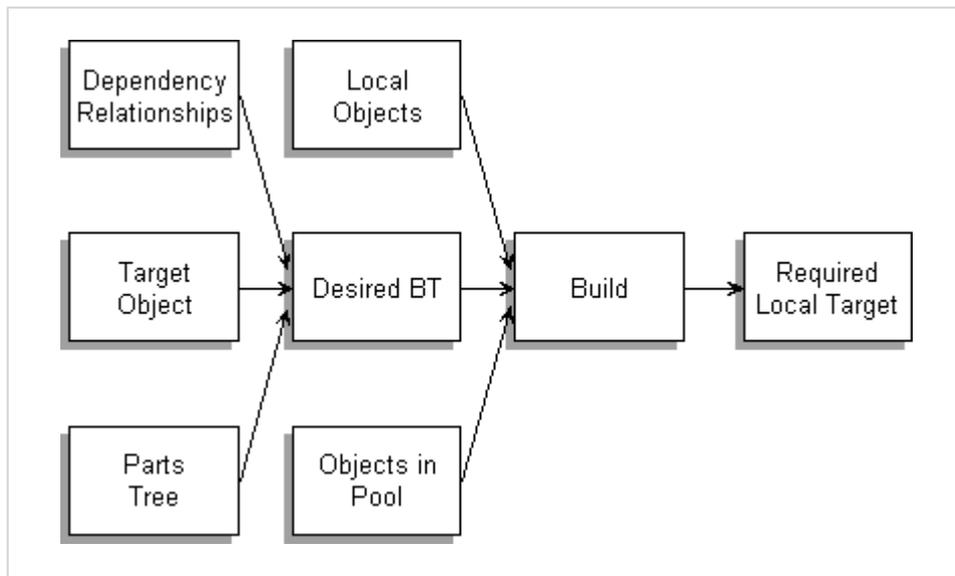
Monitors also enable relevant personnel to be kept informed about activities occurring in a project: for example, the progress of a part or CR through its life-cycle.

Build

AllChange provides sophisticated and intelligent *build* facilities.

To build the required configuration automatically, a description of the dependency relationships between objects (which may be automatically generated) plus a set of arbitrary translation rules is required.

Given this information **AllChange** will build a configuration, searching the workspace plus designated pools for an *acceptable* object, rebuilding only those objects which are not already available.



The **AllChange** build maintains *build threads* (BTs) 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 needed to create the objects.

When an object is to be rebuilt its BT is compared against a "desired BT" which defines the versions of the required parts and the translation rules that should be applied: an object is only rebuilt if its BT does not match the desired BT.

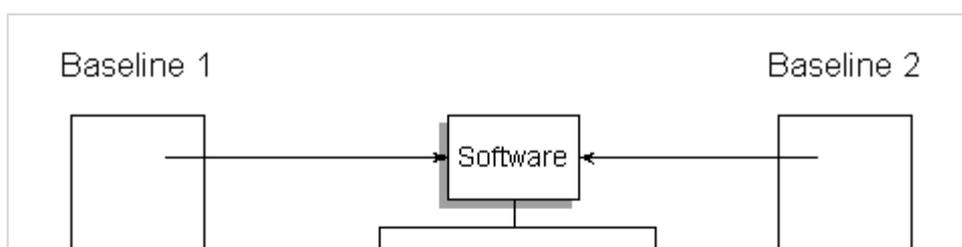
The BTs mean that, for example, the **AllChange** build can distinguish between objects built with different compiler flags (e.g. optimisation or debugging) and also distinguish between objects built from different versions of source code components.

In this way **AllChange** can manage multiple configurations simultaneously.

BTs are also valuable documents since they provide an accurate description of how an object was created.

Baselines

A *baseline* is an inventory of the current state of a product or subsystem at a particular point in time and records both the structure and details of the items included in it.



AllChange baselines provide a mechanism for preserving the definition of a system and the specific component versions that go to create it.

AllChange allows two types of baseline to be taken:

A Part Baseline

This is a baseline of parts and may be either a "design" baseline, which records the structure of the system, or a "release" baseline, which records the specific versions of the required components.

A Meta Baseline

This is a baseline of baselines, that is to say, a group of baselines. This can be useful, for example, for combining several part baselines for different sections of a product into a single baseline which logs an entire product release.

Any product/ subsystem may be the subject of a baseline. Baselines may be created using a variety of different mechanisms to suit your release processes. Listed below are some of the ways you can define precisely which versions of which objects are to be included in your baseline.

You can:

- Take a snapshot by using various selection criteria.
- Use the actual completion and/ or implementation of specific Tasks and selected Change Request's.
- Automatically create and maintain an incremental baseline by ensuring that whenever new code is checked in, the baseline is updated.
- Use a design baseline to define which objects are to be included and subsequently create a release baseline from this, by applying version selection criteria to the original design baseline.
- Use the previous release baseline and apply the changes made to implement CR's which are to be released.

Baselines may be used for a variety of purposes including:

- The versions of parts contained in a baseline may be used as the default version to be accessed.
- A build may be done against a baseline.
- Baselines may be compared to easily identify which parts changed between releases of a particular product. Baseline comparison is also a useful way of identifying changes that occurred during concurrent development and can be helpful to assessing the feasibility of merging two or more streams of work.
- Baselines provide full functional and physical auditability of a configuration.

Release Management

The **AllChange** *release* manager creates copies of releasable objects in a release directory.

Objects to be released are copied from a pool to the release directory. You can construct rules to ensure that no object is released that is not composed of a version in a particular baseline, thus ensuring that the release is repeatable.

Objects may then be released to the outside world.

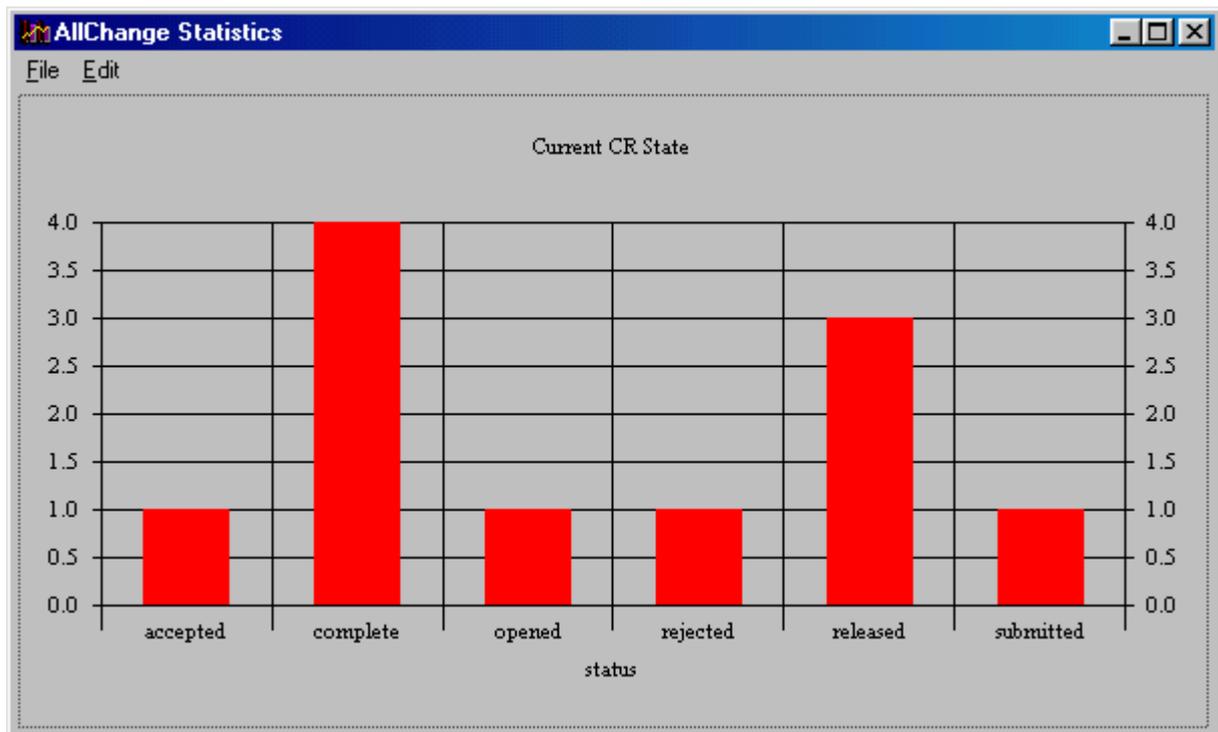
All of the release management activities that need to be performed such as promoting releases through different test environments and generation of release notes can be automated through the baseline life-cycle facilities ensuring that your releases are built from secured components and reducing manual overheads and minimising the risk of errors which may occur with a manual process.

Reports

AllChange includes a powerful yet flexible *report generator* which you can use for both formal and ad hoc reporting.

You are able to construct report formats that are specific to your site or even a particular person. You can generate any style of report that you require containing any selection of the available information.

Metrics may be generated for use in quality management and may be displayed graphically or exported to third party tools.



A range of report formats are supplied which provide all the basic information that you are likely to need for most queries. These may of course be tailored as desired.

AllChange provides you with the facility to obtain information on any aspect of the system. The report generation tool provides a drag and drop facility for creating the reports that you need.

Reports may have multiple levels allowing information from several different databases to be included in a single report. This allows you to construct reports according to the information you want to obtain rather than be constrained by the information in that particular section of the **AllChange** repository.

AllChange reporting facilities may also be closely integrated with MS Office tools allowing:

- Reports to be produced by **AllChange** and exported directly into MS Word.
- Reports to be generated within MS Word without the need to interact with the **AllChange** Graphical interface.
- Change Management information to be viewed as, for example, Gantt Charts in MS Project allowing Project Managers to obtain immediate and up to date information concerning the Activities and Tasks individual project team members are under taking including:
 - Comparing actual against planned.
 - Impact of slippage against current end dates.
 - Resource implications of re-scheduling tasks between team members.
 - Initial Gantt charts created direct from the data held in the **AllChange** repository.

Configurability

Much has been said about the ability to tailor **AllChange** to your specific requirements.

There are four keys to this Configurability:

1. **ACCEL**, the **AllChange Command Evaluation Language**, is a simple yet powerful in-built command/ programming language. It provides access to all fields of all databases, hooks to **AllChange** functions and commands, conditional actions, evaluable conditions, variables, user defined functions, access to the GUI and operating system calls (including various means of exchanging information with other programs).
2. All **AllChange** commands have a site definable *entry condition*, allowing any condition to be specified for permission to execute a command. 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).
3. All **AllChange** commands have site defined *actions* which are executed when **AllChange** performs its internal processing. This provides an open interface to other tools and allows commands to be effectively tailored, to cause **AllChange** to *actively* participate in the configuration management process (e.g. one action may trigger another or may cause mail to be issued to relevant personnel to ensure they are kept informed).
4. The **AllChange** user interface, itself, may be tailored to site/ individual requirements and preferences. It also allows sets of commands and/ or parameters to be *fetched* from a file: this helps you to standardise and reproduce commands and parameters whenever needed.

Special Administrator tools and facilities are provided for setting up the system as required at your site. The **AllChange** Configuration Editor — ACCONFIG — provides a single intuitive environment for setting up and configuring **AllChange** according to your configuration management plan.

Terminology

AllChange uses a number of terms to refer to different features and facilities that it supports. These terms are used throughout the documentation however it should be noted that site specific mappings for some of these terms may have been defined by your **AllChange** Administrator and so may appear differently in **AllChange**

Standard Intasoft Term	Description	Mapable
CR	A CR (Change Request) is a change management record used for recording fault reports, change requests, problem reports etc.	Yes
Part	A part is a configuration item under AllChange control. The Parts database may be thought of as an archive or library	Yes
Subsystem	A Subsystem is a type of part that may have children. Subsystem type is analogous to a directory in an operating system filing system.	Yes
Component	A Component is a type of part 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.	Yes
Baseline	A baseline is a snapshot of the current state of a product or subsystem at a point in time.	Yes
Meta-Baseline	A meta-baseline is a baseline containing references to other baselines	Yes
Check-out	Parts are checked out to create a file copy in a workspace. Check-outs are logged in the check-outs database.	Yes
Version	A Component may have many versions or revisions as it is changed throughout its life	Yes
Branch	Versions may lie on branches. Branches allow alternate versions of the same component to be developed in parallel.	Yes
Class	Parts, CRs and Baselines may all have different classes. Each class represents a different type of object	Yes
Status	A status is a stage in a life-cycle through which parts, change requests or baselines may pass.	Yes
Status Log	This is an audit trail of the progression of an object through its life-cycle	Yes

Pool	A shared directory where files may be placed for reference purposes	Yes
Workspace	A directory, usually personal, where components with files may be checked out to and checked in from.	Yes
Check Out	The process of extracting a copy of a version from the parts archive into a workspace	No
Check In	The process of returning a copy of a version back to the parts archive	No

For details of how to map the standard Intasoft Nomenclature see the **AllChange Administrators Manual**

User Interface

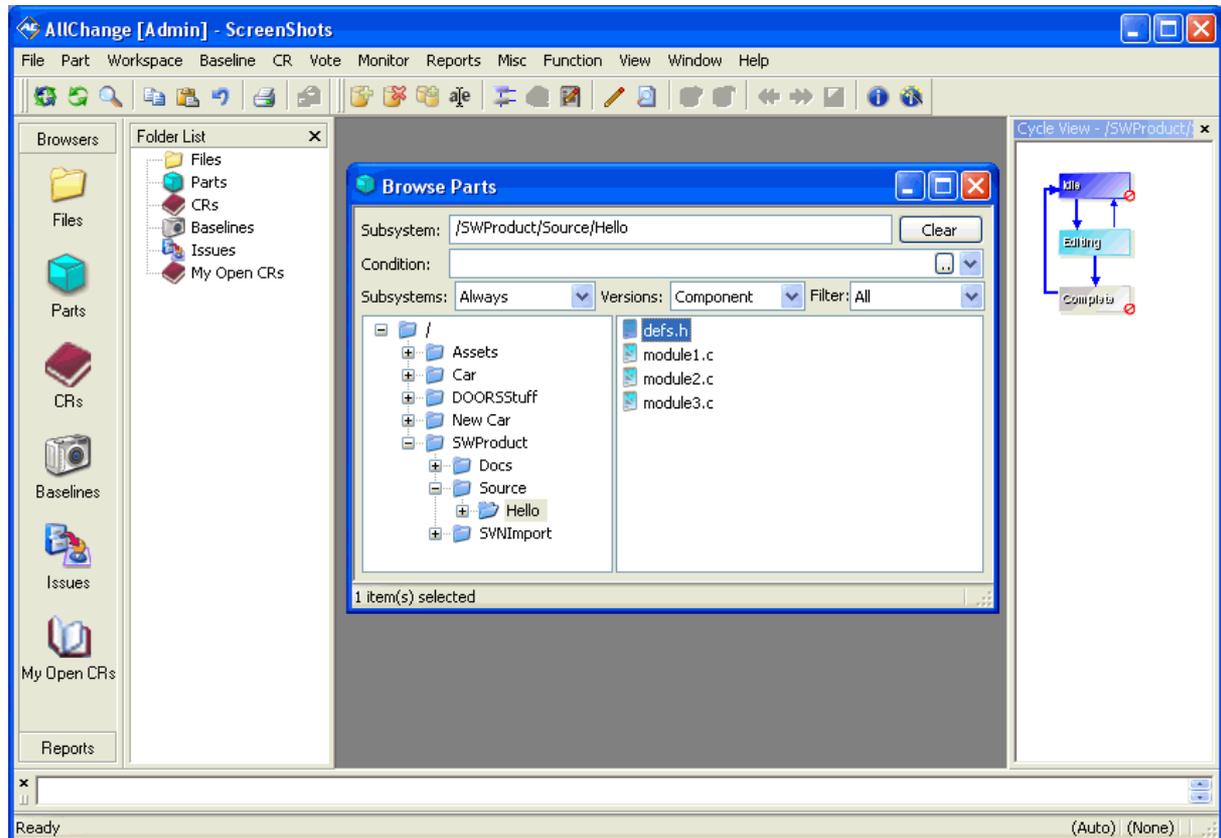
AllChange may be used via its own Graphical interface (**AllChange** Environment — ACE), directly from within many popular software tools or via an Internet/ Intranet connection using the industry standard Web Browsers. This provides you with the flexibility you need to perform many operations directly from within the tools and environments you are familiar with. Such as:

- MS Windows 95/ 98 and NT Explorer.
- Microsoft Word.
- Visual Basic and Visual C++.

ACE currently supports Windows.

You are able to browse and view each aspect of the database enabling easy access to all of your data.

You can perform operations upon items in the system by using command dialogs based on the items selected in the database windows. Existing menus and toolbars can be tailored to site requirements. The interface may be further extended by adding site-specific functions and dialogs to implement your chosen operations.



AllChange Documentation

About AllChange Documentation

AllChange is supplied with the following manuals to help you to use the facilities effectively:

- *AllChange User Manual*
- *AllChange Reference Manual*
- *AllChange Administrator Manual*
- *AllChangeVC Tools Manual*
- *Upgrade Notes* if appropriate

Document Conventions

AllChange commands will appear in bold type face (e.g. **Add Part** is the command to add a new part).

Italics are used to *emphasise* a particular point.

References to options on dialogs will appear in bold type face (e.g. the **Edit** option is used to ...).

Anything in teletype font indicates that this may be actually typed or represents an operating system or physical object (e.g. a file name would appear as `afile`).

Screenshots shown in this manual are taken under various Windows versions. Other Windows versions of the software may have a slightly different appearance.

This user guide is oriented towards using **AllChange** through a windowing system. Each **AllChange** command described has a corresponding command line which is specified in the appropriate chapter in the reference manual.

Related Documentation

The following additional documentation may be useful:

- Microsoft Windows
- Development Tools being used
- Your Site Standards
- Your email system and any other tools which are integrated with (or you wish to integrate with) **AllChange** for use at your site.

About this Manual

The *AllChangeUser Manual* provides the new user with information about **AllChange** as well as a guide for the experienced user. It includes information on how the GUI environment works and how to achieve common operations. Further details about any particular function or command may be found in the **AllChange** Reference Manual

The *AllChange User Manual* is organised into several different sections:

- Introduction — provides a general overview of the system and introduces some of the basic concepts.
- **AllChange** Programs — gives a summary of the programs supplied with the **AllChange** system and their purpose.
- Introductory guide — several sections introducing you to the concepts and most of the facilities in the **AllChange** system.

AllChange Programs

The **AllChange** system contains a number of programs. This section provides a brief summary of the purpose of the principal among them.

The Intasoft Group

Under Windows, having installed the **AllChange** software package a group will be created containing various shortcuts which allow the different **AllChange** programs to be invoked and access to the on-line help and other documentation. By default the group will be named **Intasoft AllChange**, but this may be determined by each workstation at installation time.

The **AllChange** programs are

Icon	Description
	ACE — the AllChange Environment
	ACCONFIG — the AllChange Configuration Editor
	ACC — the AllChange Command Line Environment
	Provides a window for trace output
	ACEDIT — the AllChange text editor
	ACUPGRAD — the AllChange upgrade tool
	Backs up a database in a project directory
	Project settings editor

Only the Administrator will require access to all of these. A user installation will provide shortcuts giving access to ACE.

Other programs may be supplied with the system, but may not have a corresponding icon.

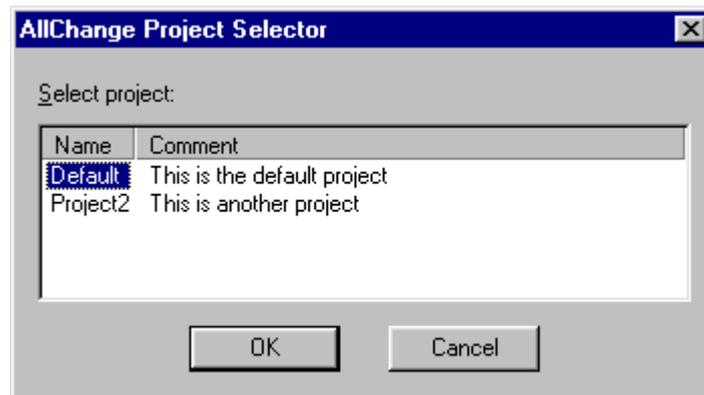
ACE

ACE — **AllChange** Environment — is the menu-driven program used to run the **AllChange** system. It will be implemented as a GUI using the native windowing system, depending on platform.

Entering ACE

ACE may be run by entering the command name (`ace`) to the operating system. Under Windows the program may be invoked by selecting the icon in the Intasoft group.

If there is more than one project available to you ACE will prompt for which project is to be used:



On start up ACE will attempt to open the database to be used for the specified project.

If no projects exist or the project directory is not specified then an error will be issued of the form:

```
No project directory specified
```

You will then need to ensure that your project definition specified the correct project directory if you have one (see below) or create a project if you do not — this would normally be performed by the **AllChange** administrator, see the **AllChange Administrator Manual** for details.

If a project is specified but no database exists an error will be issued in the form:

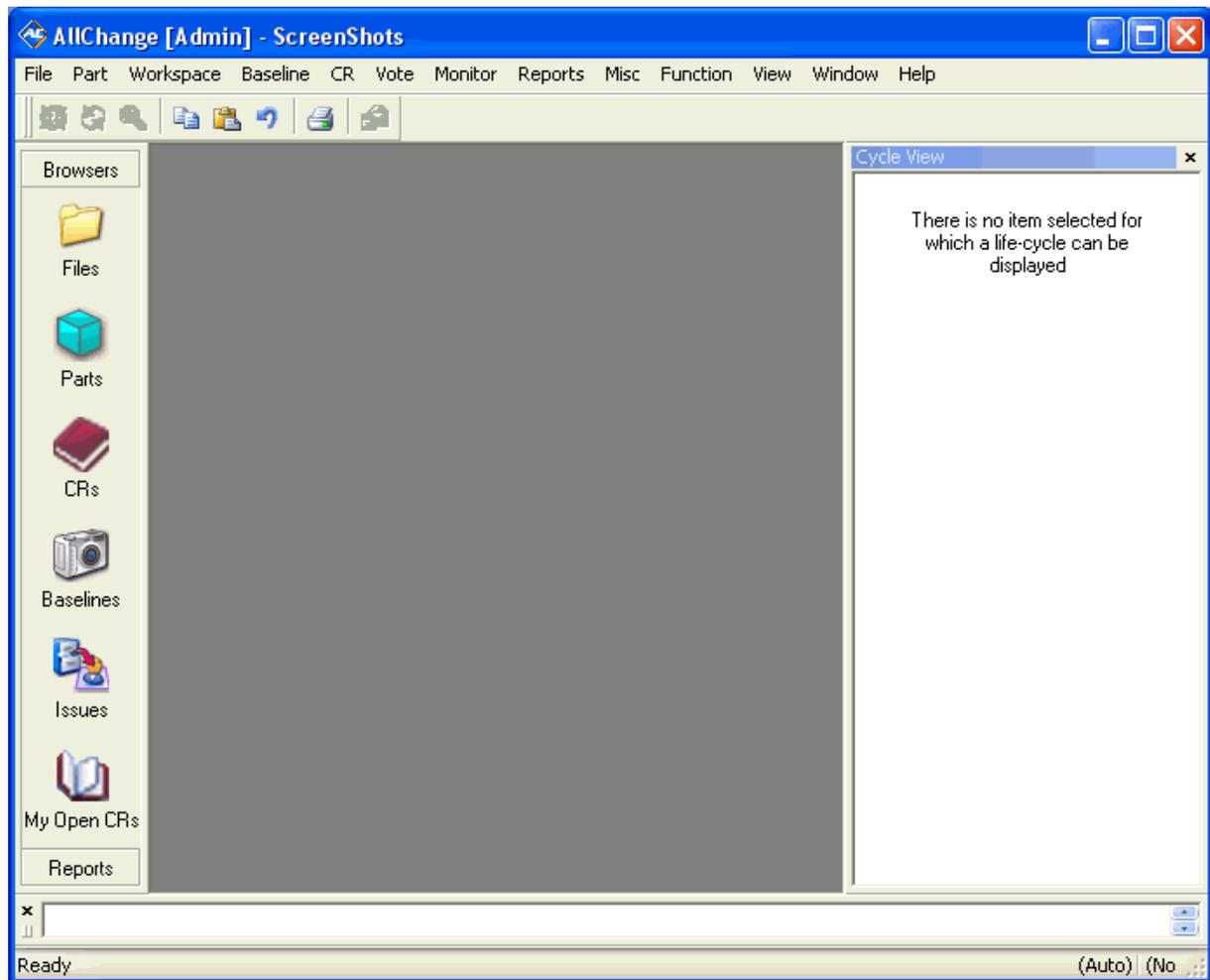
```
Database does not exist - (expected location of database)
```

There could be two reasons for this error:

1. ACE is looking in the wrong place for the database. This could be for a number of reasons including:
 - Your ACE icon (or command line) is not specifying the correct **AllChange** project
 - Your environment/ project definition does not specify the correct **AllChange** project directory
2. The database really does not exist in which case you will need to create it. Ensure that you create the database in the same location that ACE will look for it. This task would normally be performed for you by the **AllChange** Administrator, see the **AllChange Administrator Manual**.

See your **AllChange** Administrator to resolve the error.

Once successfully started the main window will be shown together with a menu bar.



Exiting ACE

In order to exit ACE select **File | Exit**.

ACC

ACC is the command line program used to run the **AllChange** system; its use is discouraged under Windows 95/98 and NT.

ACC provides a shell-like environment (e.g. C-shell under UNIX, or the MS-DOS environment). It runs on any standard text-based command session. The user sits within the program, typing in lines of text which are acted upon much like an operating system "shell". ACC may also be used to conveniently execute "scripts" of **AllChange** commands which require no user interaction.

Entering ACC

ACC may be run by entering the command name (`acc`) to the operating system. Under 32-bit Windows the program may be invoked by selecting the icon in the Intasoft **AllChange** group.

Like ACE on startup ACC will attempt to open the database to be used; if one does not exist an error will be issued (see section **Entering ACE** for more details).

A command line prompt will appear.

Alternatively, ACC may be invoked from the operating system command-line (or batch file) to execute a script of commands by use of a fetch file:

```
acc @file
```

where *file* specifies the operating system path to the script file. In this case it reads and executes **AllChange** commands from the file and finally exits, without any interaction. It also recognises an

environment variable (or `-E . . .` command-line argument) named `ACFLUSHOUTPUT` as meaning flush all output to standard output/ error immediately. These two options are used, for example, by `ACUPGRAD`, the **AllChange** Upgrade Tool, when running ACC to execute upgrade scripts; they may also be useful for user scripts.

As a special case ACC recognises the command line argument `-web`. This is used to service the web browser interface.

Exiting ACC

In order to exit ACC enter the command:

```
quit
```

ACC Shell

ACC does not have a separate manual as there is little to say. However, when used interactively it does have a line editing and history recall mechanism to allow easy viewing, recall and editing of commands entered. This interface is known as "readline" (note that this has nothing to do with FSF code of the same name).

[Figure 3.1](#) shows the keys used by the readline interface.

Figure 3.1: ACC readline keys

Key	Action
←	move left in line
→	move right in line
Backspace	delete left in line
DEL or CTRL+D	delete right in line
HOME or CTRL+A	move to start of line
END or CTRL+E	move to end of line
ESC or CTRL+U	restart line
RETURN	accept line
↑	fetch previous line from history
↓	fetch following line from history
F8 or ESC+8 or ALT+P or ESC+P or CTRL+R	search history (backwards) for next line starting with same sequence of characters as current line up to cursor position

In addition, ACC (only) provides a command, **history**, which displays the saved history of commands entered so far.

The readline interface is normally enabled whenever entering **AllChange**. The readline interface may be explicitly disabled by setting a `NOREADLINE` environment variable or entry in the `[AllChange]` section of the `.ini` file; this may be needed if problems are experienced with the terminal, another program, pipes etc. If readline is disabled **AllChange** reverts to using standard operating system line editing facilities.

ACCONFIG

ACCONFIG — **AllChange** Configuration Editor — is a tool for configuring the **AllChange** system.

It is for use by **AllChange** administrators only.

ACCONFIG provides facilities to:

- edit configuration files
- create new projects

It will be implemented as a GUI using the native windowing system, depending on platform. It is documented in the **AllChange** Administrator Manual.

ACSERVER

ACSERVER permits **AllChange** programs to run in client/server mode on a 32-bit Windows network in an efficient manner. The Administrator should determine whether it is to be run and how it is set up.

ACSERVER is documented in the **AllChange** Administrator Manual.

Backupac

`backupac.bat` is a script file supplied with the Windows versions of **AllChange**. It backs up the database and associated log files to a copy.

`backupac` is documented in the **AllChange** Administrator Manual.

Acdbgwin

`adbwin` is an executable supplied with the Windows versions of **AllChange**. It helps the Administrator trace what is happening in the system.

`adbwin` is documented in the **AllChange** Administrator Manual.

Project Settings Editor

The project settings editor is a registry editor for **AllChange** projects. It may be used to modify the registry entries used by **AllChange**.

This should not normally need to be used as **AllChange** creates any required registry entries. However in certain cases it may be necessary to modify the registry manually.

The project settings editor is documented in the **AllChange** Administrator Manual

VC programs

AllChange uses version control programs supplied with **AllChange** at various times to accomplish low-level tasks concerning external operating system files. Principal among these are **Takeout** and **Putaway** to implement low-level version control (storing the actual versions of files in a history archive from which they may be extracted at any time), and **Build** to execute the required operating system commands for configuration building.

Refer to the **AllChange VC Tools Manual** for details of VC programs if required.

Running AllChange Programs

AllChange programs are run in the normal operating system fashion for launching programs, e.g. typing the name of the command or clicking on an icon.

AllChange programs check that the user invoking them is *licensed* as an **AllChange** user. The Administrator is responsible for adding all new users to the list of licensed users. Most **AllChange** programs also require that the **AllChange** database already exist when they are invoked for obvious reasons.

AllChange supports both named and concurrent user licensing models, in both cases all users must be registered (licensed) as discussed above. In the former case only the licensed number of users may be registered and in the latter only the licensed number of accesses to AllChange at the same time are permitted. If the maximum number of permitted accesses has already been reached then on starting ACE the user will be given an error message informing them of this.

The behaviour of **AllChange** programs may be influenced by command-line arguments, startup files, system-wide files, registry entries and environment variables. Most of these are documented in the **AllChange Administrator Manual**, as it is anticipated that this is information that only the administrator needs to know as in most cases **AllChange** will make use of and create/ modify these entries for you.

It is worth pointing out, however, that an important consideration when invoking an **AllChange** program is that the **AllChangeproject** — which contains the data and configuration — should be the right one for whatever the user is working on. For some sites there is no choice: there is only one project/database shared by all users i.e. there is only one project. For others there may be several different **AllChange** projects.

The projects that are available will be presented on starting up an **AllChange** application unless only one project is available to the user or a default project has been specified to be used always.

The `ACPROJECT` environment variable may be used to determine startup project. It is also possible to set this from the command line (or from an icon).

The following command line arguments are accepted by most **AllChange** programs:

`-Evar=value`

Allows environment variables to be set on the command line, which permits different icons to be configured for different environment and project requirements. Particularly useful is `-EACPROJECT =...`, which sets the **AllChange** project.

`-readonly`

Accepted by `ACC` and `ACE` only: causes the program to open the database for read-only access, so information may be retrieved but not altered. This may be used to prevent unintentional alteration of the data when read access is all that is required.

`-minimised`

Accepted by `ACE`: causes the program to be started minimised and in a non-interactive mode. This is used by integrations with third party applications which do not require that the user interact with `ACE` directly.

`-norestorestate`

Accepted by `ACE`: causes the program not to restore saved role, workspace and current working part. This is used by integrations with third party applications which do not require that the user interacts with `ACE` directly.

Getting Started

About Getting Started

In this section the **AllChange** GUI interface — ACE— will be introduced, as well as some basic setup hints.

ACE is invoked by selecting **Start | Program Files | Intasoft AllChange | AllChange**

ACE, the **AllChange** Environment, provides a typical GUI interface to **AllChange** and includes many customisable elements for ease of use. For example, the toolbar is customisable under Windows. **AllChange** may have been configured to run in client/server mode in which case all database accesses will be performed by the server. Alternatively it may be run as a standard application accessing the database directly. See the **AllChange** Administrator Manual for further details of client/server operation.

For general usage of the windowing system you are using see the appropriate user manual.

Basic Setup Requirements

The Basic Setup Requirements

AllChange must have been correctly installed on your network (a *full install*) and configured in order for it to work correctly.

It will then need to be installed on each workstation that is to use it. In order to install **AllChange** for a client PC a **Workstation Install** should be performed (run `setup\setup.exe` in the directory in which **AllChange** has been installed). This will set up various environment information that **AllChange** needs and stores it in the windows registry. See the **AllChange Administrator Manual** for details of registry entries and the environment.

AllChange has a concept of an **AllChange** project which comprises the **AllChange** configuration and its associated data. One or more **AllChange** projects must have been defined before you can use ACE.

AllChange will require certain information from your configuration and from your registry, from the command line or from the environment, in order for it to work correctly.

The following will be required:

- The **AllChange** system directory (`ACDIR`), this is where the **AllChange** software and system configuration files are to be found - this must be defined in your registry or on the command line. The installation program will define this in your registry for you.
- The **AllChange** project that is to be used - this will be prompted for unless a default has been defined. This means that an **AllChange** project must have been defined, this would normally be done for you by the **AllChange** administrator. The following information about a project is required:
 - The name of the project
 - The directory in which the project configuration and non database project data is to be stored — this is known as the *project directory*
 - A SQL server database

This information may be obtained from the project definition

- A database must exist for the **AllChange** project.
- You must be a registered **AllChange** user.

You may be registered as:

- A *full AllChange* user in which case you will have access to all aspects of **AllChange** within any access control rules that have been defined.

- A *cr only* user in which case only CRs and Monitors will be available to you for creation and modification purposes, although you will have read access to other parts of the database.
- A read-only user in which case you will be able to browse and view the data and run reports, but may not perform any sort of update.
- If you are a full **AllChange** user you will probably need access to a workspace .
- You will probably need to be assigned one or more user roles.

If any of these are not the case see your **AllChange** Administrator.

AllChange Projects

The concept of an **AllChange** project is an **AllChange** configuration and its associated data.

The data from one **AllChange** project may not be accessed from another **AllChange** project. If information sharing is required across more than one product (or development project), then all of them may be maintained in one **AllChange** project database (e.g. CRs may need to be common to all products).

If there is no information sharing between products, then it may be beneficial to maintain these in separate **AllChange** projects. This will reduce the size of a single database and improve performance.

A project definition comprises a minimum of the following information:

- The name of the project
- The directory in which the project configuration and non database project data is to be stored this is known as the *project directory*
- A SQL server database

The **AllChange** project used may be selected on startup or may be defined on the command line (`-EAC-PROJECT= . .` argument) or in an environment variable (`ACPROJECT`). If only one project is available for you, or you have elected to use a particular project, then this will be used automatically.

By using the command line, different projects may be accessed from different icons.

Home Directory

Certain configuration files are sought in the user's *home directory*. This is a directory which is unique to each user where he can create and edit files to tailor the system to his own requirements.

The home directory is taken from a `HOME` environment variable, or from a `HOME= . . .` entry in the registry; if neither of these is set then the NT `HOMESHARE`, `HOMEDRIVE` and `HOMEPATH` environment variables, if they exist are used.

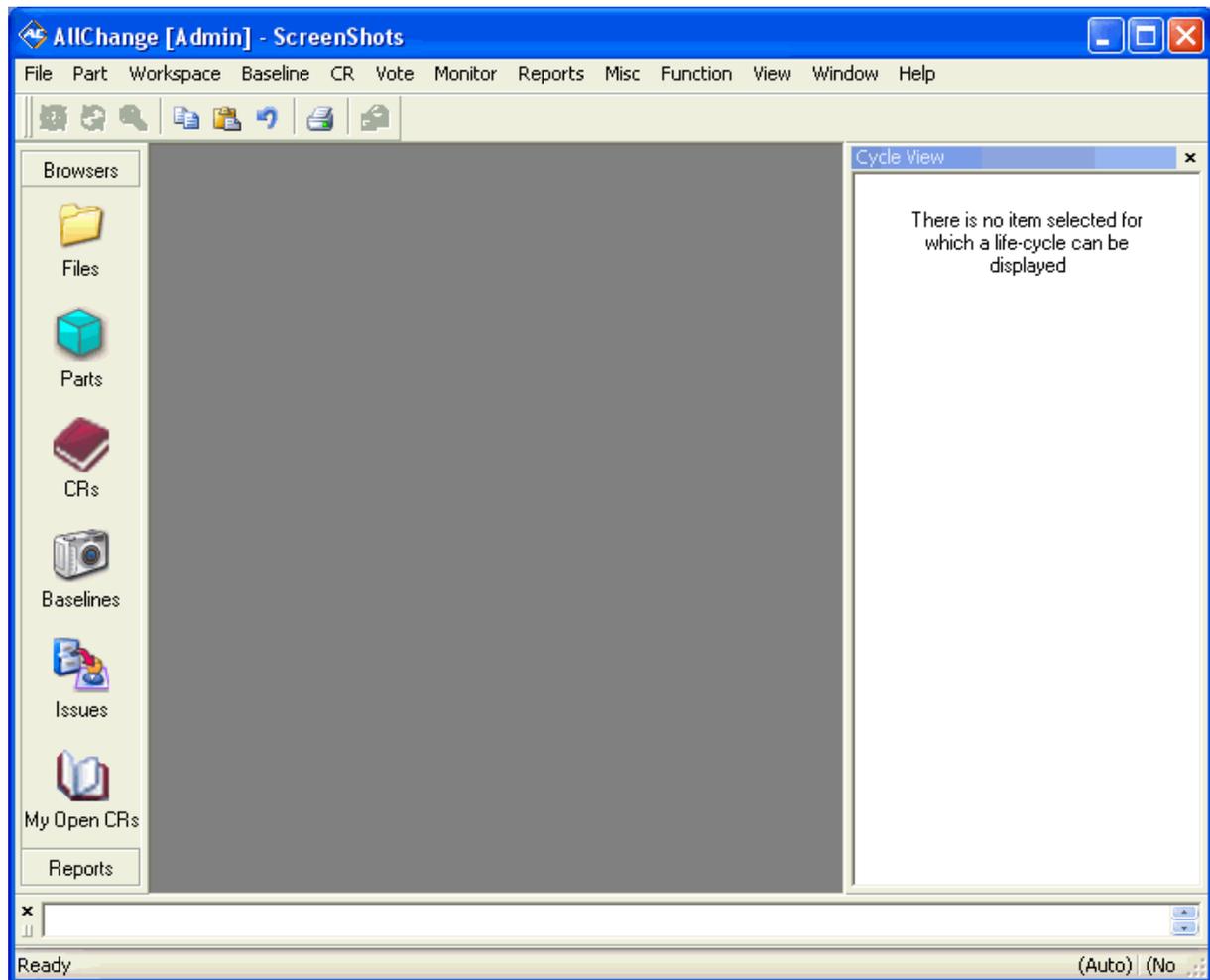
Alternatively, if a special **AllChange** home directory is desired, then an environment variable (or registry entry) called `ACHOME` may be created. If this is present, then it will be used by **AllChange** as the home directory instead of the real home directory. This can be used to separate off **AllChange** specific configuration and support files from other files in the users home directory.

`ACHOME` may be set by specifying the required directory in the **Misc** tab of the **Misc | Options** dialog in ACE.

The Main Window

About the Main Window

ACE supplies a multi-windowed GUI interface to the **AllChange** system.



The initial ACE screen for Windows is shown above. It comprises the following main features:

Main Menu: this is a standard menu offering access to the various **AllChange** functions

Toolbar: the toolbar provides access to common operations according to the currently active child window.

Shortcut bar: this provides user configurable icons which give access to browsers, reports and other useful shortcuts.

Folder List: this provides user configurable icons which give access to browsers, reports and other useful shortcuts.

Cycle View: a dockable window providing a graphic representation of the life cycle of items such as CRs, Parts and Baselines.

Output Window: a dockable window showing messages and the results of various operations

Status bar: this is at the bottom of the ACE Window and shows status information, progress information and messages.

Some of these features may be removed from the screen from the **View** menu.

The Window Title

The title of the **AllChange** window will show the **AllChange** project currently being used. If running client/server then it will also show this in the title. If the user is an administrative user then it will also **Admin**

Status Bar

The status information shown at the bottom of the main application window is:

Role

This shows the currently selected user role if there is one: this may be changed in the [Role Browser](#) (available from the **Misc** menu).

Workspace

This shows the workspace currently attached to, if any. To find out the current directory, hover the mouse over this pane of the status-bar. A tip will be displayed showing the current directory. The [Workspace Browser](#) allows you to attach to a new workspace (available from the **Workspace** menu).

NOACTION/NOCOND

This shows whether command and life-cycle actions/conditions are to be activated. Values shown may be:

- **NOACTION**: this indicates that no actions will be performed
- **NOCOND**: this indicates that no conditions will be performed
- **NOCMD**: this indicates that no conditions or actions will be performed
- **Blank**: this is the normal case and indicates that both conditions and actions will be performed.

Conditions and actions may be switched off using the **Set System Flag** dialog available from the **Misc** menu. This is only intended for use by the Administrator.

Double clicking on a statusbar item will pop up the appropriate window to allow that item to be changed.

The Toolbars

The toolbars provide access to commonly used actions appropriate to the window that is currently active (the *current window*). The toolbar items are displayed as icons.

Any toolbar item that may not be used for the current window will be disabled.

When selecting a toolbar button, if the current window is a browser then any items selected will be taken as the items to be acted upon. If it is a viewer then the item viewed will be taken as the item to be acted upon.

By default toolbars are *docked* under the main menu bar, but may be moved (by dragging with the mouse) to any position desired or *docked* at either side of the main window.

Each toolbar button will show a brief description if the mouse pointer is over it for a short time and a longer description in the status bar. To access the toolbar menu, right click on a toolbar or dock.

Toolbar settings may be reset to the default settings using **Misc | Clear Settings**. If your user type changes (e.g. from CR only to a full user) you should reset your toolbar settings in order to get the default toolbar settings for a full user.

Copy, Paste and Undo

If an edit control (including the edit control in an editable combobox) has focus, these work in the normal way, i.e. **Copy** copies the currently selected text from the edit control to the clipboard, **Paste** inserts any text in the clipboard into the edit control at the caret (cursor) location and replaces any selection. **Undo** will reverse the last edit operation performed including typing.

If the focus is not an edit control then the appropriate action will for the item selected will be made, in general this means that on Copy or Cut a note is made of the item selected and the action requested. On Paste the appropriate command dialog will be presented. Undo is not supported for non edit control selections.

The standard keyboard shortcuts are also available: `Ctrl+C` or `Ctrl+Ins` perform a copy, `Ctrl+V` or `Shift+Ins` perform paste, `Ctrl+Z` or `Alt+Backspace` perform undo, `Del` performs delete. In addition under the new shell (95/98 or NT4) clicking the right mouse button in an edit control brings up a menu with these items on.

The **Command Output** window also has a special copy item in its system menu. This will copy the selection or the entire contents if no selection is made.

The Shortcut Bar and Folder List

Both of the bars contain user configurable icons, which can be used to open a browser window, run a report, or invoke an ACCEL command or operating system command. This can be very useful for fast access to frequently run reports or queries run in a browser.

Both the Folder List and the Shortcut bar may be closed if not wanted.

See [Customising the User Interface](#) for further details.

The Cycle Viewer

Displays the life cycle of the currently selected item and allows its status to be changed or a vote to cast by clicking on the desired status (if the progression is valid for the current user profile).

See [cycle viewer](#) for further details.

The Main Menu

The Main Menu is divided into categories based on the different types of object that **AllChange** operates.

Selection of **File**, **Part**, **Workspace**, **Baseline**, **CR** or **Monitor** provides access to the *browse* and *view* windows for that type of object (i.e. windows allowing these objects to be examined). The *command dialogs* appropriate to operate on that type of object are also available (i.e. dialogs which allow actions to be performed on that type of object). Site specific functions may also appear.

Note that some or all of the above terms/nomenclature may have been mapped to in-house terminology in which case the menu option will appear using these site specific terms, see [Terminology](#) for which terms may be mapped

Selection of [Report](#) will provide access to the **AllChange** reporting facilities.

[Misc](#) provides access to various facilities including the role browser.

Selection of **Function** will lead to a menu of the site specific functions which have not been placed on other menus. Selection of any of these will invoke that function.

View provides access to facilities to control the items shown in the main window.

Report provides access to the integral **AllChange** report generation tool ACREPORT.

Selection of [Help](#) provides access to the **AllChange** help system.

Items on the pulldown menus will be disabled if they are not valid at the current time. Menu items are disabled according to definitions which may be tailored for each site (see [Customising the User Interface](#)).

The default configuration, for example, will disable menu items which require that you are attached to a workspace if you are not currently attached to a workspace.

The Misc Menu

The Misc Menu has the following items:

Set Role

Allows the current role to be changed.

Browse Roles

Provides access to the Role Browser window which allows the current role to be changed.

Eval

Allows an **ACCEL** expression to be evaluated

AllChange Command

Provides access to the command dialog which allows **AllChange** commands to be invoked using a command line interface.

Version Control Command

Allows the underlying version control tools to be invoked directly.

OSCommand

Provides access to the operating system command dialog which allows an operating system command to be invoked.

Options

This provides access to various user options/preferences.

Set System Flag

Provides access to the command dialog allowing various **AllChange** flags to be set.

Save Windows On Exit

If this is set then on exiting **AllChange** the size and position of any open windows is saved in the registry. On re-starting ACE these settings will be restored.

Clear Settings

Causes ACE user specific settings to be cleared. See [Customising the User Interface](#) for details

Select All in List

If the focus is in a list then this will select all the items in the list.

Refresh Windows

Refreshes all open windows from the database. This may be useful to ensure your information is current where there are concurrent users, or if **Autoupdate** is switched off.

Read

This causes the current window to be reread from the database. This may be useful if you wish to ensure that the information in the current window has not been changed by another user. If any changes have been made in a viewer to the current record and the record has not been **Updated**, then you will be informed of this and asked whether you want to discard these changes (i.e. re-read the record anyway).

Next

Reads the next record from the database for the current window

Previous

Reads the previous record from the database for the current window

Copy

Copies the current selection to the clipboard

Cut

Copies the current selection to the clipboard and marks it for deletion

Paste

Copies from the clipboard into the selection

Update

If the current window is a viewer and any changes have been made to the current record, then the database record will be updated. If the database record has changed (possibly by another user) since it was last read and displayed in the viewer then an update may not be performed until the record has been re-read.

The View Menu

The View menu provides access to facilities to customise the view of the main window. It provides the following facilities:

View

The View submenu allows the style of list views to be selected between detailed, small icons, large icons, list and folding views.

Show Obsolete Items

If selected then items which are obsolete will be shown in browsers (in a dimmed font). If not selected then they will not be shown at all.

Shortcut Bar, Folder List, Output Window:

If selected the appropriate item is shown, otherwise it is closed

Clear Output Window:

Clears any text currently in the output window

Preview Pane:

If selected shows the preview pane for the current up front browser - this setting is saved individually for each browser.

Cycle View Bar:

Allows the life-cycle for items in **AllChange** to be viewed and updated. The cycle-viewer is displayed in a docking window, which may be attached to any side of the main window.

Toolbar:

Provides access to the toolbar customization facilities.

Output Window

The Output window is used to display the output of any **AllChange** commands which have been issued.



The Output window will show the output of **AllChange** commands executed (e.g. report output if not sent to a file).

Hyperlinks will be shown as clickable links. Hyperlinks will be activated for common protocols such as "http", "ftp", "mailto", etc, and also for the "allchange" protocol, and any other registered **AllChange** protocols. The link is followed by single-clicking the link. Right-clicking on links will show a menu where the link may be opened or copied to the clipboard.

Output is appended to the current contents.

Under Windows the system menu on the output window has a **Copy** item. This copies the contents of any selected text — or the entire window contents if there is no selection — to the clipboard. The Windows keyboard shortcuts **CTRL+C** and **CTRL+INS** also copy selected text as usual. There is also a **Print** item on the menu which will print the contents of the Output window. A **Clear** menu item empties the contents of the window.

A log of activity during the current session or since the window was cleared may be shown in the output window as a history of **AllChange** command lines. This may be enabled using the **Set** command and switching on **echo**. The output window may also be used to show debug and trace information. See [Customising the User Interface](#) for details of output window options.

Browsers

The **AllChange** GUI is based around the use of browsers for the various aspects of the **AllChange** data. Each of these parts of the data is referred to as a database and the browser for that database will list its contents. It may be used to select the objects which are to be acted upon, thus providing an object based approach to using the system.

A browser takes the form of a child window (the browse window) or a tab on a viewer.

Any items selected in the current browser will be copied (if appropriate) to a command dialog when it is selected.

Browse windows are provided for the following:

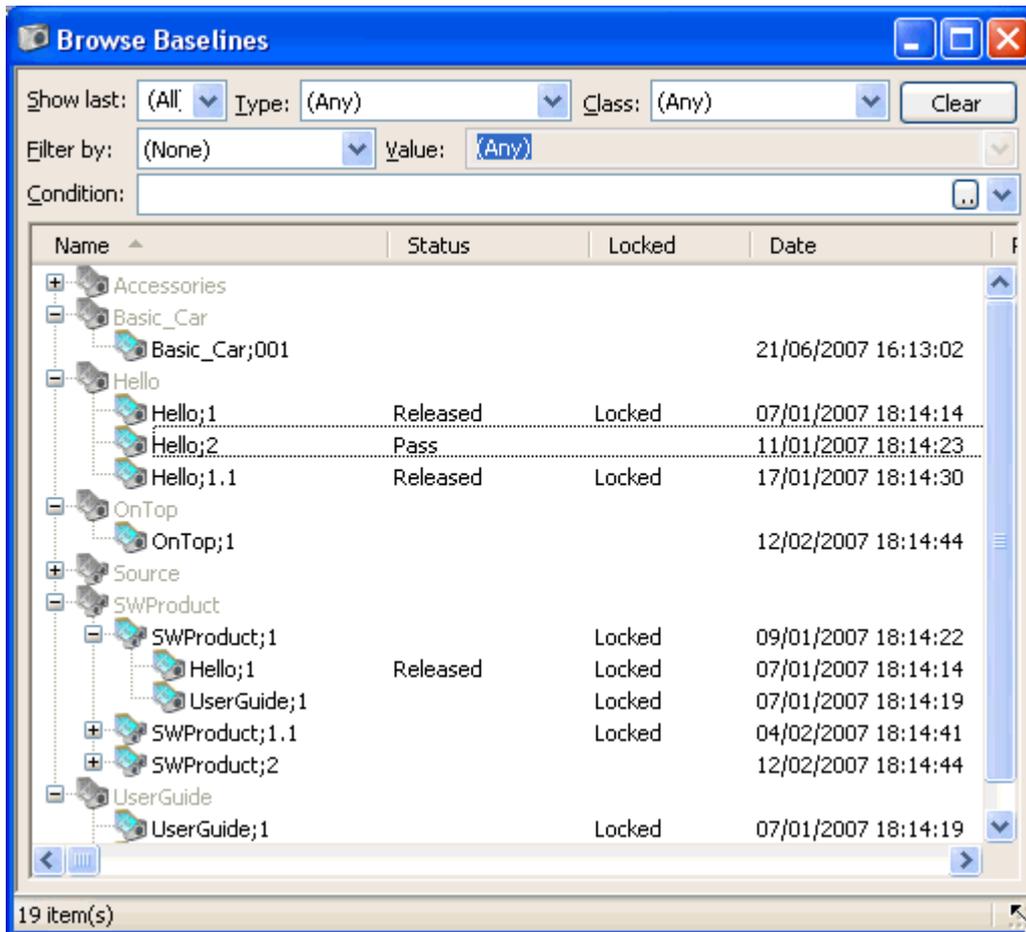
Files	Shows files and directories for workspaces and pools.
Parts	Shows the parts known to AllChange and allows the current working part to be changed.
Part Items Affected	Shows the relationships from parts to other crs, baselines and parts
Check-outs	Shows parts that are checked out.
Instances	Shows instances of parts that exist.
Baselines	Shows baselines that exist.
Baseline Items Affected	Shows the relationships from baselines to other crs, parts and files
CRs	Shows existing CRs.
CR Items Affected	Shows the relationships from baselines to other baselines, crs, parts and files
Monitors	Shows any monitors.
Workspaces	Shows any workspaces and allows a workspace to be attached to.
Roles	Shows any roles and allows the current role to be changed.

Note that some of the above terms may have been mapped to site specific nomenclature, see [Terminology](#) for which terms may be mapped

Most browsers have various mechanisms for filtering what items are shown. Sometimes the criteria specified in a filter may mean that the browser can take a long time to be updated/filled depending on the size of the database being searched. The updating process may be aborted by pressing the `ESC` key as indicated in the status bar at the bottom of the screen.

Browsers display various information about the items in the database being browsed. This will be displayed in a tabular form with columns of information about each item.

The baseline browser is shown below as an example.



The leftmost column of all browsers will show the name/identifier for each item/database record shown in the browser. The contents of this column is fixed. For some databases (e.g. check-outs), additional information is required to identify the item (the part name and the workspace checked out to for check-outs); in this case the second column will also be fixed. This is in order to allow ACeTo correctly identify the items selected in the browsers.

The other columns of information shown in a browser may be tailored to individual requirements.

The following features are supported by browsers:

Filter By

Allows the browser data to be filtered by the **Value** selected/specified. Pattern matching may be used where the Value is an edit control: use * to represent any character e.g. starts-with*, *ends-with, *contains1*contains2*. All data fields valid for the database are offered as filters. For CRs & Baselines, Text is offered among the filter fields, the **Value** for these is always treated as though it had * at start and end (regardless of whether you type them), so always a "contains" match.

Condition

Allows an arbitrary condition to be specified in ACCEL to further filter the browser content. The ... button provides access to the [ACCEL condition editor](#) which provides a user friendly interface to the a subset of the full language facilities.

The condition is parsed and, if possible, corresponding clause(s) injected into the query sent to SQL. Where this happens the search can be fast.

- The parser can only handle a limited subset of ACCEL. The simpler the condition, the

more likely it is to be included in the SQL query, e.g. a single simple <field><-operator><value> is likely to succeed.

- The parser is designed to handle the typical ACCEL generated by the ACCEL condition editor. It knows about the code generated for the various operators offered there, and common in-built ACCEL functions (e.g. `match_wild()`) and user-defined functions (e.g. `call(IsArbFieldDefined, ...)`).
- The parser can handle an ACCEL **and** operator in a condition, but only some **or** and **not** operators.

Sort by Column

For all browsers the browser contents may be displayed sorted by any column which is displayed. Simply click on the column title and the window will be redisplayed sorted by the column selected.

Column Reordering

The order in which the (non fixed) columns are presented may be changed by dragging the column to the desired position, see [Defining Browser Contents](#).

Column Selection

The (non fixed) columns which are displayed may be selected according to personal preferences. Right click on a column heading and select **Add/ Remove Column**, see [Defining Browser Contents](#)

In-line Rename

Items shown in browsers may be renamed in-line (where appropriate) by clicking twice on the item in the browser.

Del Key

The `Del` key may be used to delete the currently selected items in a browser. When an item is deleted an option becomes available to delete any references to other items the deleted item may have had.

Preview Pane

Shows a *preview* or summary of the content of an item selected. The preview pane may be switched on and off for each browser from the **View** menu. Hyperlinks will be shown as clickable links. Hyperlinks will be activated for common protocols such as "http", "ftp", "mailto", etc, and also for the "all-change" protocol, and any other registered **AllChange** protocols. The link is followed by single-clicking the link. Right-clicking on links will show a menu where the link may be opened or copied to the clipboard.

Print

Allows the content of the browser to be printed. The print output will include the name of the browser or viewer the list is part of, and, for viewers, the name of the list. Prints from browsers will list the current browser filter, condition etc. Find window prints will include a description of the search criteria used to produce the results.

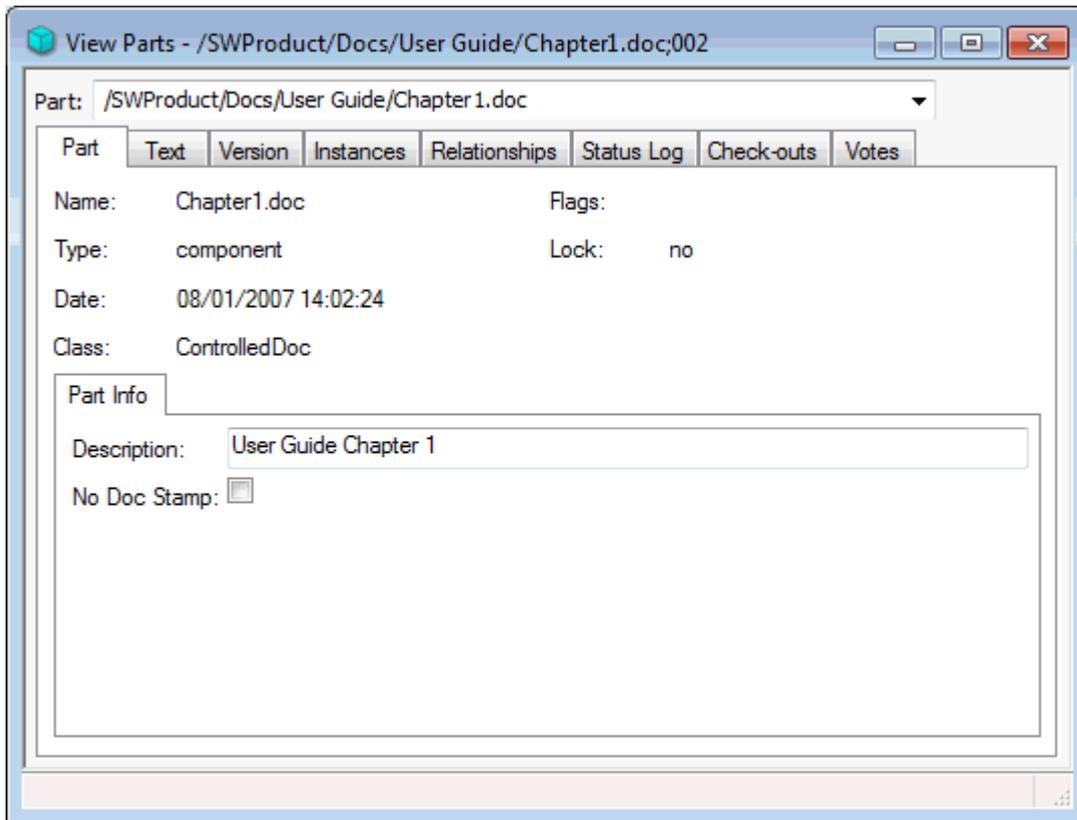
Views

Allows the browser content to be displayed in different ways. Select **View | View** or right click in unused space and select **View**, see [Views](#).

Viewers

Several databases have a Viewer. This is a window in which the details of one item are shown ([Parts](#), [Check-outs](#), [Instances](#), [Baselines](#), [CRs](#) and [Relationships](#) have viewers. *Note that some of the above terms may have been mapped to site specific nomenclature, see [Terminology](#) for which terms may be mapped*).

The Part viewer is shown below, this is available from **Part | View**.



The item for which details are currently shown will appear as part of the window title.

The item for which the details are to be shown may be selected:

- by entering into the edit control at the top of the window: this may specify the complete name of the item or a partial identification in which case the first match will be used.
- by selecting the item in an appropriate browser or list in a viewer tab and:
 - Double clicking.
 - Select **View** from the appropriate pull-down menu
 - Select the view toolbar button
 - Right Click and select **View** (this is the same as double clicking) or **View in new window** (this will open a new viewer window for the selected item) from the context menu.

The viewers contain alterable items (Edit Controls, Option Controls, etc.) for elements of the item shown that may be altered in the viewer directly. (Do not forget to **Update** the database after making changes, *Return* will perform an update or **Update** is available from the **Misc** menu or the toolbar).

Some edit controls may contain hyperlinks. Double clicking on the hyperlink to jump to the link. The link may be edited in the normal way as an edit control, but in addition a hyperlink editor is provided which may be accessed from the right click menu or CTRL K.

Most controls will show a tool tip when hovering over the control.

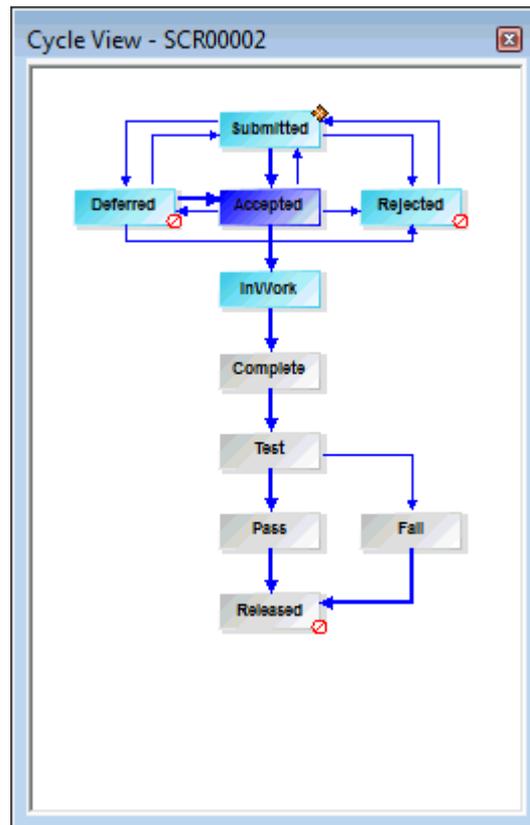
Fields which are not alterable may often be changed by other dialogs which are available either from the viewer directly or from the pull-down menu for that database.

In 'Add' dialogs some fields are compulsory. Compulsory fields are defined with their labels in red, Tabs containing compulsory fields will also be shown in red.

If a viewer is the current up-front window then the item shown will be copied to any appropriate command dialogs.

About the Cycle View

By selecting **View | Cycle Viewer** it is possible to see a graphical representation of the [life-cycle](#) of items such as CR's, Baselines and Parts in **AllChange**. The viewer is displayed in a docking window, which may be attached to any side of the [Main Window](#).



The viewer displays each status in a different colour indicating which is the current status, whether it is a vote status, the valid and invalid progressions from the current status, invalid progressions dependant on the current user, and progressions which are driven by other life-cycles. The Default progression and the other alternative progressions for this life-cycle are also indicated by colours.

If the status is classified as a Closed status then a  will be shown at the bottom right of the status box.

If the status is a vote status then a  will be shown at the top right of the status box.

Each of the colours used for this display may be tailored by the user from an [Options](#) dialog available from a menu displayed by right-clicking in the empty space in the cycle viewer.

As well as the statuses there may also be annotations providing useful information about the life cycle.

Hovering the mouse pointer over a status or progression will show a tool-tip for that status or progression which may also provide useful information.

The status of an item can be changed by using the cycle viewer. With a cycle displayed in the viewer double click on a status which is a valid progression or right click and select **Move to Status** and the status will be updated. If the status of a part is being changed in this way, and the status has the CheckOut or CheckIn attribute, then a CheckOut or CheckIn action is performed rather than a direct status change.

A vote may be cast using the cycle viewer. With the cycle displayed in the viewer double click on the vote status or on a status which is a valid progression and can be voted for, or right click and select **Cast your Vote**. This will open the [Cast Vote](#) dialog allowing you to cast a vote or modify a vote already made. It also allows you to pass the vote to the next set of voters in a serial vote.

If this is prevented by the standard restrictions on this progression, then the user will see the usual error message, and the status change will not occur.

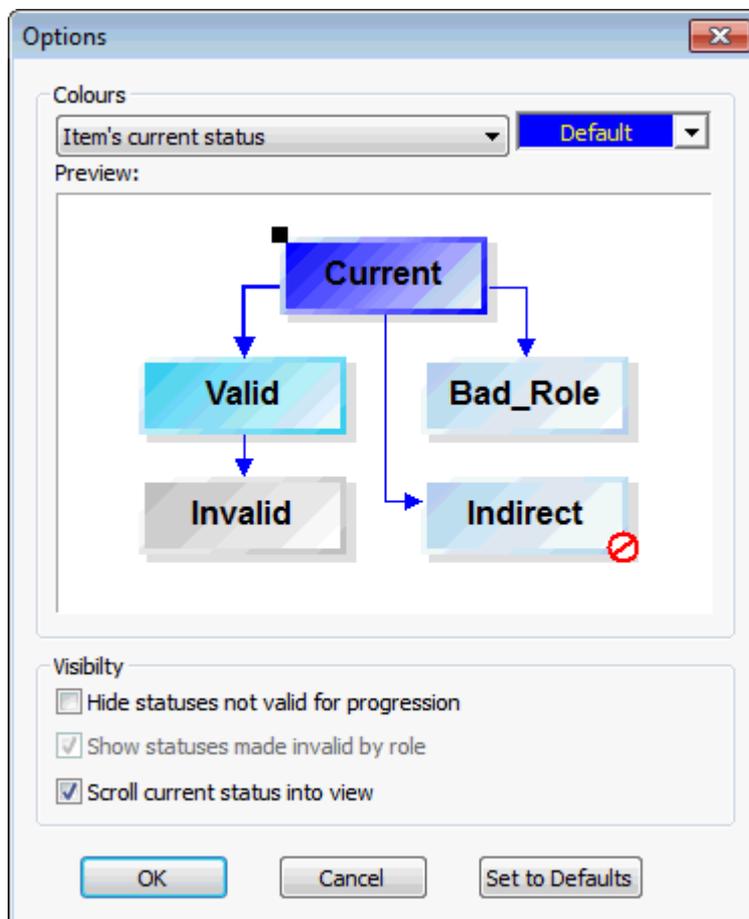
When clicking on an item in a browser, or opening an item in a viewer, the cycle viewer will be updated to display the cycle for the item. If no item is selected, or if multiple items, or items with no class or cycle are selected, then the viewer displays an appropriate message.

Cycle Viewer Options

Various features are available by right-clicking in the empty space in the cycle viewer:

Options

Opens a dialog controlling the appearance of the cycle in the viewer allowing [colours](#) and [visibility](#) of statuses to be controlled. The **Set to Defaults** button will reset all selections to the supplied out-of-the-box settings.



Print

Along with standard print options **Page Setup** allows the Header and footer to be defined for the printed version of the life-cycle

Zoom

Enlarges or reduces the contents of the viewer. Use **Zoom to Fit** to cause the cycle to be automatically scaled to fit into the cycle view window.

Export to Image

Allows the cycle view diagram to be exported to an external image file for inclusion in word-processing documents etc. A variety of image file [formats](#) are supported and may be selected in the **Export to Image** dialog. Note that statuses/progressions which are not visible according to user and administrator options will not be included in the exported image.

The image file formats supported are Bitmap (BMP), Meta-file (WMF), JPG, TIF, GIF, PCX and PNG.

Some of the image file formats have settable options. The options appear on the **Export to Image** dialog. Meta-file (WMF), GIF, and PNG files can be made transparent, by checking the check-box on the dialog. JPG files can have their encoding set to Standard or Progressive, and can have their quality set, from an options dialog available from the Options button on the "Save As" dialog.

WMF files are saved as vector graphics, which allows for resizing. All other files are bitmap images. GIF and TIF images are saved in 8-bit colour. The background is set to white for all images. Therefore, where images are set to have a transparent background, all white pixels are made transparent, except for Meta-files, where they are simply drawn with no background.

Colours in the Cycle View

Statuses are displayed in different colours specified by the following:

Items current status

Indicates the current position of this item in the cycle

Statuses valid for progression

Indicates statuses which may be progressed to from the current status

Statuses not valid for progression

The item cannot be progressed to these statuses with this colour

Statuses not valid for user's role

The current role does not allow the item to be progressed to statuses with this colour

Statuses progressed to by other cycle

Indicates statuses marked with an attribute to indicate that progressing must be performed from an enclosing life-cycle

Current status awaiting vote

Indicates that the current status is awaiting a vote

Statuses reached only by a vote

Indicates statuses which can only be reached as a result of a vote

Normal Progression

The expected progression for this item

Default Progression

The default status for this item if progressed by *AllChange*

Visibility of statuses in the Cycle View

Hide statuses not valid for progression

Use this option to hide statuses which cannot be progressed to and are not relevant at this time

Show statuses made invalid by role

Use this option to see statuses which could be progressed to if the user was using a different role

Scroll current status into view

Use this option to cause the current status to be automatically made visible, if it was not already visible, in the cycle view window on a status change by scrolling

Note that administrators may override the visibility of statuses in the cycle view

Getting Help

Help within ACEwill invoke HtmlHelp.

Help is available by various different means:

User Manual

- via a `HELP` button on a menu; this will give the help most appropriate to the menu
- via the Help pulldown menu.
- via the `F1` function key which will give help appropriate to the current menu and field.

Creating and Managing Parts

About Creating and Managing Parts

All configuration items (CIs) making up a product or system that are to be managed by **AllChange** are defined to **AllChange** and referred to as *parts*.

Parts form the central core around which most operations are based.

Note that the term part may have been mapped to site specific nomenclature, e.g. CI (Configuration Item) in which case all references to part will show using the mapped nomenclature instead.

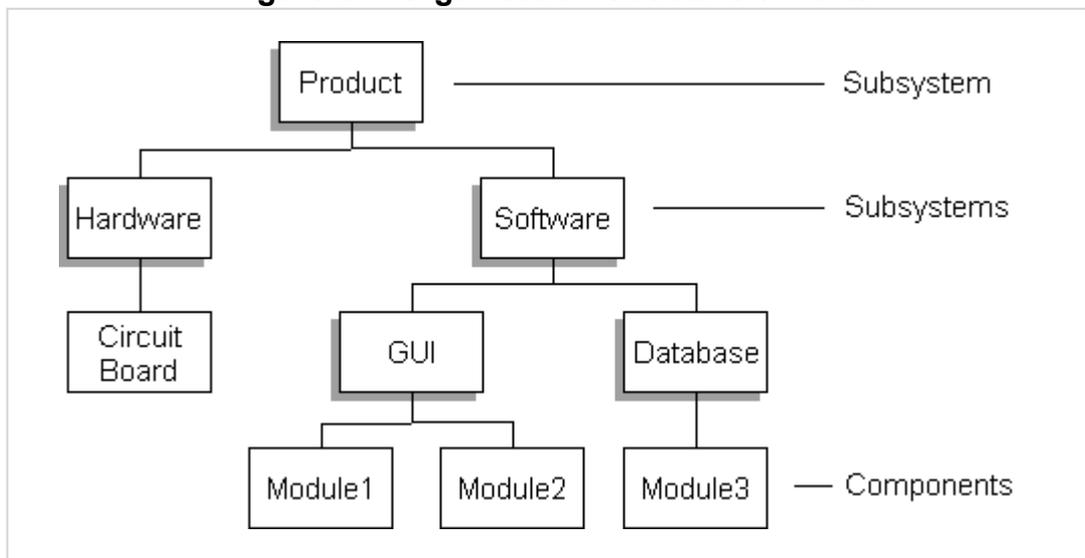
Parts and the Hierarchy

Each CI declared to **AllChange** is known as a *part* and information about each part is stored in an underlying database — the *parts database*.

The parts are defined to **AllChange** as a hierarchical organisation structure of *subsystems* and *components*. Subsystems are analogous to directories and usually have a physical directory associated with them. Components are analogous to files and usually have a physical file associated with them (e.g. source modules, documentation, design specifications). They may also refer to external objects which do not have an on-line file (e.g. hardware, paper documents). **AllChange** will track the version history of all components. Furthermore, versions of components may have instances associated with them. This may be used, for example, to reflect the manufacturing of specific instances of an item or specific delivery of a copy of software.

This organisation models the product structure and provides the framework for most operations.

Figure 5.1: Organisation Structure of Parts



[Figure 5.1](#) shows the relationship between subsystems and components; the product is divided into software and hardware elements, the hardware is comprised of a circuit board and the software comprises a graphical user interface and database access routines. The GUI is comprised of two source modules and the database access routines are provided by one source module. The software code for these two elements of the software could, for example, even be implemented in different programming languages.

Each part has a name and a parent allowing the hierarchical definition of a system to be defined. This is analogous to an operating system filing system where there are files and directories — directories may have children which are either files or directories, but files have no children.

Parts may be of type *subsystem* or *component*. In [Figure 5.1](#), **Product**, **Hardware**, **Software**, **GUI** and **Database** are all subsystems and will probably represent a physical directory.

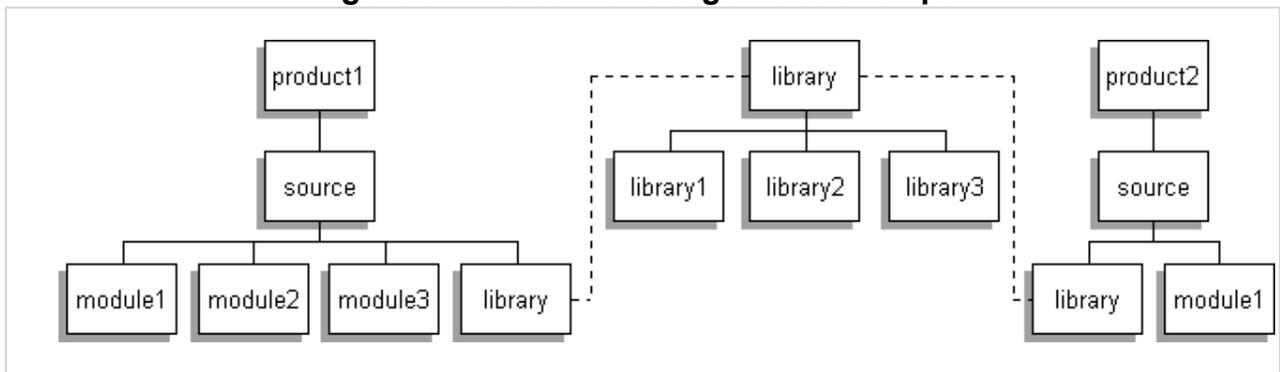
Subsystems may have children which may be further subsystems or components. Components may not have any children (but may have *versions* associated with them).

In [Figure 5.1](#) the **Software** comprises three components which are called **Module1**, **Module2** and **Module3**; these will have corresponding on-line files which are used to store the version history of the files. The **Hardware** comprises one component called **CircuitBoard**; in this case it does not represent a file as it represents a hardware item which makes up the product, though it may still have versions associated with it in order to track the existence of various versions without physically storing the contents.

In addition to straightforward hierarchical relationships between parts it is also possible to define *usage relationships* and to specify the parts which are *affected by* a particular part.

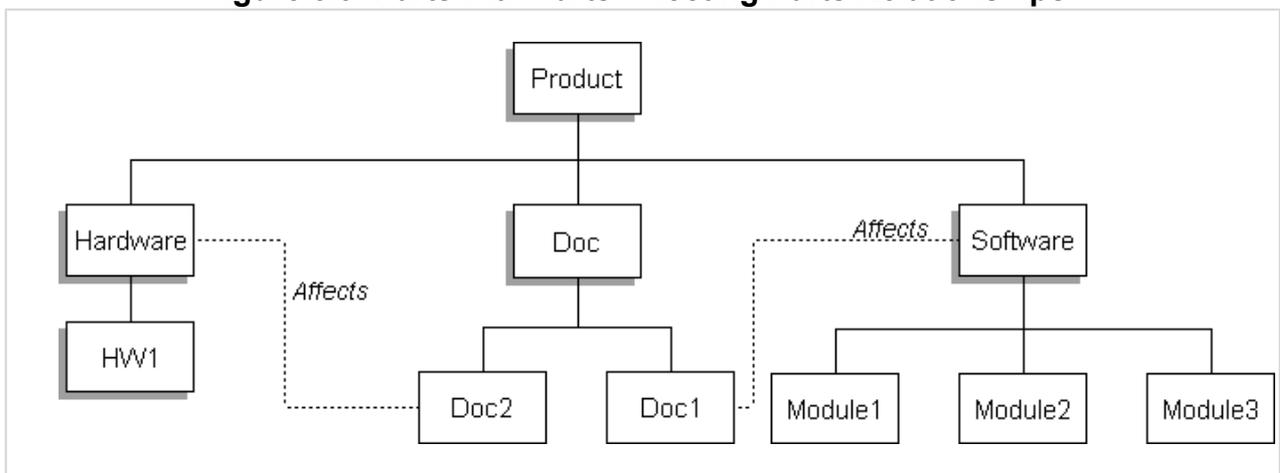
These relationships are illustrated in figures [5.2](#) and [5.3](#).

Figure 5.2: Parts with Usage Relationships



Usage relationships are generally used to express the re-use of a subsystem in several products as shown in the example in [Figure 5.2](#) where a *Library* is used by two *Products*. See [Uses Type Parts](#) for further details of usage relationships.

Figure 5.3: Parts with Parts Affecting Parts Relationships



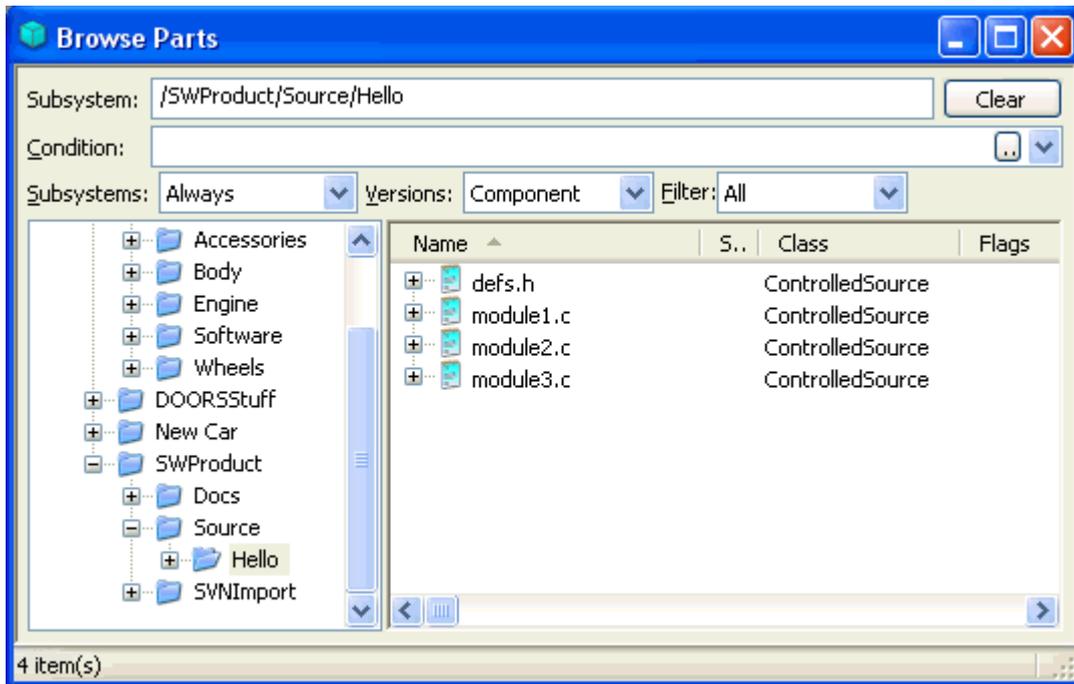
Parts Affecting parts relationships may be used to denote dependencies for impact analysis purposes. See [Relationships](#) for further details of parts affecting parts relationship

Browsing the Parts Hierarchy

About Browsing the Parts Hierarchy

The part browser (available from **Part | Browse**) allows you to view the parts that have been defined to **All-Change** and the relationships between them.

The part browser will probably act as the main focus for using **AllChange**.



The part browser has a similar look and feel to Windows Explorer; it contains a *tree-view* in the left hand pane showing the subsystems which are defined and a *list-view* in the right hand pane showing the contents of the current working part (shown in **CWP**).

The parts in both panes will be shown with an icon to the left of the part name to convey information about the part. The icons used are shown in the table below:

Icon	Description
	Subsystem
	Uses
	Component (not issued)
	Component (issued read only)
	Component (issued for edit)
	Version (not issued)
	Version (issued read only)
	Version (issued for edit)
	Instance

The parts shown in the right hand pane may be changed by selecting or double clicking on the required subsystem in the left hand pane.

Context menus are available on items in either pane (i.e. right click on selected item(s) and a menu will be shown with the available operations that may be performed on the selected items).

Right clicking with no items selected in blank space in the right hand pane will show a menu allowing you to modify the presentation of the right hand pane. The following options are available:

View This provides a sub menu allowing you to select the style with which you would like to see the parts shown in the right hand pane. (cf. Explorer)

Show Issue State If this is selected then the icons shown for the parts will reflect the issue status of the part with respect to the *current workspace* and shown above – see [Checking Files In and Out of](#)

AllChange. Note that if this is selected the time taken to display the right hand pane will be significantly longer.

If the **Detail View** is selected then the right hand pane will show various information about each part. The precise information and the order in which it is displayed is tailorable to each users requirements. The personalised settings will be saved in the registry.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title.
- The information may be sorted by any column by selecting the column title.
- The columns may be reordered by dragging them and dropping them in the required position.
- Print
- [Folding View](#) showing the version tree for components and instances of versions

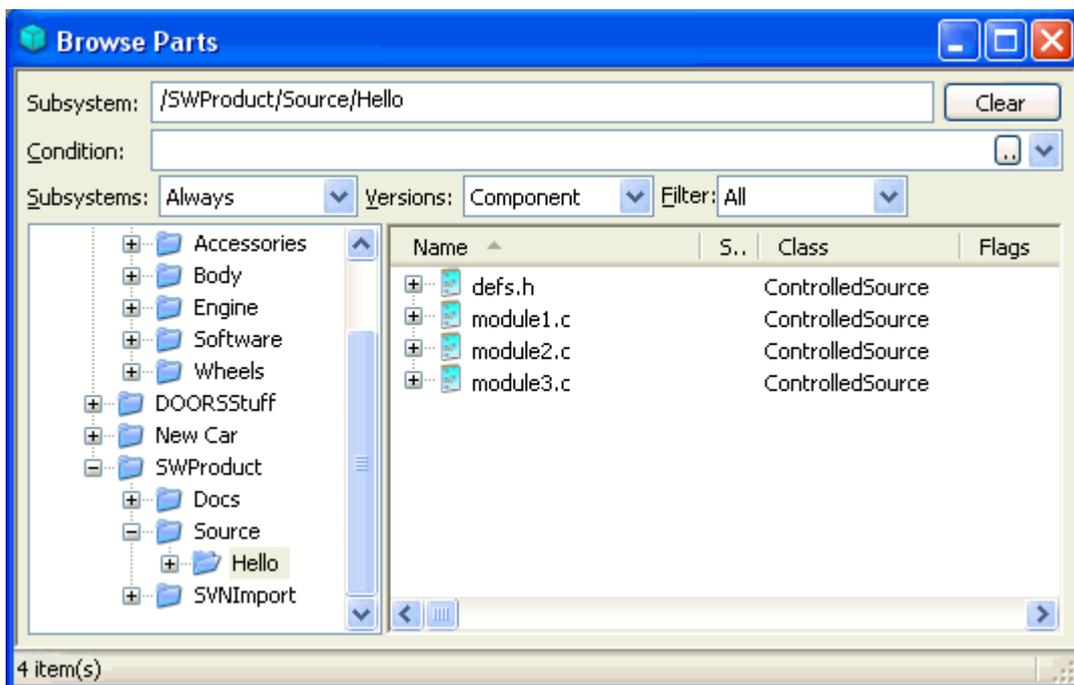
These features are common to all browser windows, see [Browsers](#) for a more detailed description.

In addition to the part browser there is a **Find** facility which provides a single list of all parts which match a pattern or other criteria, see **Find** for details.

About Part Paths

Parts are named/ identified using a path to the part and terminated by the name of the part. This is called a *part name*.

The part path is a list of subsystem-type part names separated by the / (slash) character. (This is identical to UNIX paths and very similar to Windows paths.)



The path for the current working part (**CWP**) is shown by the *open* subsystem icons in the subsystem list.

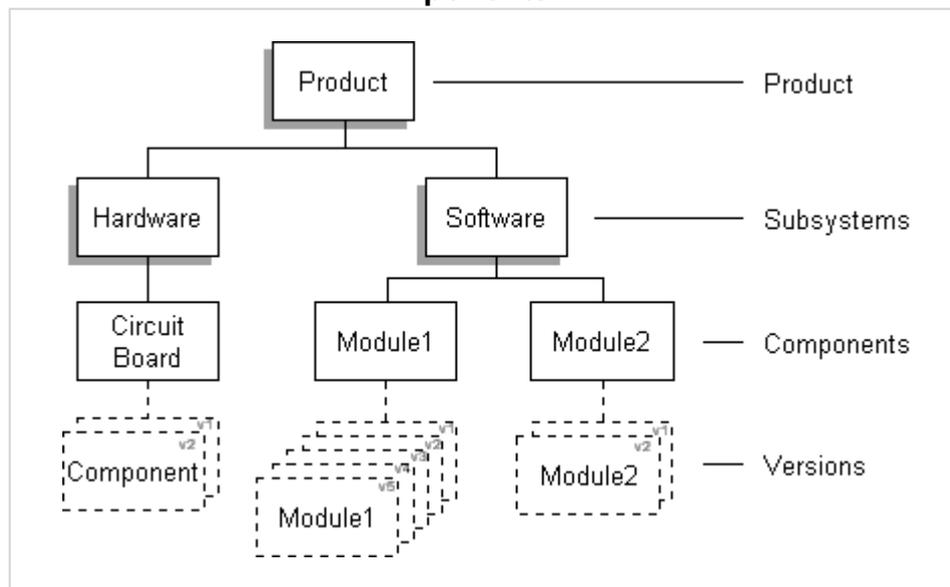
The path for the current working part in the screen shown above is identified by joining the subsystems which make up the current part (product + sw + source) with a / character to give /product/sw/source. subsys1 is a child of /product/sw/source.

The part named / is the *root* of the part definition tree and always exists. When ACE is entered using a new database, only the root part will exist. All parts are descendants of /.

Part Versions

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. See [Figure 5.4](#).

Figure 5.4: Organisation Structure showing Versions of Components



Each version of a component part will have a corresponding version in the external version history (VC) file, unless there is no file (e.g. the version represents the version of a hardware item or paper document). This stores the actual physical contents of that file version, while the database stores information about the existence of that version and its state etc. The parts are therefore a secure archive of the items making up a product, their organisation and the revisions that each component has been through.

Version numbers are generated automatically by the system when creating a new version. Versions are numbered sequentially starting from 1. A version may be identified by adding ;*version-id* at the end of the part name (e.g. `myfile;003`).

The System Administrator may configure the system to define the number of digits required for a version identifier (including no fixed number) — the default is 3 digits.

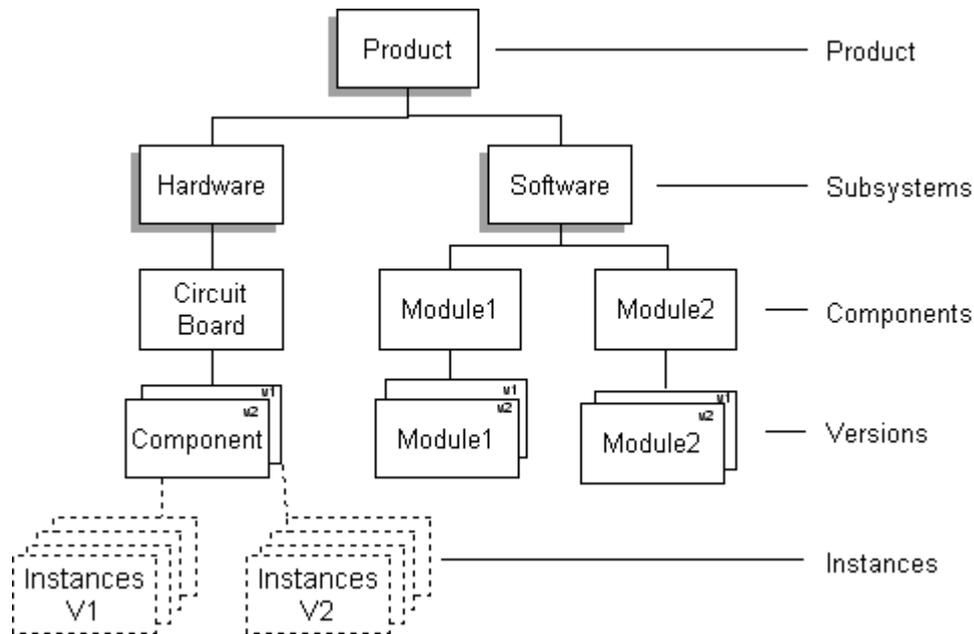
If the **All** option is selected in the **Versions** combo-box on the part browser, then all the versions of each component will be displayed.

Each component has a *default* version, which will normally be the latest version. The default version will be used whenever a specific version is required but a particular version number has not been specified.

Default may be selected in the **Versions** combo-box to show just the default version.

Instances

Component versions may have instances associated with them, these can be used to represent specific occurrences of a version. This can be useful, for example, to represent the manufacture of hardware items or the distribution of software items.



Instances are automatically numbered when they are created. Instances may be identified by adding : *instance-number* to the end of the version id, e.g. mycomponent;003:0004 would identify instance 4 of version 3 of mycomponent.

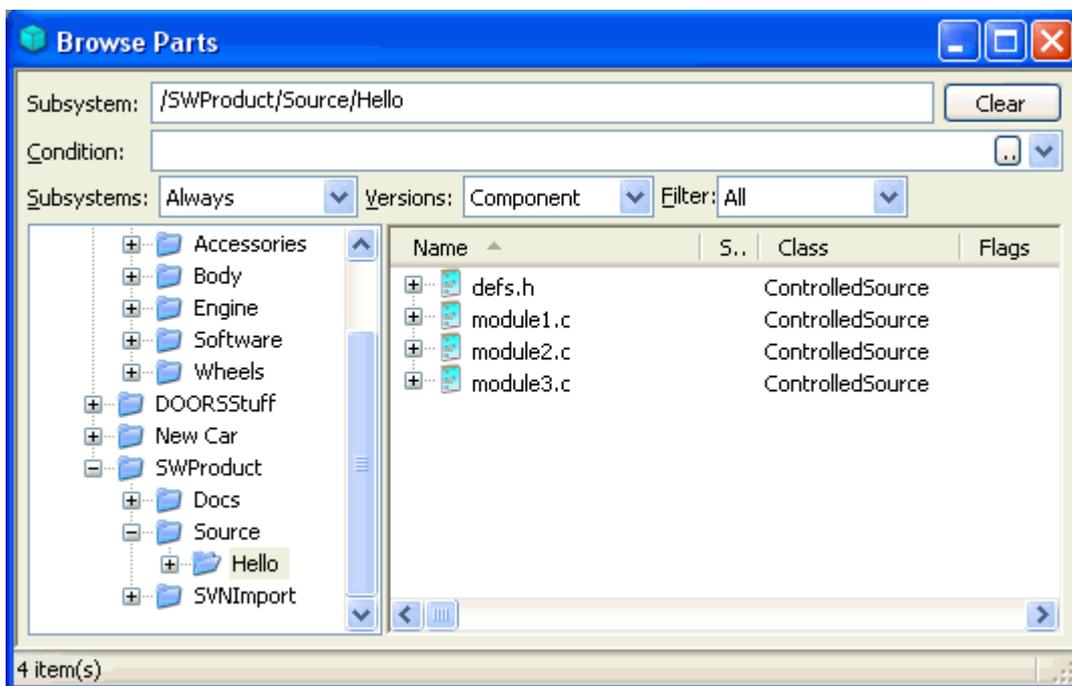
The **AllChange** Administrator may configure the system to define the number of digits required for a version identifier (including no fixed number) — the default is 4 digits.

In the folding view in the parts browser, a version may be expanded to show its instances.

Instances may be disabled by the AllChange Administrator in which case nothing referring to instances will be available.

Part Browser Filters

The part browser has various filters to change the information displayed.



Subsystems combo box determines whether subsystems are to be shown in the selection list.

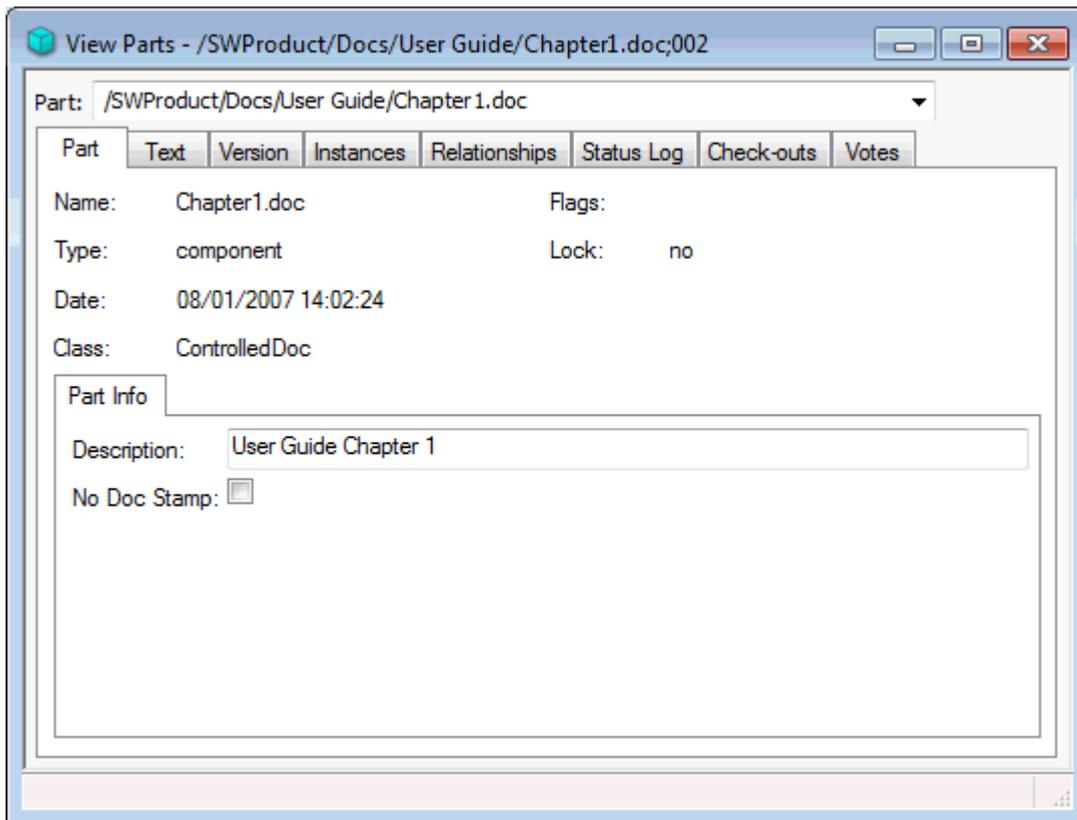
	<p>Always means always show subsystems</p> <p>Never means never show subsystems</p> <p>Apply Condition means show subsystems which match the condition specified combo box determines which versions are to be shown in the selection list:</p> <p>Component means no versions, just show the component. If any other value is selected then only versions of a component will be shown (<i>not</i> the component itself).</p> <p>Default will show the default version. This will normally be the latest version, but may be modified to your specific requirements.</p> <p>Top will show the latest version on the main line of development.</p> <p>Registered will show the registered version if there is one, otherwise the top version.</p> <p>All will show all versions.</p> <p>All Edit will show all versions including those which have the <code>NoVersion</code> flag and are just reserved because the part is checked out for edit.</p>
Versions	
	<p>combo box filters out the components/ versions to be shown in the selection list:</p> <p>All shows all items as selected by Versions.</p> <p>None shows no versions or components.</p> <p>Check-outs shows items which are checked out read only to the current workspace.</p> <p>Check-outs for Edit shows items which are checked out for edit to the current workspace.</p> <p>Any Check-outs shows items which are checked out read only or checked out for edit to the current workspace.</p> <p>Not Check-outs shows items which are not checked out to the current workspace.</p>
Filter	
	<p>allows a condition to be specified. All parts shown in the selection list must match the criteria specified by the condition. This could be used, for example, to show only those versions which were created on a particular date. Instead of an ACCEL condition a pattern match may be expressed and only those parts which match the pattern will be shown (e.g. <code>*.c</code> will show only those parts ending in <code>.c</code>).</p>
Cond	

View Part Information

About View Part Information

Information about the parts in the system is stored in the parts database. Parts have a number of fields and relationships to other databases associated with them. The fields and relationships may be set by various different commands or used for querying/ reporting.

To view the details of a particular part use the part viewer; this may be accessed by **Part | View**, or double clicking (or using the right click context menu) on a part in any list of parts (e.g. the part browser).



The information about the selected part is shown in a series of tabs. Select the tab required and the information for that tab will be shown. The tabs for the part viewer are:

Part

Shows general information about the part which is applicable to subsystems and components.

Text

Shows the descriptive text associated with the part and version.

Version

Shows information about versions of a component . The tab will only be available if the **Part** tab is showing a component type part.

Instances

Shows information about instances of versions of a component. The tab will only be available if the **Part** tab is showing a component type part and if instances are enabled.

Relationships

Shows information about relationships to other items such as CRs, Baselines and other parts.

Status Log

Shows the status audit trail for the part.

Check-outs

Shows the workspaces to which the part is checked out.

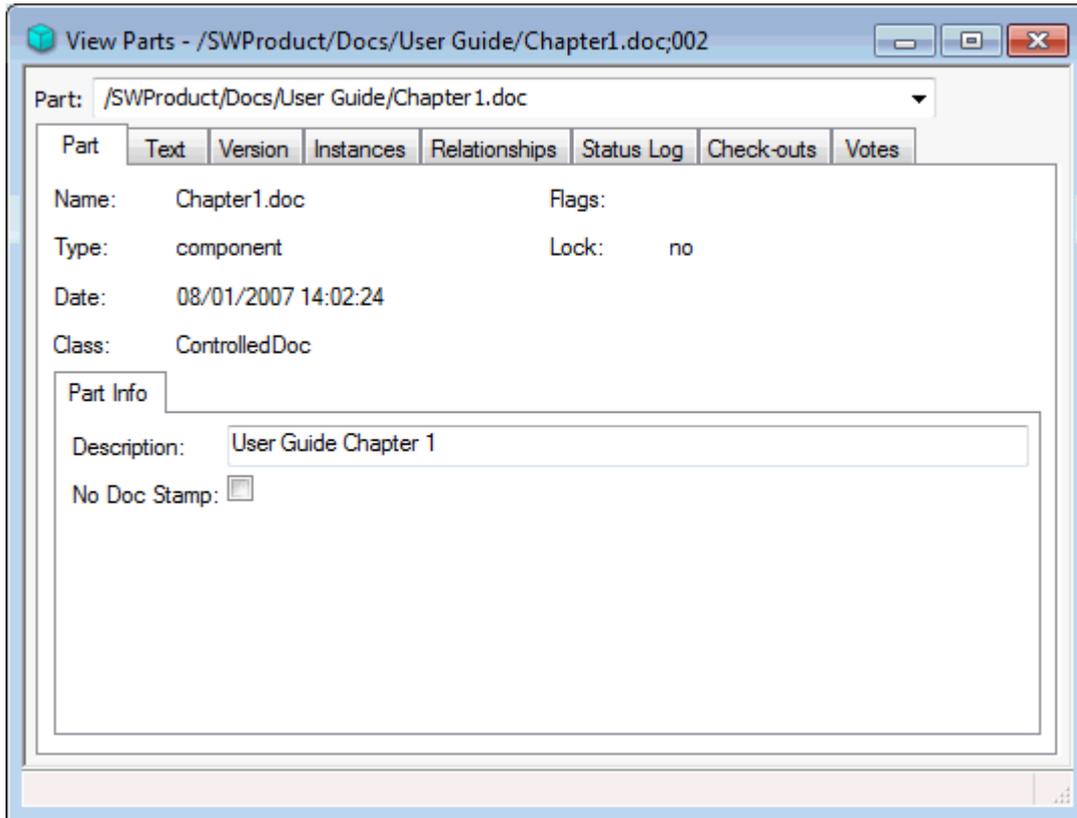
Votes

Shows the votes associated with the part. Note that this will only be present if voting is enabled.

The Part Tab

The Part Tab

The part tab shows fields of the subsystem or component.



The fields are:

Name:

The name of the part. The set of characters permitted in part names (excluding the version id) encompasses most of the characters allowed in Windows filenames. This is intended to make it easy to have the same part names as filenames, especially when importing a group of existing files into **All-Change's** control. The characters allowed are:

Only ASCII printable characters (i.e. top bit not set, no control characters)

None of the characters \ : * ? " < > | ^ = ! / ;

The **first** character must not be one of - @ \$

Although the ' (single quote) character is allowed, in view of its special significance as an ACCEL quote character we do not guarantee that it will not cause problems in certain circumstances and would advise against using it unless absolutely necessary.

Type:

The type of the part (subsystem, component or uses).

Class:

The class of the part.

Status:

The current status of the component or subsystem. Use **Part | Status** to change the current status. This will only be displayed if subsystem or component life-cycles are being used.

Lock:

Part is locked against change (Yes or No).

Arb1... Arb40:

Arbitrary, site-specific information about the part. The field names may be configured as required for your site.

Flags:

Special characteristics of this part:

Part Type

The type of a part may be one of:

- subsystem
- component
- uses

Note that the terms subsystem and component may have been mapped to site specific nomenclature in which case all references to either of these will show using the mapped nomenclature instead.

Subsystem type parts may have children which are either of type component or subsystem. Subsystem type is analogous to a directory in an operating system filing system and will have a directory associated with it.

Component type parts are at the bottom of the organisation hierarchy and have no children, but they have versions. Component type is analogous to a file in an operating system filing system and will have a file associated with it which will contain the complete version history of the file that the component represents.

Components represent actual physical components of a system or product, although these need not be machine readable items.

Uses type parts indicate that this part uses another part, see [Uses Type Parts](#) for details.

The type of a part is determined at the time that the part is created and may not be changed.

Part Class

The class is used to classify the part, e.g. source code, documentation or hardware. Classes are site specific and should be defined by the **AllChange** Administrator.

Note that the term class may have been mapped to site specific nomenclature in which case all references to class will show using the mapped nomenclature instead.

The class determines a number of attributes which affect the behaviour and permissions on items of that class: For parts it can determine:

- the life-cycle for parts (if there is one)
- for component parts the template to be used for the initial version
- whether a CR is required to authorise changes to component parts
- whether component parts are to be kept checked out read only
- to restrict the operations which may be performed

The class also can be used for searching and reporting purposes and in conditions to other commands so that, for example, only parts of a certain class are included in a baseline, see [Creating and Managing Baselines](#) for information on baselines.

The class of a part would normally be set when adding the part to the database (or checking in the file for the first time). Thereafter the class should only be altered with care, particularly if the class defines a life-cycle, see [Part Life-cycles](#).

If it is necessary to change the class this may be achieved using **Part | Alter | Alter Class**.

Part Flags

The flags of a part indicate special conditions for the part. Flags are only applicable to components and not subsystems.

Flags may be:

NoFile

This indicates that the part has no physical representation on the computer. It might be used for items such as hardware, in order to track its versions and configurations. The location field will not be used by the system.

NoVC

This indicates that the part is not to be maintained under version control. This might be used for example for compiler header files or libraries if you wish their existence to be known but not have them under version control. This flag is only applicable to component type parts. Command actions will be issued for this component, but there will be no versions.

Flags may be set on the creation of a part or by altering the field (using **Part | Alter | Alter Flags** or **Part | Version | Alter Flags** as appropriate). Care should be taken if a flag is added after creation of the part: for example if a part is created and several versions of that part are created, the **NoVC** flag should not be set since there are already some versions in existence.

Part Arbitrary Information

Each part has fields which may be used to hold arbitrary, site-specific information about the part. Each component/ subsystem has 40 arbitrary fields of its own. They are displayed on one or more tabs on the viewer as defined for your site. Each site should have a policy as to whether these fields are to be used at all and, if so, what information they should contain. If these fields are in use the **AllChange** Administrator should have assigned them a name indicative of their function and where appropriate specified the valid values for each field.

An arbitrary field on the component could be used, for example, to hold a description of the component.

Obsolete Parts

If the entire part is *obsolete* then this will be shown at the top right of the part view window as **OBSOLETE**. This indicates that the part and *all* its versions are obsolete and no longer used. This flag may be used if a part no longer exists in a current release, but needs to be kept for historical purposes and to enable a back-out to a previous release which did include this part.

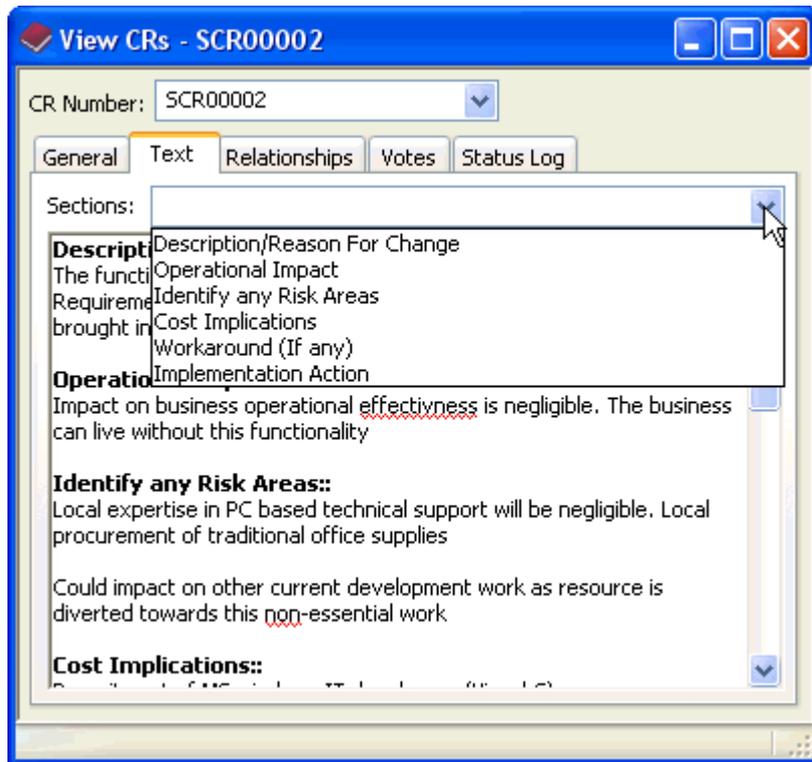
This obsolete flag may be changed using **Part | Alter | Make Obsolete**. The obsolete flag may be used to restrict the part browser, reports etc. to include only those parts which are currently active.

Obsolete parts may also be associated with CRs to indicate that making the part obsolete is a part of the change required to implement the CR, this in turn can be useful for populating baselines, see [Updating a Baseline from CRs](#) for details. 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 automatically associated with the CR. If the CR Item Affected relationship has an arbitrary field defined called Obsolete, then this is set to the value Yes.

By default obsolete items are not shown in browsers, this may be changed using **View | Show Obsolete Items**.

The Text Tab

The text tab shows the text associated with the CR, Part or Baseline.



No constraints are imposed on its format or contents by **AllChange**. However any line matching a site defined pattern as a section start will be listed in the **Sections**. See the **AllChange** Administrator Manual for details on defining the section pattern. The default pattern is lines ending in : : are treated as sections. Section headings will be displayed in bold. Selecting a section will scroll the text to the start of that section.

Hyperlinks in the text will be shown as clickable links. Hyperlinks are activated for common protocols such as "http", "ftp", "mailto", etc, and also for the "allchange" protocol, and any other registered **All-Change** protocols. If the text is read-only, then the link is followed by single-clicking the link. If editable, then the Ctrl key needs to be held down while clicking to follow the link. Right-clicking on links will show a menu where the link may be opened or copied to the clipboard.

When items are created they have an initial text "template": it is up to the individual site to supply suitable empty template forms for their items and to ensure that users complete these in an acceptable fashion. Different classes of item may have different initial text templates.

Classes of items may be defined as having *protected text*. This means that the text for the item may be entered as normal at creation time, but thereafter may only be appended to within each section together with an automatic stamp of the date/time of the addition and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the CR, Part or Baseline menu. If the text is *protected* the background colour of the text tab will be by default *greyed out*, so show that text may not be typed in directly. The background colour used for this may be changed according to user preferences, see [Customising the User Interface](#) for details.

Text spelling is checked in the text tab. Any misspelled words are underlined with a red wavy line. See [Modifying Text](#) for more information on spelling options etc.

Searching in Text

Text may be searched for a specific word or string. Also text may be replaced using a Find/Replace facility.

To invoke Find, right-click the text field and select Find, or use the shortcut keys Ctrl+F or Ctrl+D. This will open a dialog box where the search string can be entered. Options may also be specified here to match the case, or match a whole word. Once a word has been searched for, pressing F3 will repeat the search starting from the current cursor position. Pressing F3 without having specified a search string will open the dialog as shown on pressing Ctrl+F/Ctrl+D.

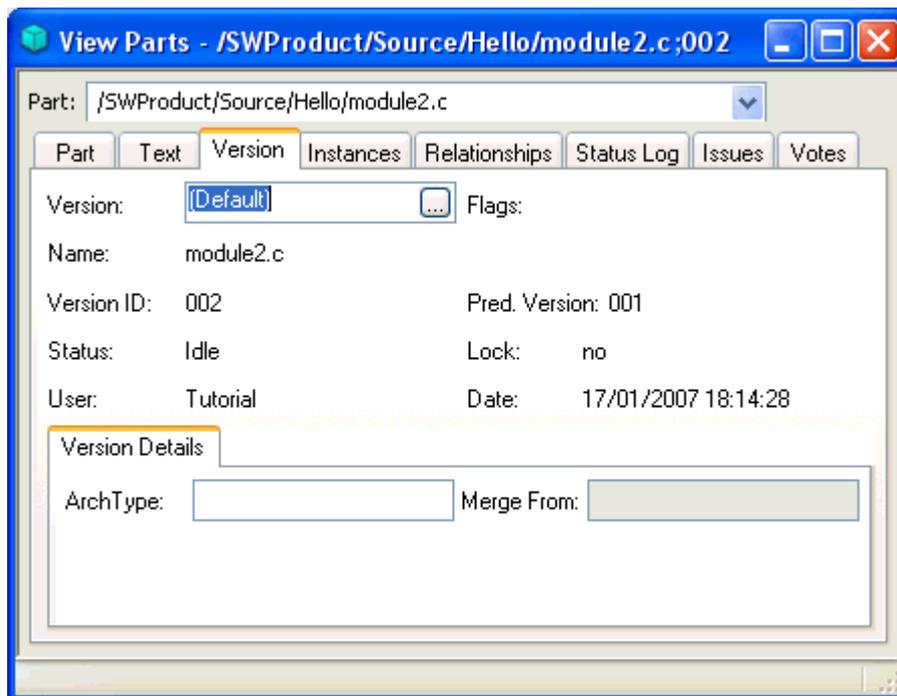
To use Find/Replace, press Ctrl+H, or right-click the text and select Replace. A dialog will open allowing the search string to be specified, along with the replacement string. Options may also be specified here, as per Find. If the text is protected against editing, then the Replace functionality is disabled.

The Version Tab

The Version Tab

The Version tab shows fields of a specific version of a component. This is not present if the part selected is a subsystem.

Note that the term version may have been mapped to site specific nomenclature in which case all references to version will show using the mapped nomenclature instead.



The fields are:

OBSOLETE:

This indicates that a specific version is obsolete.

Version:

Specifies which version is shown. Selecting the **Version** button allows the version to be selected. A dialog will be shown containing all the versions of the component, plus some special values (**Default**, **Registered** and **Top**). Select the required version and its details will be shown and that version will be the *current* version for the part viewer.

Name:

The name of the component whose version is shown.

Version ID

The version identifier for the version currently shown as selected by **Version**

Status:

The current status of the version. Use the **Status** button to change the current status.

Lock:

Part is locked against change (Yes or No).

User:

The user who created the version.

Date:

The date when the version was created. If the version is currently checked out for edit, then this is the date that the version was checked out. Otherwise the date is the date that the version was checked in unless the Putaway store file time configuration option is set in which case it is the date stamp that the file had at the time it was checked in.

Flags:

Special characteristics of this version.

Pred. Ver:

The version-id of the version's predecessor.

Varb1... Varb40:

Arbitrary, site-specific information about the version. These field names may be configured as required for your site.

Obsolete Versions

If a specific version of a part is *obsolete* then the obsolete flag may be set for the version.

Obsolete parts may also be associated with CRs to indicate that making the part obsolete is a part of the change required to implement the CR, this in turn can be useful for populating baselines, see [Updating a Baseline from CRs](#) for details. The out-of-the-box configuration requires a CR to be specified when making a controlled part obsolete. The part and version are then automatically associated with the CR.

This obsolete flag may be changed using **Part | Version | Make Obsolete**. The obsolete flag may be used to restrict the part browser, reports etc. to include only those versions which are not obsolete.

By default obsolete items are not shown in browsers, this may be changed using **View | Show Obsolete Items**.

Version Flags

The flags of a version indicate special conditions for the version.

Flags may be:

NoVersion

This indicates that this part is a reserved version and not an actual version. It is only applicable to component version parts and is not user-settable. This flag is set when a part is checked out for the purposes of making a change. If the version was created as a result of optimistic locking then this will not have a fixed version ID and even its predecessor may change. If the version was created as a result of pessimistic locking then the version will be the final version ID that will be used when the part is checked back in

DuplicateVersion

This indicates that this part is a duplicate version of another version. It is only applicable to component version parts and is not user-settable. This flag may be set when new versions are created by a life-cycle status change.

Version Arbitrary Information

Each version has fields which may be used to hold arbitrary, site-specific information about the version. They are displayed on one or more tabs on the viewer as defined for your site. Each site should have a policy as to whether these fields are to be used at all and, if so, what information they should contain. If these fields are in use the **AllChange** Administrator should have assigned them a name indicative of their function and where appropriate specified the valid values for each field.

A version arbitrary field could be used to hold a version specific comment.

Part and Version Status

The current status of a subsystem, component or version is given by the status field. Each version of a component has its own status field.

Note that the term status may have been mapped to site specific nomenclature in which case all references to status will show using the mapped nomenclature instead.

If no life-cycle is in effect for a part or if the life-cycle has not been started (e.g. by setting the status to the initial status) then it will be blank.

Life-cycles and changing the current status are described in [Part Life-cycles](#).

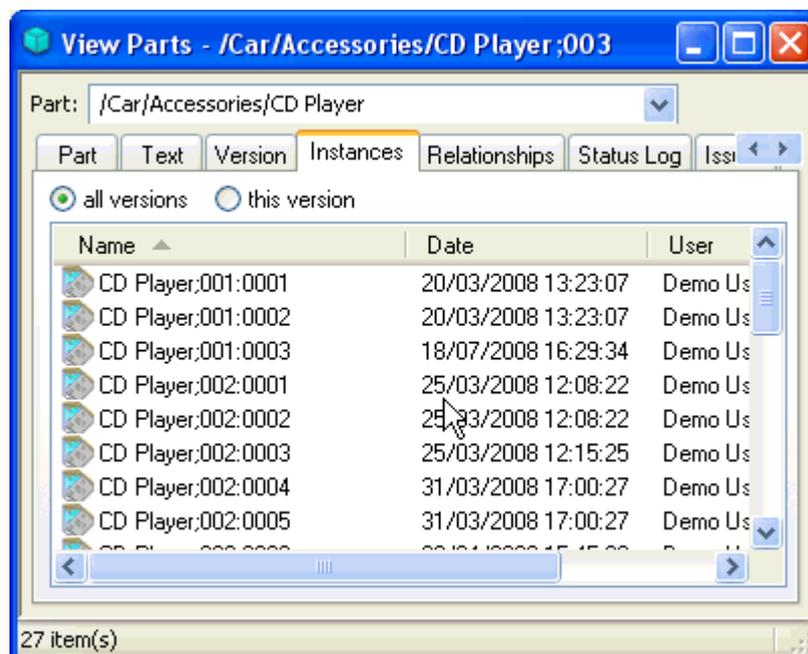
Lock

The locked field of a part or version indicates that it is locked against further change. Each version of a component has its own locked field. This field is set by taking a baseline and effectively indicates that this part is contained in at least one locking baseline. See [Creating and Managing Baselines](#) for further details.

The Instances Tab

The Instances tab shows the instances of versions of a component. This is not present if instances are disabled. If the part shown is a subsystem then this is not available,

Note that the term instance may have been mapped to site specific nomenclature in which case all references to instance will show using the mapped nomenclature instead.



all versions: select this to show the instances for all versions of the current component

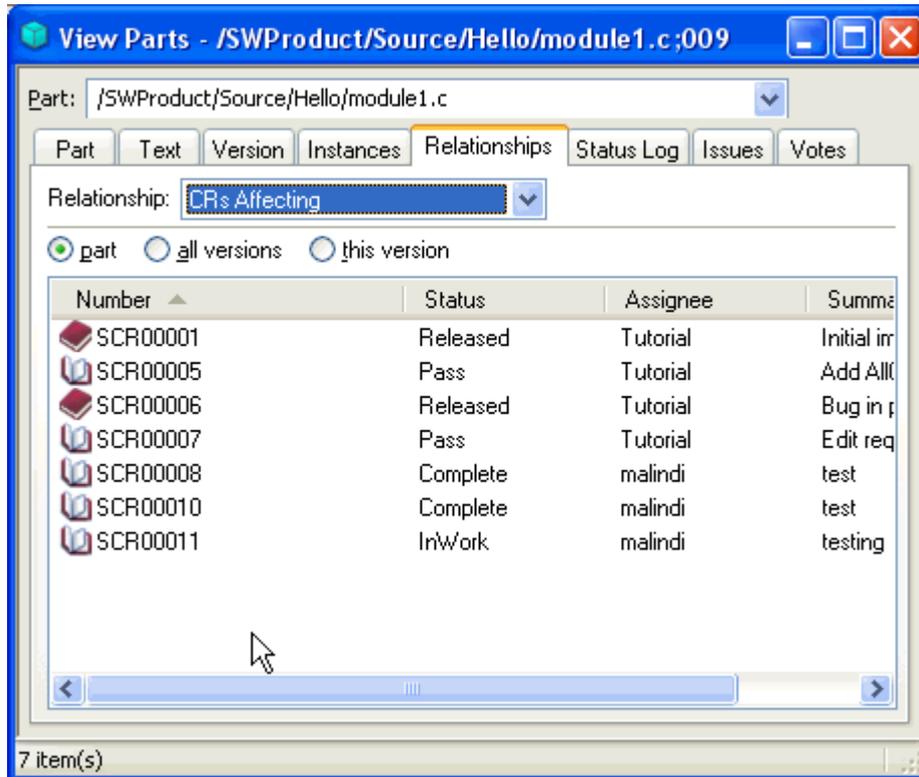
this version: select this to show the instances for the current version of the current component

Double clicking on an instance in the list will open the Instance Viewer for the instance selected.

The Relationships Tab

The relationships tab shows the relationships of the selected part to other items.

Note that the terms CR, part and baseline may have been mapped to site specific nomenclature (e.g. RFC, CI, Release) in which case all references to that term will show using the mapped nomenclature instead.



The **Relationship** to be viewed can be selected from the drop down list as follows:

CRs Affecting

Shows a list of the CRs which are *affected* by the part. It shows the CRs which have the part selected as a *partaffected*.

CRs Solved

Shows a list of the CRs which are *solved* by the part. It shows the CRs that have a version of the part selected as a *versionsolved*.

Baselines

Shows the baselines that contain the part selected.

Parts Affected

Shows the parts that the part selected affects, i.e. point to the selected part.

Parts Referring

Shows the parts that affect the selected part, i.e. refer to the selected part.

As appropriate for the relationship shown radio buttons may be available to filter the list of items shown:

- **part:** select this to show the relationships to the subsystem or component selected
- **all versions:** select this to show the relationships to any version of the selected component
- **this version:** select this to show the relationships to the current version of the selected component

As appropriate for the relationship shown there may be actions performed to add or remove items from the list. These actions will be available from right click context menu, or from buttons available when the

show/hide action buttons button to the side of the list is selected. When adding a new relationship the radio button selection of **part** or **this version** determines whether the relationship is to the component or the version currently viewed.

Double clicking on an item in a list will open the viewer for the item selected.

Additional details about the relationship may be view using **View database Relationship** menu item on the right click context menu, where *database* is CR, part or baseline as appropriate. This is also available from the appropriate main menu, **CR, Part or Baseline | View database Relationship**

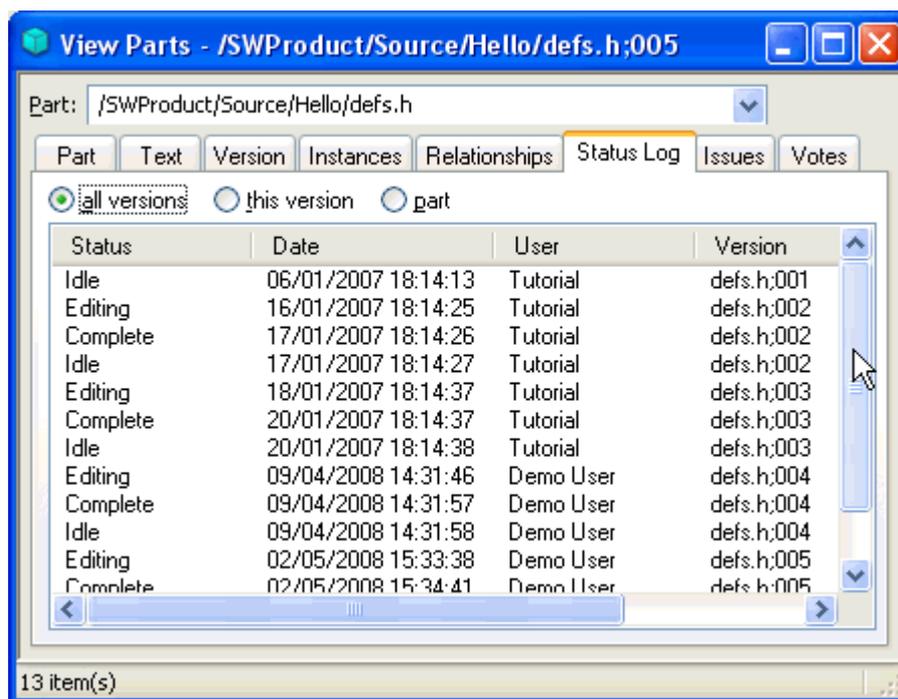
All relationships for each of CRs, Parts and Baselines may also be browsed using the appropriate relationship browser available from **CR, Part or Baseline | Browse database Relationships**

The Status Log Tab

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on, or viewed in the **Status Log** tab.

Note that the term status log may have been mapped to site specific nomenclature (e.g. Audit Trail) in which case all references to status log will show using the mapped nomenclature instead.

The status log therefore provides a historical record/ audit trail of the progress undergone by an item.



all versions: select this to show the complete status log of all versions of the current component

this version: select this to show the status log for the current version of the component only

part: select this to show the status log of the part (no versions)

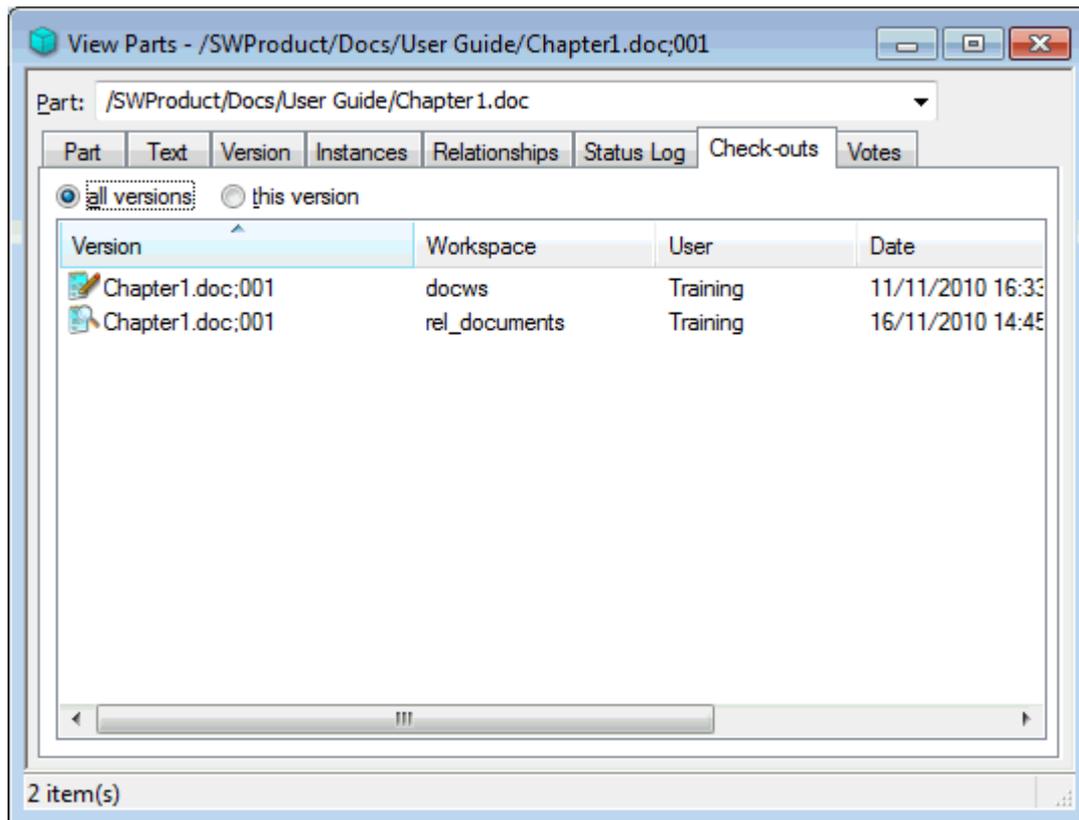
The status log will show the statuses that the part has been through, showing the status, the date and time the status was entered and the user who entered the status.

The status log has one field which may be used to hold arbitrary, site-specific information about the status log. This may be shown if it is used.

The Check-outsTab

The Check-outs tab shows the workspaces to which the current component is checked out.

Note that the term check-outs may have been mapped to site specific nomenclature (e.g.issues) in which case all references to check-outs will show using the mapped nomenclature instead.



all versions: select this to show the workspaces which have any version of the current component checked out to it

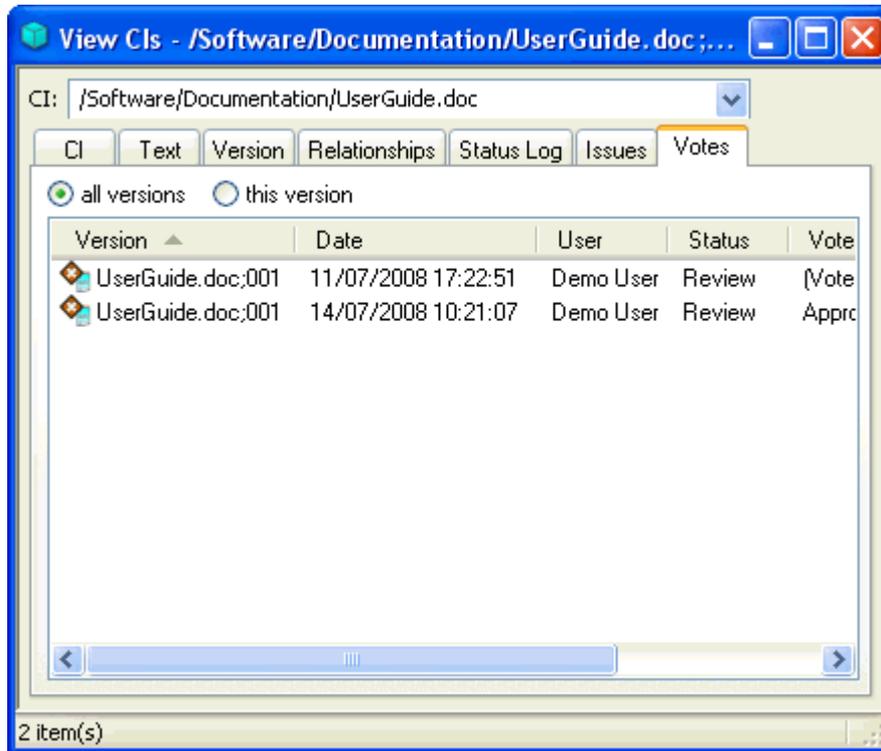
this version: select this to show the workspaces which have the current version of the current component checked-out to it

Double clicking on a part in the list will open the Check-out Viewer for the version selected.

The Votes Tab

The Votes tab shows a list of all the votes which have been cast for the part.

Note that the term vote may have been mapped to site specific nomenclature (e.g. approval) in which case all references to votes will show using the mapped nomenclature instead.



all versions shows all votes for any version of the part viewed.

this version shows all votes for the version of the part viewed.

The details of a specific vote may be seen in the Vote Viewer by double clicking on the vote or from the right click context menu.

Votes may be cast when the part is in a vote status by use of the cycle viewer or **Vote | Cast Vote**

Browsing and Viewing Part Relationships

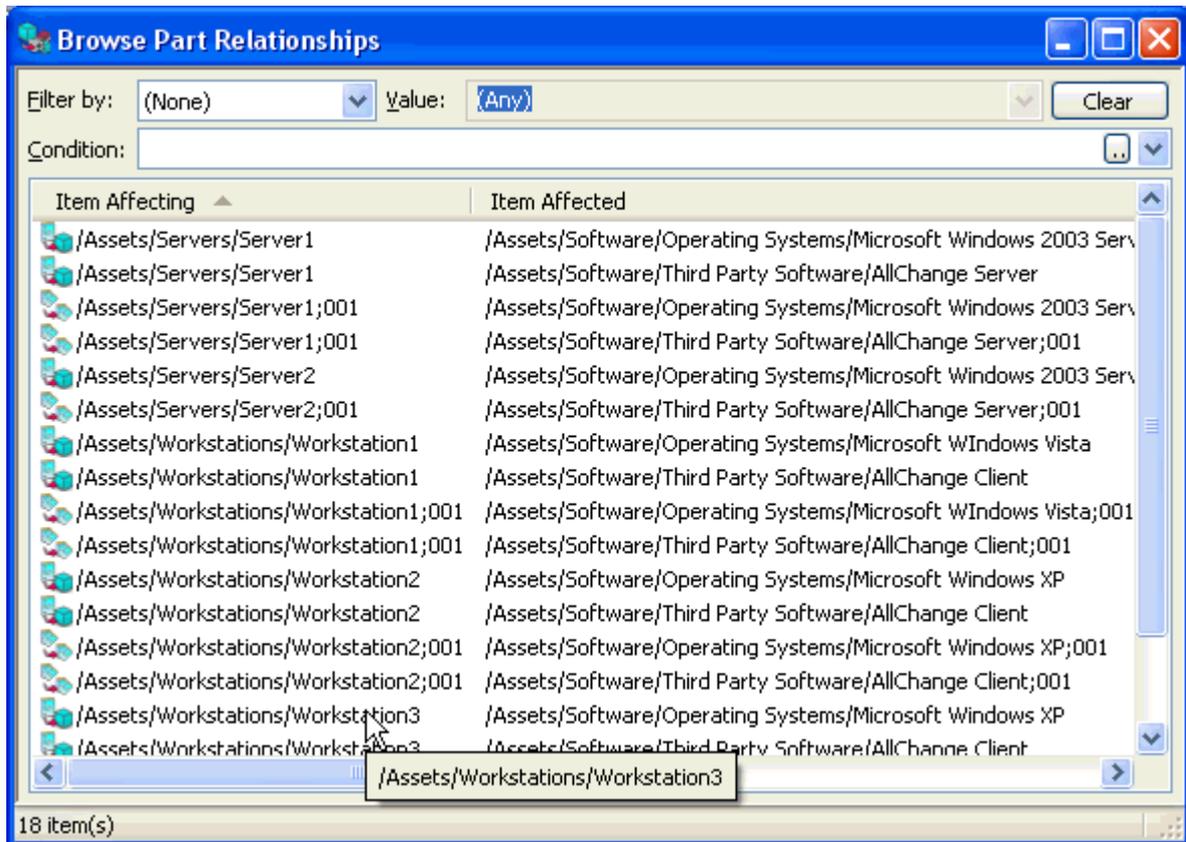
The non-hierarchical relationships between parts are known as parts affected. Relationships may be created between subsystems, components and versions to subsystems, components and versions. These relationships can be examined using:

- [Part Relationships Browser](#)
- [Part Relationship Viewer](#)
- [Relationships Tab](#) of Part Viewer

These relationships may be created and deleted from the Relationships Tab of the Part Viewer and may also be deleted from the Part Items Affected Browser.

Part Relationships Browser

The Part Relationships Browser allows you to browse the non-hierarchical relationships between parts, it is accessible from the Parts Menu



The precise information shown for each part item affected is user definable (see [Customising the User Interface](#)).

The **Filter By** may be used to restrict what is displayed according to the **Value**.

The **Condition** may be used to further limit the items affected shown to any arbitrary criteria.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.
- The information may be sorted by any column
- [Folding View](#) showing the relationship relationships for each part

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

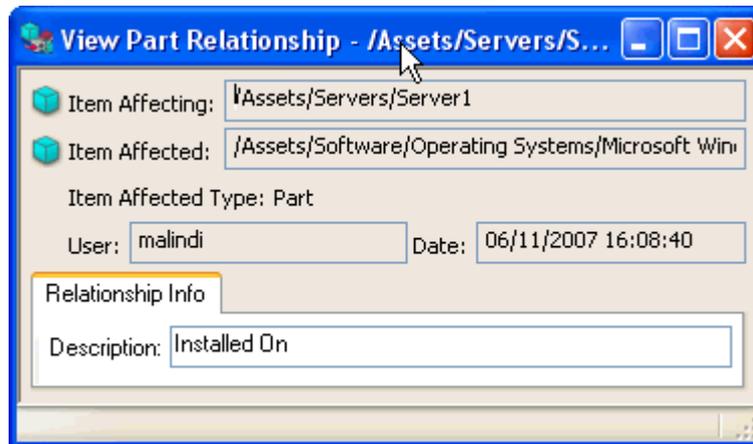
Double clicking on an item in the browser will show the details in the Part Item Affected viewer window.

Part Relationships Viewer

Information about the non hierarchical part relationships to other parts in the system is stored in the part relationships (also known as items affected) database. Part relationship entries have a number of fields associated with them.

To view the details of a particular part relationship use the part relationship viewer; this may be accessed by **Part | View Part Relationship**, or double clicking (or using the right click context menu) on a part relationship entry in the partrelationship browser or right click in relationships tab of the part viewer.

The Part Item Affected Viewer shows the detailed information about a part relationship to another part



The fields are:

Item Affecting:

The part which holds the relationship. This is the part affecting

Item Affected:

The part which is related to the **Item Affecting**, this is the part affected. i.e. the **Item Affecting** points to the **Item Affected**, or the part affecting points to the part affected.

User:

The user who created this relationship. *Note this may be blank for data created prior to version 7.2 of AllChange*

Date:

The date when this relationship was created. *Note this may be blank for data created prior to version 7.2 of AllChange*

Arb1... Arb40:

Arbitrary, site-specific information about the item affected relationship. The field names may be configured as required for your site.

Browsing and Viewing Instances

Component versions may have instances associated with them, these can be used to represent specific occurrences of a version. This can be useful, for example, to represent the manufacture of hardware items or the distribution of software items, see [Instances](#).

Note that the term instance may have been mapped to site specific nomenclature in which case all references to instance will show using the mapped nomenclature instead.

Instances may be examined using:

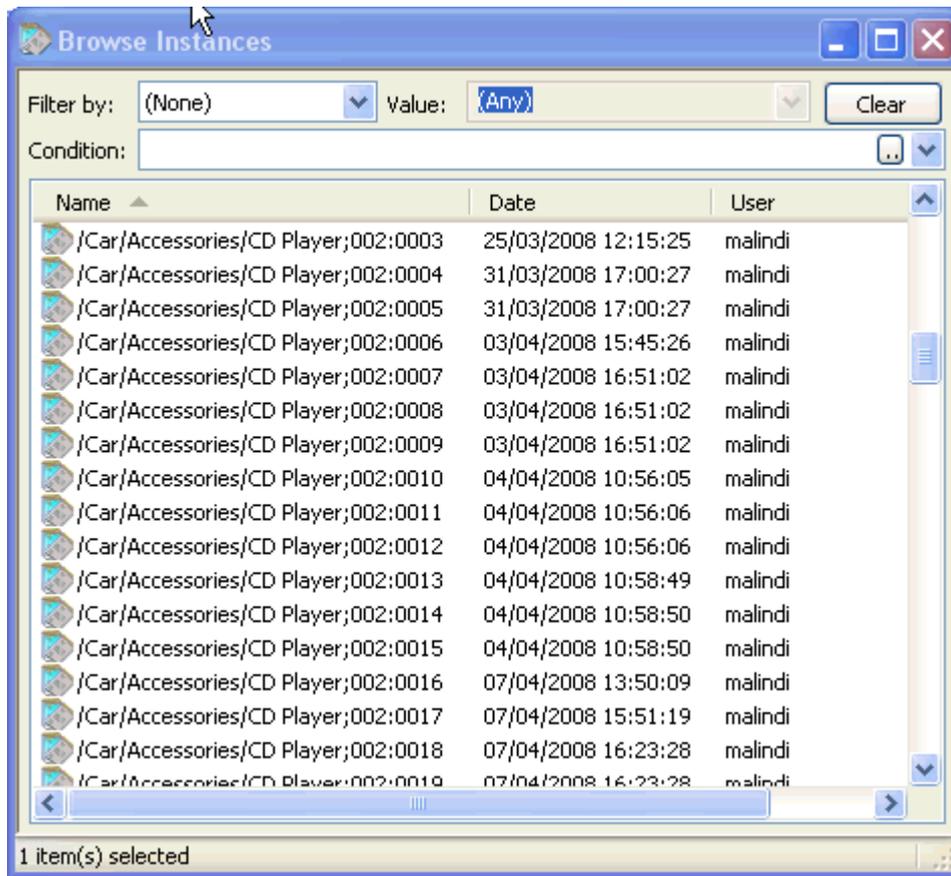
- [Instances tab](#) on Part Viewer
- [Instance Browser](#)
- [Instance Viewer](#)

Instances may be added using **Part | Instance | New Instance** and may be deleted from the Instance Browser or Instances tab on the Part Viewer. Instances may be modified from the Instance Viewer.

Note that instances may be disabled in which case no operations relating to instances will be present.

Instance Browser

The Instance Browser allows you browse the instances of versions of component parts. It is accessible from **Part | Instance | Browse**.



The precise information shown for each instance is user definable (see [Customising the User Interface](#)).

The **Filter By** may be used to restrict what is displayed according to the **Value**.

The **Condition** may be used to further limit the instances shown to any arbitrary criteria.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.
- The information may be sorted by any column
- [Folding View](#) showing the instances for each version

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

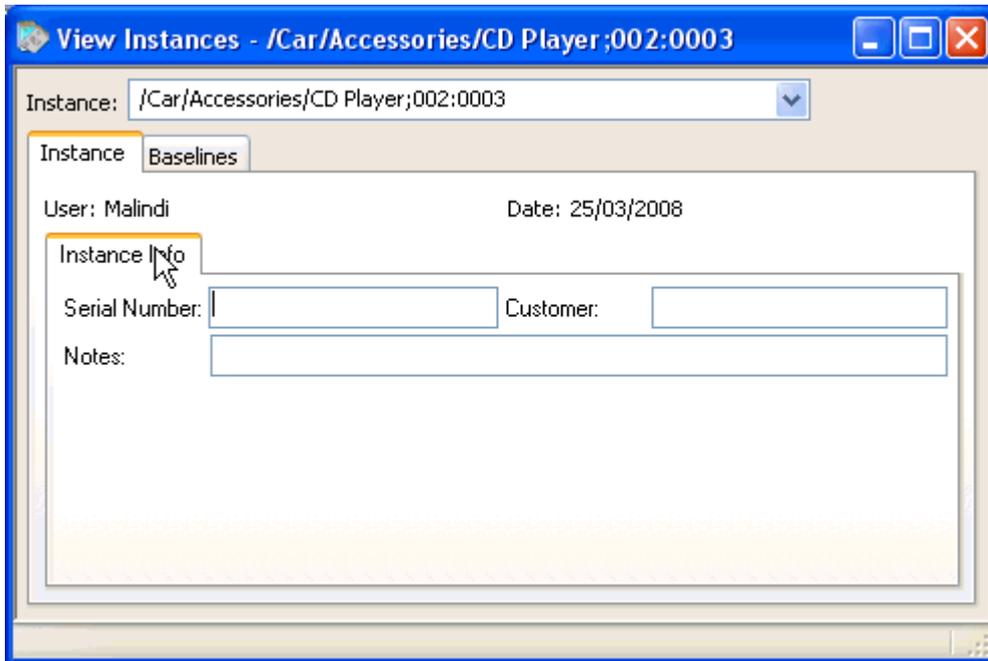
Double clicking on an item in the browser will show the details in the Instances Viewer window.

Instance Viewer

Information about the instances in the system is stored in the instances database. Instances have a number of fields associated with them.

To view the details of a particular instance use the instance viewer; this may be accessed by **Part | Instance | View**, or double clicking (or using the right click context menu) on an instance in the instance browser or instances tab of the part viewer.

The Instance Viewer shows the detailed information about the instance.



The fields are:

User:

The user who created the instance.

Date:

The date and time that the instance was created.

Arb1... Arb40:

Arbitrary, site-specific information about the instance. The field names may be configured as required for your site.

Baselines:

The instance baselines in which the instance is included.

Creating a New Part

About Creating a New Part

New configuration items may be defined to **AllChange** by creating new *parts*. There are several different situations where new parts need to be created:

1. New parts are to be created explicitly in order to create the configuration item for the first time, see [Creating New Parts Explicitly](#)
2. New parts are to be created for existing files (directories/ files) which are to be place under **All-Change** control, see [Importing Existing Files](#)
3. New parts are to be created for existing configuration item data which are to be place under **All-Change** control, see [Importing Parts from CSV File](#)
4. New parts are to be created for existing files currently stored in subversion, see [Importing Parts from Subversion](#)

Creating New Parts Explicitly

About Creating New Parts Explicitly

In order to create a new subsystem or component type part explicitly use the **Add Part** dialog from the **Part | Add** menu, or from the toolbar when either the part browser, part viewer or file browser are the current window.

Field which are compulsory will be shown in red and a value must be entered. The Parts field specifies the part(s) to be created. If 'Allow Multiple Parts' is not ticked, then one part is created even if spaces are entered. If it is ticked, then multiple parts names may be specified, separated by spaces. In this case, any part names containing spaces must be quoted.

Subsystems

In order to create a subsystem the name(s) of the new subsystem(s) should be entered as the **Parts** in the **Add Part** dialog. If a plain name is entered e.g. `fred` then a new part called `fred` will be added as a child of the current working part shown in the status bar.

Select the **subsystem** option for the **Type** of the part.

The classes listed in the **Class** combo box will list the classes valid for subsystems or **None** if no class is required.

Values may be entered for the arbitrary fields as appropriate to your site. The **AllChange** Administrator should have configured the system to show meaningful names for the arbitrary fields used and for the combo boxes to contain list of valid values where appropriate.

Components

In order to create a component specify the name(s) of the new component(s) in the **Parts** of the **Add Part** dialog. If a plain name is entered e.g. `fred` then a new part called `fred` will be added as a child of the current working part (the current location in the part browser).

Select **component** as the **Type** of the part(s).

The **Class** combo box will list the classes valid for a component or **None** if no class is required.

When the component is created (with a first version) the content of the first version will be taken according to the following:

- If a workfile exists for the component in the current workspace then this is used
- If a template file has been defined for the class specified then this is used (note that you must be attached to a workspace valid for the component)
- An empty first version is created

Values may be entered for the arbitrary fields as appropriate to your site. The **AllChange** Administrator should have configured the system to show meaningful names for the arbitrary fields used and for the combo boxes to contain list of valid values where appropriate.

The **Flags** are optional and should be used with care:

Not Under Version Control

This should be used when the components are not to be stored under version control, i.e. no version history is to be maintained.

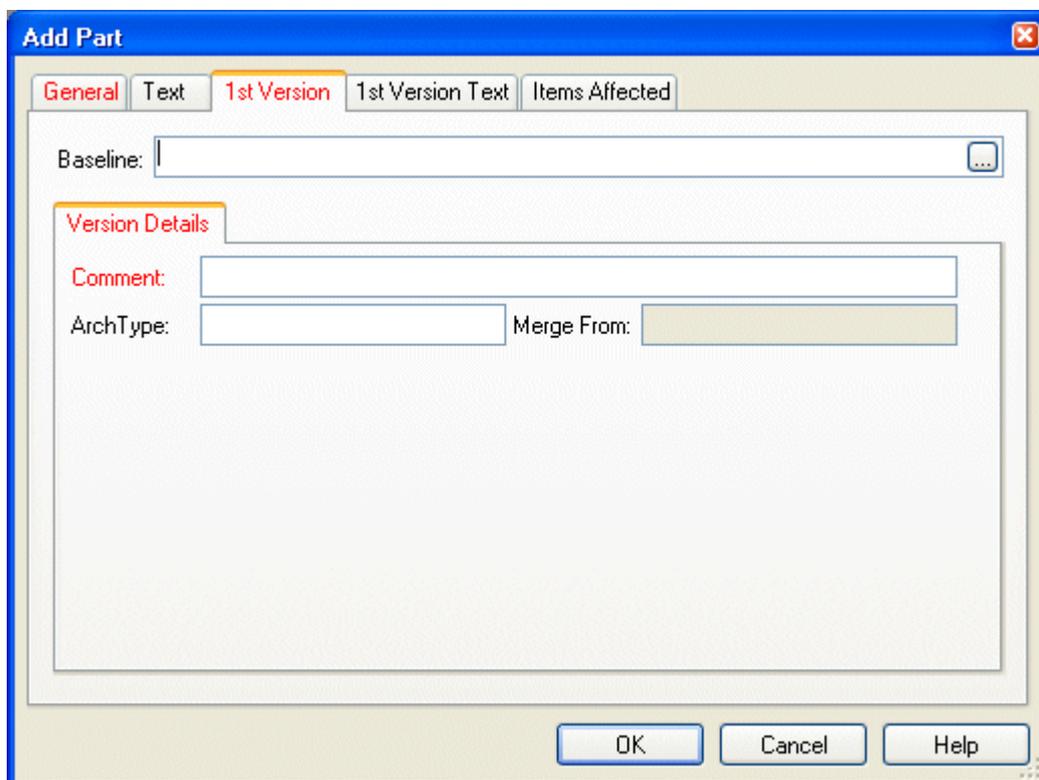
No File

This should be used to indicate that there is no on-line file to correspond to these components. This may be used for example, for hardware or paper documents, in order to track their existence and history without physically storing the contents.

The **No Initial Version** option if selected will cause the component to be created but no corresponding initial version or file. This may be useful during the design phase of a project to create the component, but there is not as yet a document or file that corresponds to it.

Unless the **No Initial Version** option is selected, a first version of the component will be created. If there is a workfile for the component in the current directory, then this will be taken as the contents of the initial version, otherwise if a template file for the class has been specified then this will be used (note that you have to be attached to a workspace) otherwise an empty first version will be created.

If an initial version of a component is to be created then a **1st Version** tab will appear on the **Add Part** dialog which may be used to specify information to be associated with the first version of the component (as distinct from the information associated with the component itself which is applicable to *all* versions of that component).

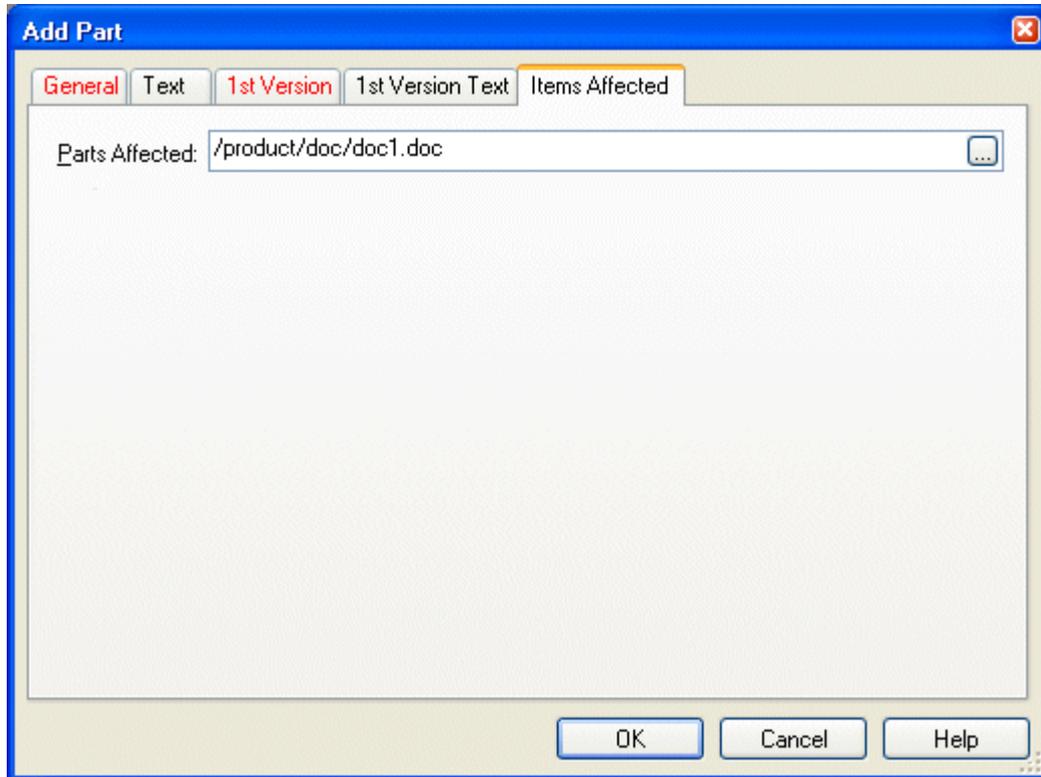


A *release Baseline* may be specified to which version 1 of the new part will be added. This may be used to ensure that baselines are kept up to date with the latest components added. Also any arbitrary fields in use may be specified for the first version.

If the **No File** flag has not been specified then an initial file version will be sought first in the work directory for the currently attached workspace and then if a template has been specified for the class, this will be used as the initial file version.

Items Affected

For both subsystems and components the new part may be defined as *affecting* other parts using the **Items Affected** tab

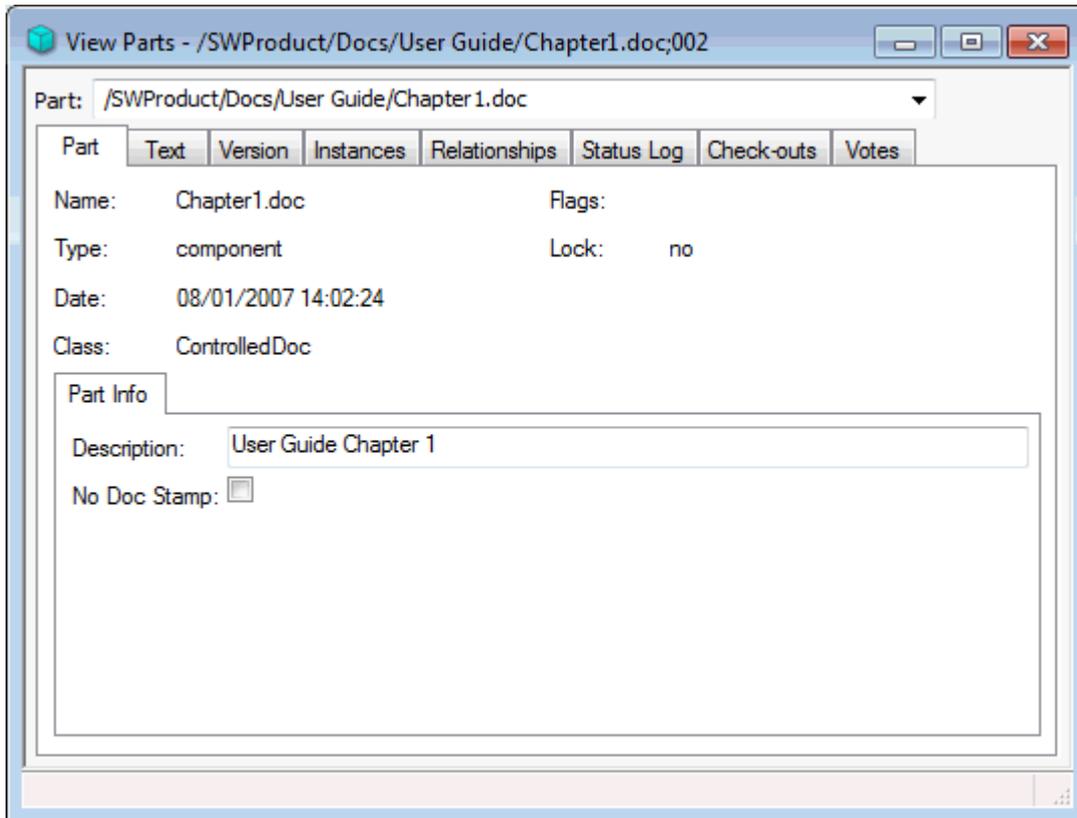


Specify the parts that are to be referred to by the new part, i.e. the parts that the new part affected (or which are affected by the new part)

Updating a Part

Updating a Part

Many of the fields of a part or version may be updated from the part viewer. Simply select the new value(s) required for any editable fields in the viewer and then select **Misc | Update**. Alternatively the **Update** button on the toolbar may be used.



Fields that are not alterable from the part viewer (e.g. **Name**, **Flags**) may be altered using special dialogs.

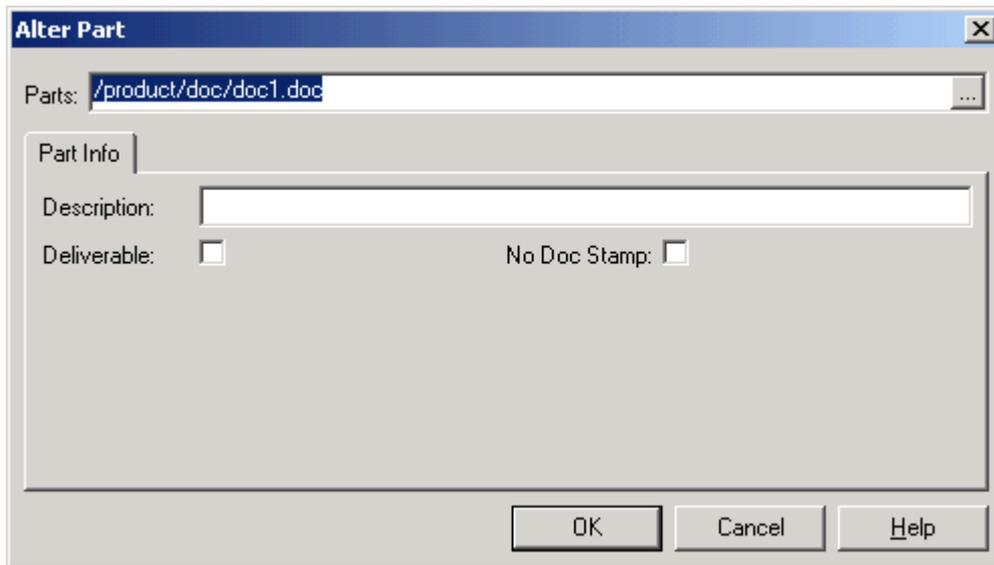
The **Status** may be changed using the **cycle viewer**, see [Part Life-cycles](#). The status may also be changed automatically by the **Check In** and **Check Out** commands.

The **Name** of a subsystem or component may be changed using **Part | Move**. This may also be used to move a subsystem or component to a different subsystem. See [Move Part](#).

Flags may be altered using the **Part | Alter | Alter Flags** and **Part | Version | Alter Flags** command dialogs as appropriate.

The CRs, Check-outs and Baselines associated with the part will be modified by other appropriate command dialogs related to that database or they may be automatically modified as a part of the life-cycle processes used.

An alternative method of updating a record which allows many parts to be changed at the same time, is to select the parts required in the part browser and then select **Part | Alter** and select the appropriate option for the information that is to be changed. For example, to change the value of an arbitrary field for several parts, select **Part | Alter | Alter** and enter the required values in the dialog.



If the Part is of a class which is defined as having *protected text* then the text on the Text tab may not be arbitrarily altered but may only be appended to within each section together with an automatic stamp of the date/time and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the Part menu.

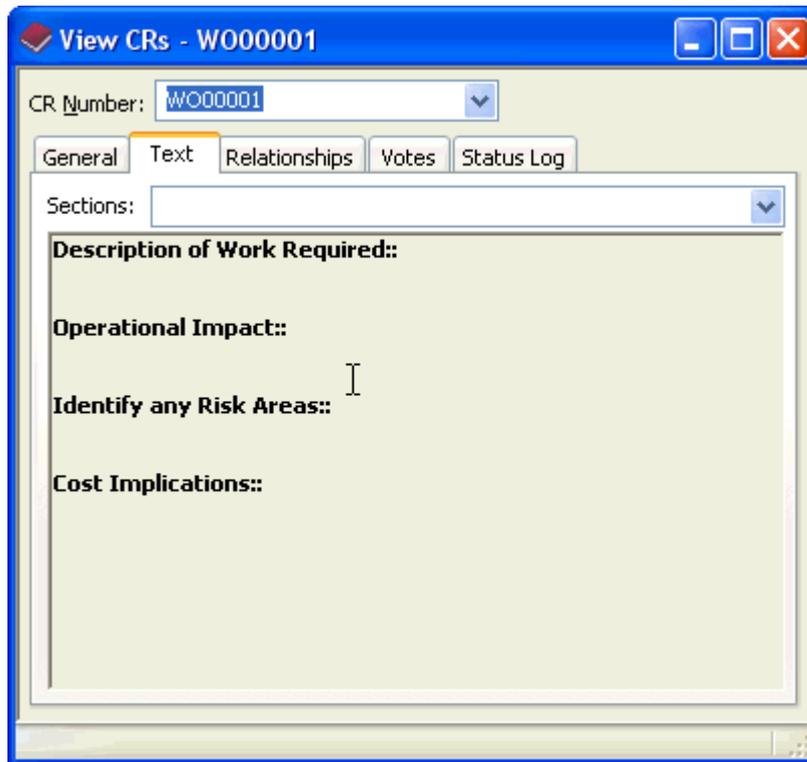
The [Insert into Text](#) facility may also be used with Parts which do not have protected text in order to obtain a date/time/author stamp to any modifications which may be made.

Text spelling is checked in the text tab. Any misspelled words are underlined with a red wavy line. See [Modifying Text](#) for more information on spelling options etc.

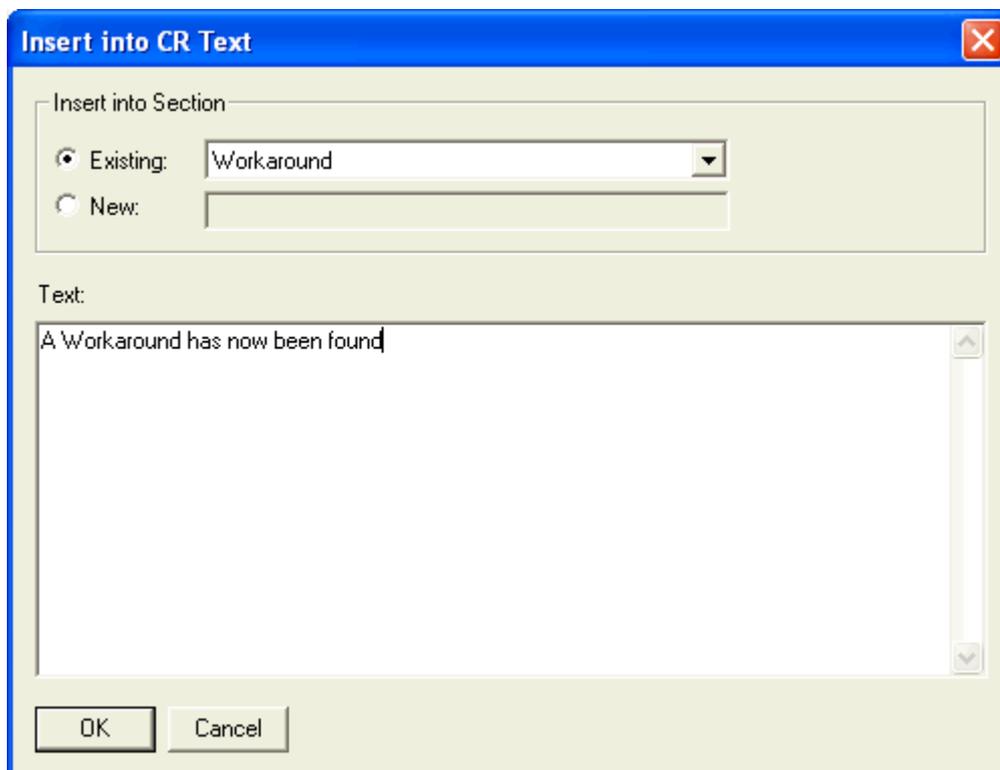
Modifying Text

If the item is of a class which is defined as having *protected text* then the text on the Text tab may not be arbitrarily altered (the edit control is shown read-only) but may only be appended to within each section together with an automatic stamp of the date/time and the user that made the addition. This ensures that text entered may not be modified and a full audit trail is automatically maintained.

[Misc | Options](#) allows the background colour of read-only text to be changed as required.



In order to append to a section, use the **Insert into Text** facility from the CR, Part or Baseline menu, having first selected an item in either a CR/Part/Baseline browser or viewer.



First you must select the section that you wish to insert into. An **Existing** Text Section may be selected from the drop down list or a **New** section may be created by selecting **New** and entering the title for the section in the edit control.

The text that is to be appended to the selected section may then be entered in the **Text** edit control.

The text entered will be appended to the section preceded by a stamp of the date and time and the user inserting the text.

Under certain circumstances it may be desirable to enable certain users to make modifications to protected text, for example to remove an undesirable entry. The users/roles permitted to do this may be defined as a part of the access control definition by the **AllChange** administrator, a user who has this permission will see the Text as editable as with classes which do not have the protect text attribute..

The **Insert into Text** facility may also be used with items which do not have protected text in order to obtain a date/time/author stamp to any modifications which may be made. Alternatively edits may be made directly in the Text tab edit control.

Spell-checking is available in both the Text tab, and the Insert into Text dialog. Note though, that if the text is protected, spell-checking is disabled in the viewer tab. See the [Spell Checking](#) topic for more details.

Importing Existing Files

About Importing Existing Files

There may often be times when you wish to add existing files/ documents to **AllChange** with the current contents as the first version of the new **AllChange** component.

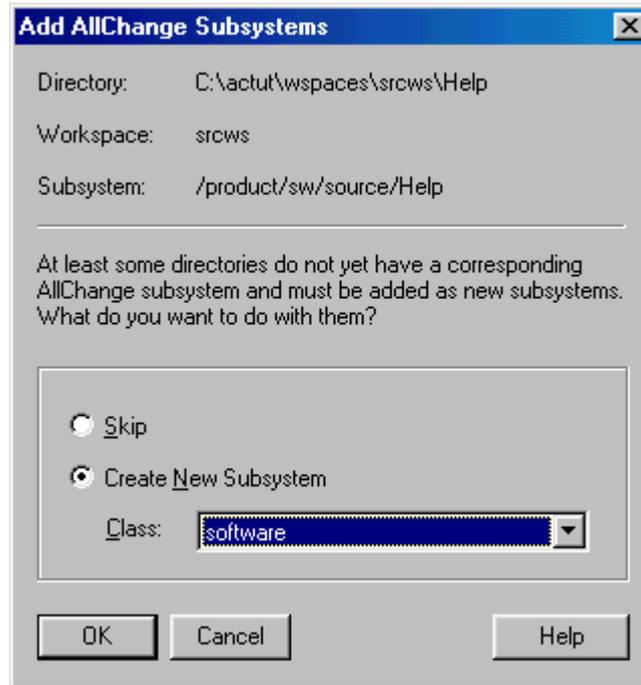
There are several different methods of achieving this depending on the environment in which you are working:

- Files may be added from the **AllChange | Check In** option on the menu available from Windows Explorer, MS Word and environments supporting the MCSCCI interface (e.g. Visual Studio, MS Access)
- Files may be imported from within the ACE interface using **File | Check In** (or dragging the files/ directories from the file browser and dropping them onto the part browser)

In all the above cases the same **Check In** function will be invoked and the behaviour will be the same in all cases.

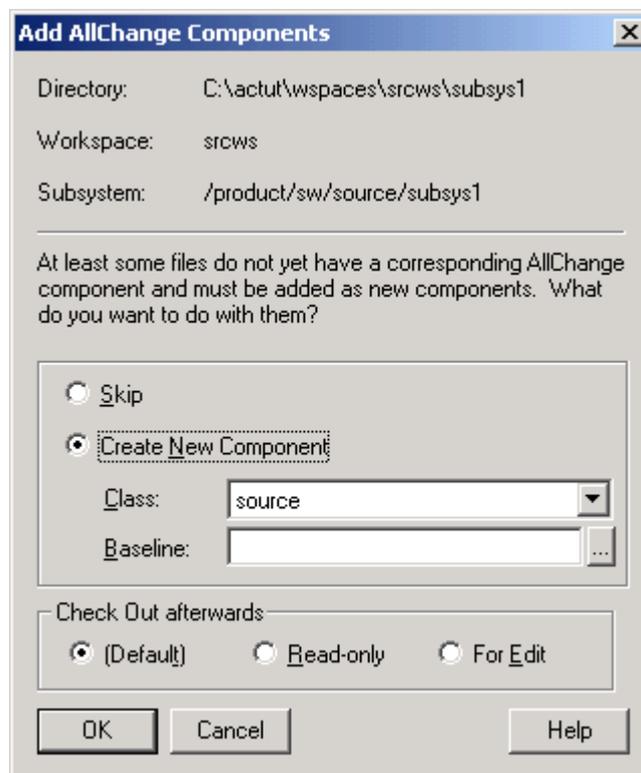
Whenever a **Check In** is performed on a selected file/ document or directory and it does not exist as a part then you will be given the option to create the necessary parts into **AllChange**.

First if any subsystems need to be created to correspond to the directory tree imported you will be offered to **Add** these to **AllChange**.



This allows you to **Skip** subsystem creation or **Create New Subsystem**. You will only be offered this once for all subsystems which do not exist. If **Create New Subsystem** is selected then the **Class** for any new subsystems created may be specified. If there are any compulsory arbitrary fields defined for the specified class then these will be prompted for in turn.

You will then be offered to **Add** any files which do not exist as new components.



This allows you to **Skip** component creation or **Create New Component**. You will only be offered this once for all components which do not exist. If **Create New Component** is selected then the **Class** for any new components created may be specified. Also a *release* **Baseline** may be specified to which the first version of the new component will be added. Note that the baseline must already exist.

If the class for the new components is defined as *requiring a CR* then a CR must be specified to authorise the addition of the new components. This will be prompted for.

If there are any compulsory arbitrary fields defined for the specified class then these will be prompted for in turn.

For full details of **Check In** see [Checking In](#)

Importing Files from ACE

Files which exist and are to be placed under **AllChange** control using ACE should be placed in a workspace directory and the corresponding workspace attached to (see [Workspaces](#)).

Select **File | Browser** and MS Windows Explorer will be displayed showing the contents of the current directory.

Select the files/ directories to be imported and then select **File | Check In**, alternatively select **Check In** on the context menu or drag the selected files to the part browser.

NOTE: files selected in the file browser for importing *must* reside within the directory hierarchy defined for the current workspace (shown in the status bar).

Check In will descend the directory hierarchy specified and allow you to add any directories which do not yet exist as parts (as subsystems) and any files which do not yet exist as parts (as components) as described above.

Importing files from Interfaces

To import files from Windows Explorer, MS Word or MCSCCI compliant environments first ensure that the files reside within a directory which is defined as a workspace to **AllChange**.

Select **AllChange | Check In** which will descend any directory trees selected or add files/ documents selected as components as described above.

Part Life-cycles

About Part Life-cycles

Life-cycles may be defined for use by parts and may be used to control access to items, implement approval procedures, log progress etc. A life-cycle is defined as a series of statuses through which a part passes. Each status has associated conditions and actions. The status conditions must be satisfied before a status change may take place, at which point the status actions will be invoked. A status may also have vote associated with it which should be completed before the status is progressed, see [Voting](#).

Note that the term status may have been mapped to site specific nomenclature, e.g. state, in which case all references to status will show using the mapped nomenclature instead.

The class of a part is used to determine its associated life-cycle — see [Part Class](#). This allows different classes of part to have different life-cycles if so desired.

Both subsystems and components may have their own life-cycles; versions inherit the life-cycle of their corresponding component. Subsystems, components and versions all have their own current status so, for example, different versions of a component may be on different statuses and move through the life-cycle independently — this is the usual case. The current status for component versions is displayed in the part browser (by default) and status of the current version is shown in the **Version** tab of the part viewer.

By default life-cycles on subsystems and components (but not versions) is disabled. If however this has been enabled the current status for a subsystem or a component is shown in the **Part** tab of the part viewer.

The conditions can be used to implement approval procedures. The actions can be used to trigger events: for example, the moving of files corresponding to the parts into and out of working areas, placing them in approval areas etc.

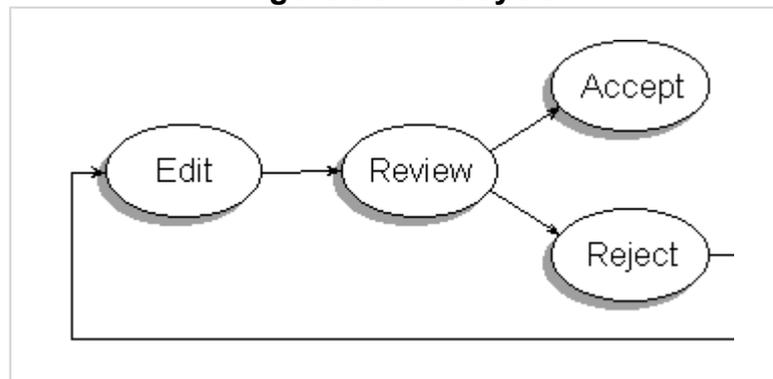
The statuses in a life-cycle may have **Check In** or **Check Out** attributes associated with them. Statuses with these attributes are used by the **Check In** and **Check Out** commands to progress the parts to the appropriate status on checking in or out, see [Checking Files In and Out of AllChange](#).

The status of a part may therefore be changed:

- *explicitly*
- *automatically by Check In and Check Out*

In [Figure 5.5](#) a possible document life-cycle is shown.

Figure 5.5: Life-cycle



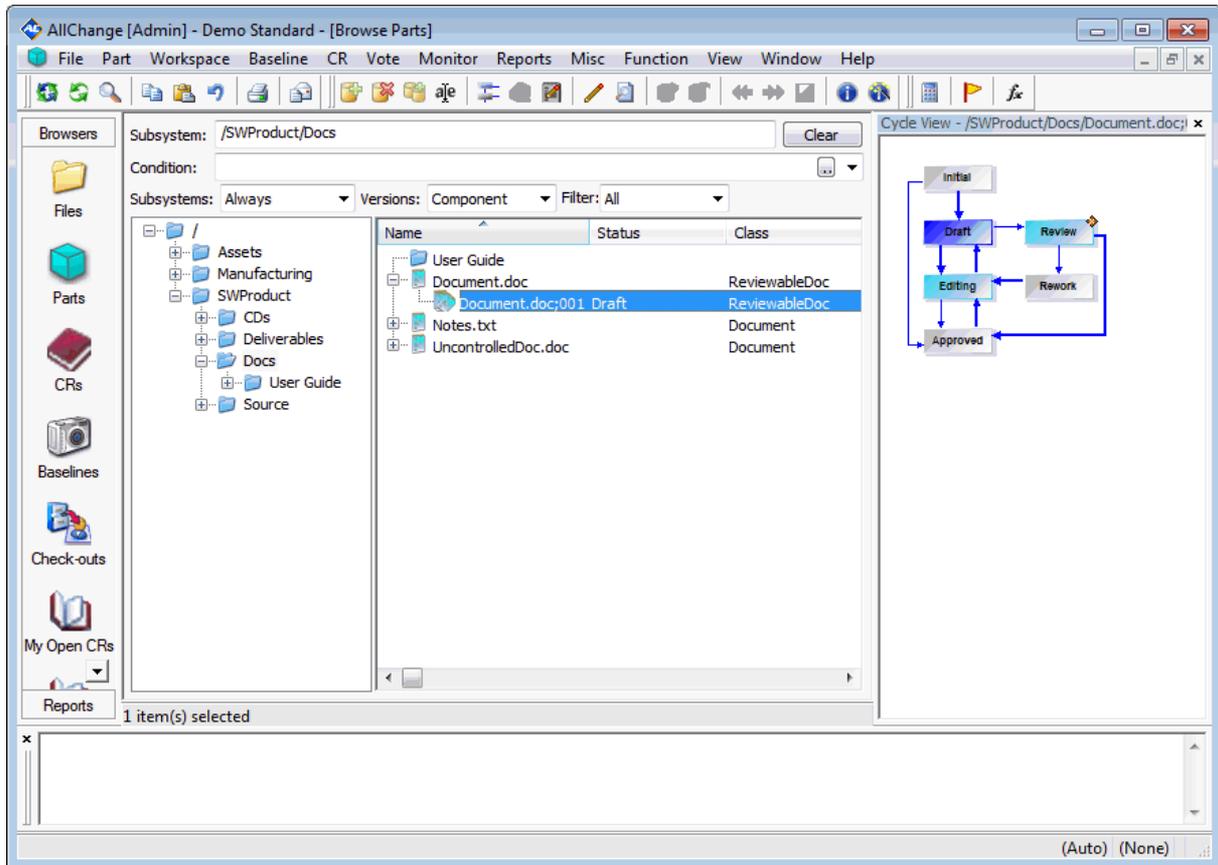
In this life-cycle the **Edit** status would be a **Check Out** status and the **Review** status would be the **Check In** status.

Changing Status

To change the status of a Part use the [cycle viewer](#). With a cycle displayed in the viewer double click on a status which is a valid progression and the status will be changed.

If the status is a **Check In** or **Check Out** status then these actions will be performed which in turn will change the status of the part.

Note that if there is an open vote on the status, then double clicking will open the **Cast Vote** dialog instead of changing the status. If the vote permits status progression (e.g. an advisory vote) then shift double click will allow the status to be changed and the vote will be closed). This is indicated on the tool tip on the status.



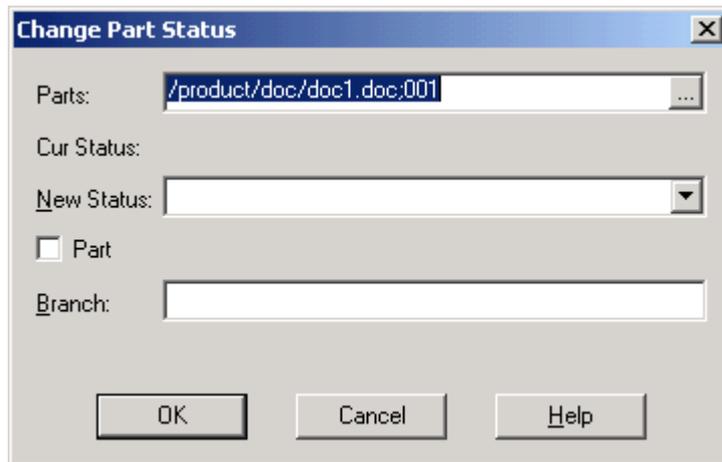
All status changes are logged in the status log database and may be viewed in the **Status Log** tab on the part viewer.

On selecting a status to change to the *current* status will be changed to the selected status and this status change will be logged in the part status log. Any conditions specific to the life-cycle for the requested status change *must* be complied with or an error will be generated. Any actions associated with the status change will be performed after the current status in the database has been updated.

The specific actions that occur on a status change will, therefore, depend on the site specific definition of the life-cycle being used. This will have been defined by your **AllChange** Administrator. In general, if part life-cycles are used at all they are used for most activities on parts.

If a component is selected as (i.e. no particular version), then the default version will be the one to which the status change is applied.

The **Status** dialog available from **Part | Status** can also be used to explicitly change the status of a Part, however this is not recommended for Check In/Out statuses.



Any parts already selected in a browser or viewer will be entered as the **Parts** which are to have their status changed. The **NewStatus** combo box will show a list of the valid progressions from the current state. They can also be viewed from the cycle-viewer.

Parts and a **NewStatus** must be selected for a change to be made.

The **part** option may be used to force the status change to be applied to a component itself rather than the default version of a component. This option is not normally used.

The **Branch** option should be used only if a new branch version is to be created, see Creating Branches

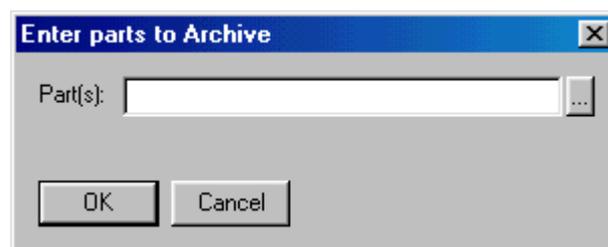
The **Other** option may be used as required for site specific approval procedures and will have its label changed according to the required input. For example, it may be used to specify the CRs against which a change is being made. It may even be used for different purposes depending on the status change being made. Your **AllChange** Administrator should have defined how this option is to be used.

Archiving Parts

The facilities to archive (i.e. remove from the current database and re-import at a later date) parts is intended for removing old/ obsolete parts from the database for efficiency.

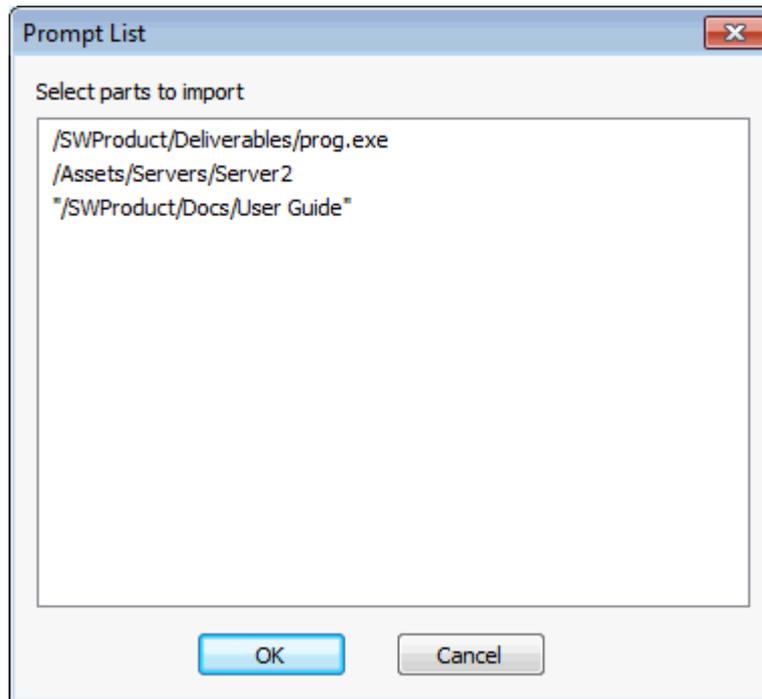
These facilities are available from the **Part | Archive Parts** menu, but are only available to **AllChange** administrators. The Archiving facilities will also only be available if the Archiving feature has been enabled by the **AllChange** administrator, see the *AllChange Administrator Manual*.

To archive parts use **Part | Archive Parts | Archive**, this will export all information about the selected parts to an external file in the project directory.



These parts may then be deleted using **Part | Delete Part**.

If at a later date you wish to re-import archived parts (e.g. to query them) use **Part | Archive Parts | Import**.

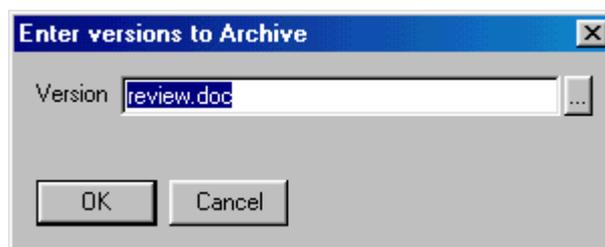


Parts are recreated with their original name and all their original information. On successful completion of the import you will be given the option to remove the imported parts from the archive. Once removed from the archive they cannot be re-imported at a later date.

Archiving Versions

It is also possible to archive individual versions of a part. This is useful for saving space and improving performance of checkin/out operations for a part with many old/redundant versions particularly if it is a binary file and each version is large.

In order to archive a version select **Part | Archive Parts | Archive Version**.



Each version archived is removed from the VC file for the part, thus saving space which may enhance performance of checkin/outs. The workfile for the version will be saved in a secure area and the **ArchType** field will be set to **OnLine**. OnLine Archived versions may be checked out for read only purposes and a copy of the saved archived version will be copied to the workspace. Archived versions may not be checked out for edit.

Archived versions of workfiles may be moved onto off-line storage and the **ArchType** field set to **OffLine** if hard disk space is at a premium. See the Administrators Manual for details of the secure storage of archived versions.

Advanced Use of Parts

Part Paths

Advanced Part Paths

In general you will select parts from a browser or viewer window, or from a browser dialog and use this as your method for choosing the parts that you wish to perform a particular operation on.

Sometimes, however, you may wish to type in part names yourself, in which case an understanding of the different ways of expressing parts and part paths will be useful.

Relative Paths

Any parts selected from a browser will usually be copied into the appropriate edit control in a command dialog as a *relative path name*.

If there is a part called `system` which is a child of `/` then the *full path* to `system` would be `/system`. The *relative path* to `system` when the current working part is `/` would be simply `system`. If there is another part called `file` which is a child of `system` (which is of type `subsystem`) then the full path to `file` would be `/system/file`. The path relative to `/` would be `system/file` and `file` relative to `/system`.

Unless stated otherwise, commands which expect parts as arguments will accept either full paths or relative paths which they will convert to full paths by prepending the current working part.

There are two special relative names for a part:

- . (dot) this indicates the current part
- .. (dot-dot) this indicates the parent of the current part

Wildcards

Wildcards may be specified in the *name* (but not the *path*) section for matching part names as arguments to most commands.

An `*` (asterisk) will match zero or more characters, e.g. `/*dir` would match all parts as children of `/` whose name end in `dir`.

A `?` (query) will match any one character, e.g. `/file.?` would match any children of `/` called `file.` plus a single character suffix.

More than one wildcard may appear in the name section, e.g. `/*file*.???` would match any children of `/` with `file` somewhere in their name and ending in `.` followed by three characters. Neither wildcard matches a `/`.

Note that, there is no special significance to the `.` (dot) character, so `*` matches all children of the current working part with or without a suffix while `*.*` matches only those parts which have a suffix.

Wildcards may not be used in the version-id section of a part name, though they may appear in the name section of a part version. In this case the wildcard(s) only match those components which have the version specified. Thus `*;003` might generate `myfile;003` and `yourfile;003` if those versions existed, but not `hisfile;003` if that version did not exist even though the component `hisfile` did exist.

`*;!b1` would match the version of all components in the current working part which were in baseline `b1`, omitting any version of components which were not in the baseline.

Specifying the Part in a Baseline

A *baseline-name*!! may precede a required path pattern and only those parts matching the path pattern specified which occurred in the named baseline will be selected.

In this case the * and ? wildcards *do* match / characters and the pattern must match the full part name stored in the baseline; the current working part has no significance.

Thus: `baseline!!*` would match all parts which were in the baseline `baseline`, `baseline!!*.c` would match any parts in the baseline ending in `.c`, `baseline!!/subsystem/*.c` would match any parts ending in `.c` in the baseline which were anywhere in `/subsystem` or any of its descendent subsystems to any depth, and `baseline!!*/manual/*.c` would match any parts in the baseline which were descendants of any subsystems named `manual`.

The *baseline-name! !path-pattern* mechanism described earlier may also be used to select versions appearing in a baseline. * and ? wildcards appearing in the path pattern do not match the ; version specifier but may be used in the version-id section, so `b1!!*.doc` selects all parts in the baseline ending in `.doc` but no versions of them while `b1!!*.doc;*` selects the versions but not the components.

In addition *!baseline-name* may be used to specify the version in a baseline. Thus `/subsystem/*;!b1` matches the version in baseline `b1` of all children of `/subsystem`.

Note that, although similar, `b1!!/subsystem/*;*` and `/subsystem/*;!b1` differ subtly: the former matches the versions of all descendants of `/subsystem` which appear in the baseline while the latter matches the version of only all current children of `/subsystem` which appear in the baseline.

If viewing a baseline when selecting certain command dialogs which operate on parts, ACE will automatically translate the baseline name into *baseline-name! !*;** which will cause all the versions in the baseline to be acted upon. For example, this may be used in order to **Check Out** all the versions in a baseline, see [Checking Out a Baseline](#).

Versions and Branches

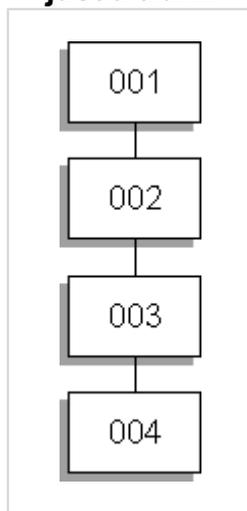
About Versions and Branches

Each component has versions associated with it. A version may be identified (or specified) by adding ; *version-id* at the end of a part name (e.g. `myfile;003`).

Versions are numbered automatically, and will be formatted according to the number of version digits specified in the configuration options.

The versions of a component may be thought of as a tree, in the simple case the version tree has just a main *trunk*.

Figure 5.6: Version tree with just a trunk



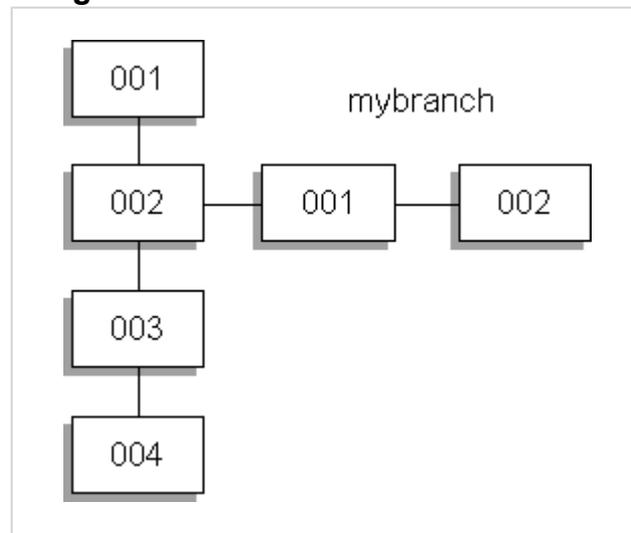
In [Figure 5.6](#) there are 4 versions on the trunk. In order to refer to a version on the trunk, simply specify the version number as the version-id (e.g. version 003 in [Figure 5.6](#) for a file called `myfile` would be `myfile;003`).

Sometimes, however, it is necessary to create a *branch*, for example:

- to fix a bug in an older version
- to have more than one line of development at the same time
- to log variants for different customers or configurations

In *AllChange* branches have names associated with them and each version on the branch is numbered sequentially.

Figure 5.7: Version tree with a branch



In [Figure 5.7](#) there is a branch called `mybranch` which originates from version 002 of the trunk. `mybranch` has 2 versions on it. In order to refer to a branch version specify the branch name and the version number on the branch as the version-id (e.g. for the first version on `mybranch` in [Figure 5.7](#) the version-id would be `myfile;mybranch.001`).

Branches may have branches themselves, although this is usually a fairly uncommon requirement.

Another method for expressing a version-id is to specify the baseline containing the version required (e.g. `myfile;!bline` would refer to the version contained in the baseline `bline`).

In summary, a *version-id* may be expressed as:

- *number*, e.g. `myfile;003`, meaning version 003 of `myfile`.
- *branch-name-list . number*, e.g. `myfile;mybranch.002`, meaning version 002 on branch `mybranch`. The *branch-name-list* may be a sequence of branch names separated by `.` (dot) (e.g. `mainbranch.experimental.test.001`). A branch name may contain (only) letters, digits and underscores and may not start with a digit.
- *!baseline-name*, e.g. `myfile;!b1`, meaning the version of `myfile` that was baselined in baseline `b1`.

Default Versions

Whenever a component type part is accessed without a version specifier there is a corresponding *default* version.

The default version is defined according to the following rules:

1. if the part is checked out to the current workspace then
 1. if it is checked out *for edit* then the *reserved* version is the default. If the parts was checked out for edit using pessimistic locking then this will be the new version created. If it was checked out for edit using optimistic locking, then this will be an optimistic lock version and not a true reserved version.
 2. otherwise the version checked out is the default
2. if the part is not checked out then as defined by any registrations (see [Workspace Registrations](#))
3. if there are no registrations then the topmost main version is the default

If a partial version-id is specified — for example a branch is specified but not the branch version — then the topmost version on the branch will be the default version, e.g. `myfile;mybranch` would access the topmost version on `mybranch` of the part `myfile`. A partial version-id of just `;` refers to the topmost version on the main line of development.

Commands which *require* a version to be specified as an argument (e.g. **check out**, **check in**) will use the current default version if no version (or partial version) is specified. Commands which accept either a version or a non-version part require a version to be specified with either a full or partial version-id to indicate the version that is wanted, e.g. `list component` lists information about the component in general while `list component;` lists information about the default version of the component (only).

Fetch Files

In addition to the mechanisms described earlier for specifying parts and versions, commands which accept these as arguments can also fetch the part and/ or version arguments from an external operating system file by means of the *fetch file* mechanism.

When a command expecting a part or version argument encounters a word starting with the @ (at) character, e.g.

@filename

it takes the rest of the word following the @ (*filename* in this case) as the name of a file which itself contains the parts/ versions desired. *Filename* may be an absolute operating system path or it may be relative to the current working directory. Part names appear in the file one-per-line. Blank lines and lines beginning with # (hash) are ignored. The @*filename* mechanism may itself appear as a line in a fetch file, causing further part names to be read from the specified fetch file before returning to the original fetch file.

A few commands accept operating system filenames as arguments. In addition to accepting these with native operating system wildcards they will accept a @*filename* argument which will be treated as above except the file contains filenames instead of part names.

Fetch files provide a very powerful means of grouping together related parts on which operations are to be performed. Unlike wildcards, they do not rely on the part names sharing similar spellings, are probably easier to remember, produce the same names every time and can be used to refer to large numbers of parts easily.

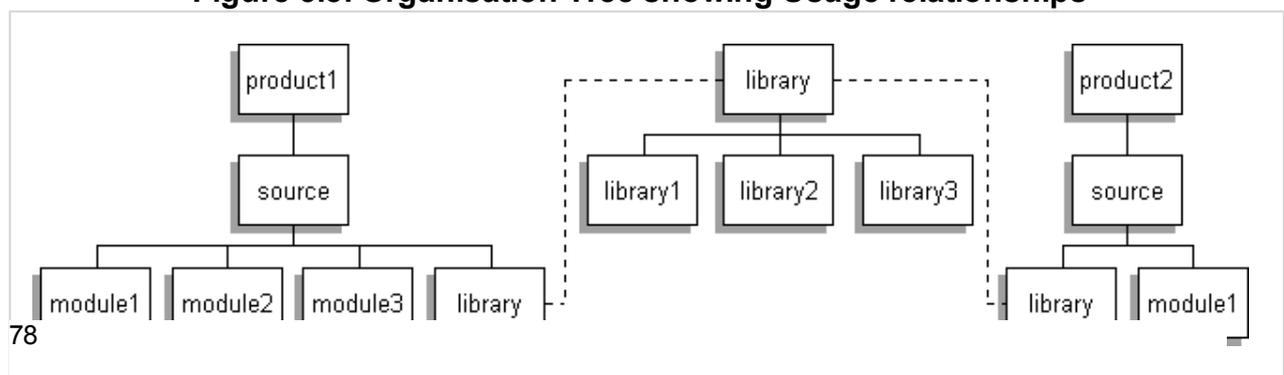
Uses Type Parts

About Uses Type Parts

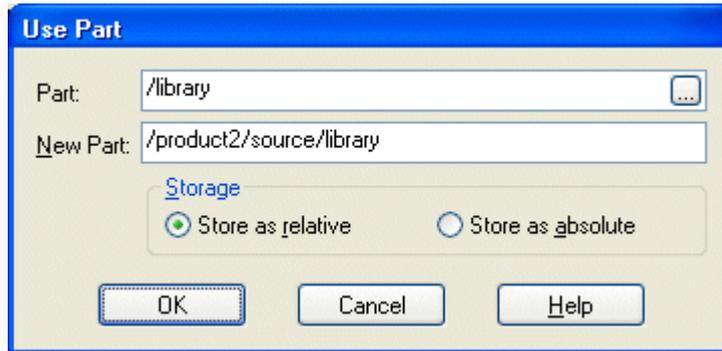
As well as subsystems and components, there is a third type of part known as a *usespart*.

Parts may *use* other parts thus allowing the sharing of CIs as illustrated in [Figure 5.8](#).

Figure 5.8: Organisation Tree showing Usage relationships



The next screen shows `product2/source/library` being created to use `/library`. Uses type parts are shown with a special icon .



The part being used may be specified as **Relative** in which case it will be relative to the new uses part created, or **Absolute**. If **Absolute** is selected and a *relative* path is specified in **Part**, then this will be made absolute by prepending the current working part.

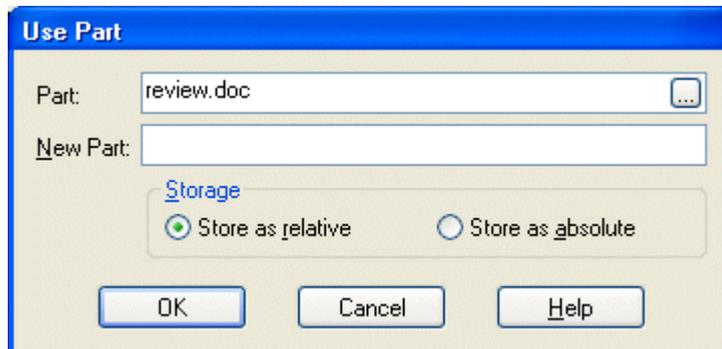
Double clicking on the uses type part in the subsystem list will change the current working part to the part *used*.

Double clicking on the uses type part in the selection list will show the part viewer for the uses type part. The part *used* will be shown in the **Location** field. All other alterable fields are invalid for uses type parts.

Creating Uses Type Parts

In order to create a part of type **Uses**, use the **Part | Use** dialog.

A Uses type part allows links or relationships between parts to be expressed. Specify the part to be *used* as the **Part** and the name of the new part which will *use* the other part as the **New Part**.



Checking Files In and Out of AllChange

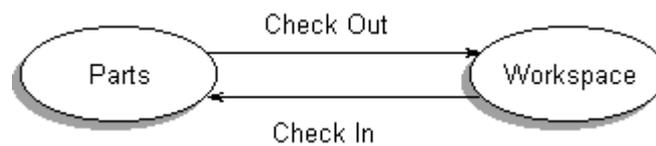
About Checking Files In and Out of AllChange

In **AllChange** most of the time you will deal in terms of *parts* which are the declared configuration items that **AllChange** is controlling for you. Each part has a corresponding directory or file associated with it and you should use the parts browser in a similar way that you would use a file browser to change directory and select files.

In order to check *files* in and out of **AllChange**, you therefore check *parts* in and out of **AllChange**.

Note that the term part may have been mapped to site specific nomenclature (e.g. CI) in which case all references to part will show using the mapped nomenclature instead.

Parts are normally checked out from the parts database in **AllChange** into a *workspace* and checked in from the *workspace* back to the parts database.

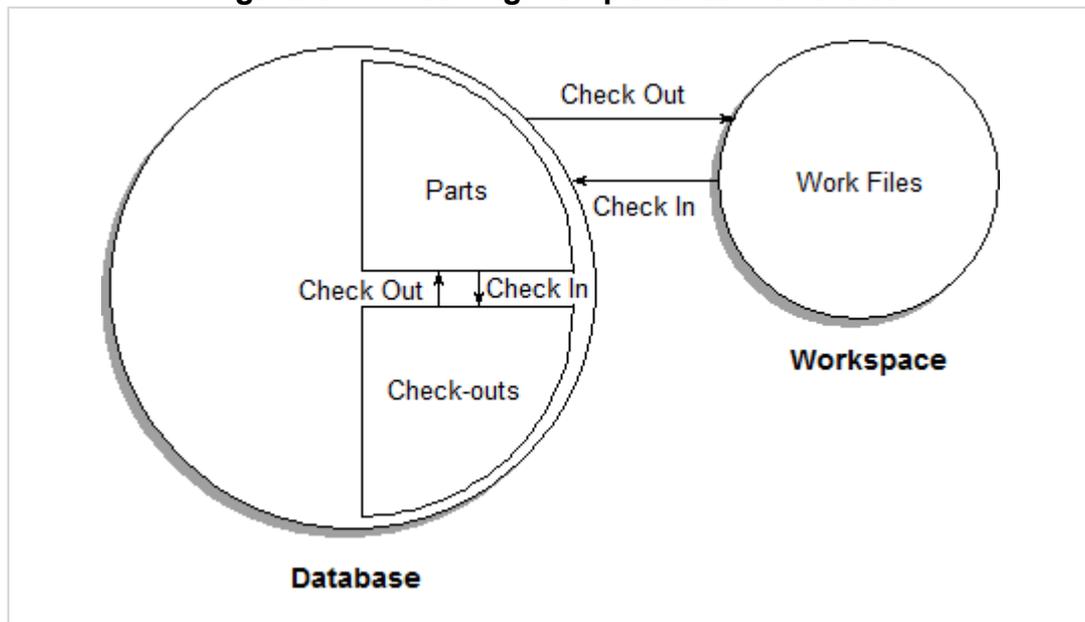


It is also possible to extract parts to a directory which is not a workspace and to check in files from a directory which is not a workspace.

A log is maintained of all parts which are checked out to all workspaces as a *check-out record* in the *check-outs* database.

Note that the term check-out may have been mapped to site specific nomenclature (e.g. issues) in which case all references to check-outs will show using the mapped nomenclature instead.

Figure 6.1: Checking Components In and Out



On checking out a version of a component, a copy of the file contents for that version (the *workfile*) is created in a file in the current workspace directory and the check-out log is updated to reflect this information.

On checking in, the opposite action is taken (i.e. the workfile is removed and the check-out log is updated to reflect the fact that the component is no longer checked out).

Checking in and out may also cause the status of the part to be changed.

Furthermore, parts may be of a class which has been defined as *Requiring a CR*. This means that in order to make a change to a part of such a class, there must be a change request (CR) — see [Creating and Managing Change Requests](#) — which authorises the change before the part may be checked out for edit. A CR which authorises a change is referred to as being a *Valid for change CR* and must comply with various criteria - these are summarised in [Valid for Change CR Criteria](#).

The CR which authorises the change must be specified on performing the check out . When a component is checked back in which was checked out against a CR, the new version created is automatically associated with the CR. This means that the CR automatically contains a list of all the parts and specific versions which were modified to implement the change. This information can be used later for populating baselines with the required code to implement CRs which are to be included in the baseline .

In the *out-of-the-box* configuration any parts which do *not* require a CR for modification will instead require a Comment to be specified as to the reason for the change on **CheckIn**.

The class of a component may also specify that the part is to be **Kept checked out**. If a component class has this attribute then when components of that class are checked in from edit, they will be automatically checked out again read only. This facility enables files to be always maintained checked out read only. This can be useful when a number of files are required in order to develop other files, e.g. for compilation purposes.

Whenever parts are checked out for the purposes of making a change, and later checked in, AllChange automatically stores a new version of the part thus maintaining a complete version history of all changes made to each part. Any old version may be retrieved if required hence changes cannot be lost. Against each version the information as to who made the change and when it was made is also automatically recorded. Using AllChange's version control facilities also enables multiple users to make changes with confidence that they will not inadvertently overwrite each others changes.

When checking out for the purposes of making a change (rather than for the purposes of viewing a file's content), AllChange supports 2 models for ensuring multiple users do not overwrite each others changes

- [Pessimistic Locking](#) - this is the default and perhaps more traditional approach to version control
- [Optimistic Locking](#) - this provides for a less rigid method of controlling parallel development

See [Locking Models](#) for further details on these two models.

Valid for Change CR Criteria

CRs which may authorise a change (i.e. a check out for edit) must match certain criteria according to the configuration that you are using.

In order to be valid for change CRs must:

- be in a status which is defined as *valid for change*. Statuses which are valid for change are defined by your **AllChange** administrator in the life-cycle definitions.
- be assigned to you, or to a group that you are a member of (only if the **Must be assigned** configuration option is selected - this may be modified by your **AllChange** administrator)
- the part to be checked out must be within the part tree defined by the CR **TopPart** field.
- the CR must have the part to be checked out as a part affected (only if the **Must be affected by part** configuration option is selected - this may be modified by your **AllChange** administrator)
- must not be *locked*, see [Locking CRs](#)

Locking Models

Locking mechanisms are methods of ensuring that multiple users can work together on the same set of parts at the same time without getting in each others way or overwriting each others changes.

AllChange supports 2 locking models:

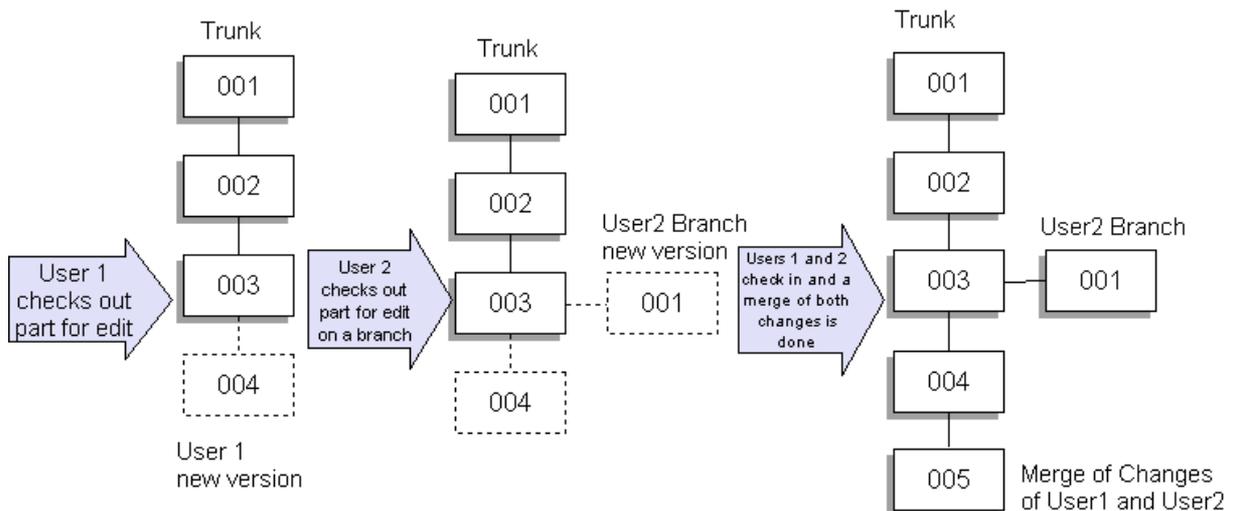
- Pessimistic Locking - with this model when a part is checked out for modification purposes (for edit), a new version is created and reserved to that user (locked) and this is the new version that will be used when the modified part is checked back in.
- Optimistic Locking - with this model, when a part is checked out for modification purposes (for edit), no fixed new version is created and the new version is not determined until the modified part is checked back in.

Pessimistic Locking

The pessimistic locking model predetermines what the version identifier of the new version will be at the time the part is checked out for edit. This new version is created at check out time with a special flag to denote that this version is a *place holder* (the **New Version** flag). Other users may see that this place holder exists but it will not be visible by default.

No other user will be able to check out this version for modification on the same line of development. To allow multiple users to modify the same version at the same time (parallel development) *branches* must be used and if required changes on different branches may be *merged* to create a further new version on that line of development. See [Creating Branches](#) and [Merging Parallel Lines of Development](#) for the facilities provided for this.

The diagram below shows this model of parallel development.



With this model users may develop independently of one another without concerning themselves with what others may be doing. At some point after changes have been checked in a single merge of the resultant changes may be made as a separate development activity.

This approach to parallel development has the following advantages:

- It is possible to prevent parallel development altogether if this is desired
- A full record of all parallel developments is maintained as well as the merged end result. This in turn means that each change may be released without others having been completed
- It is known in advance if parallel development is about to occur and a decision as to whether this is desirable or necessary can be made in the full knowledge of what other parallel developments are underway
- The amount of merging required is minimised

The disadvantages of this approach are:

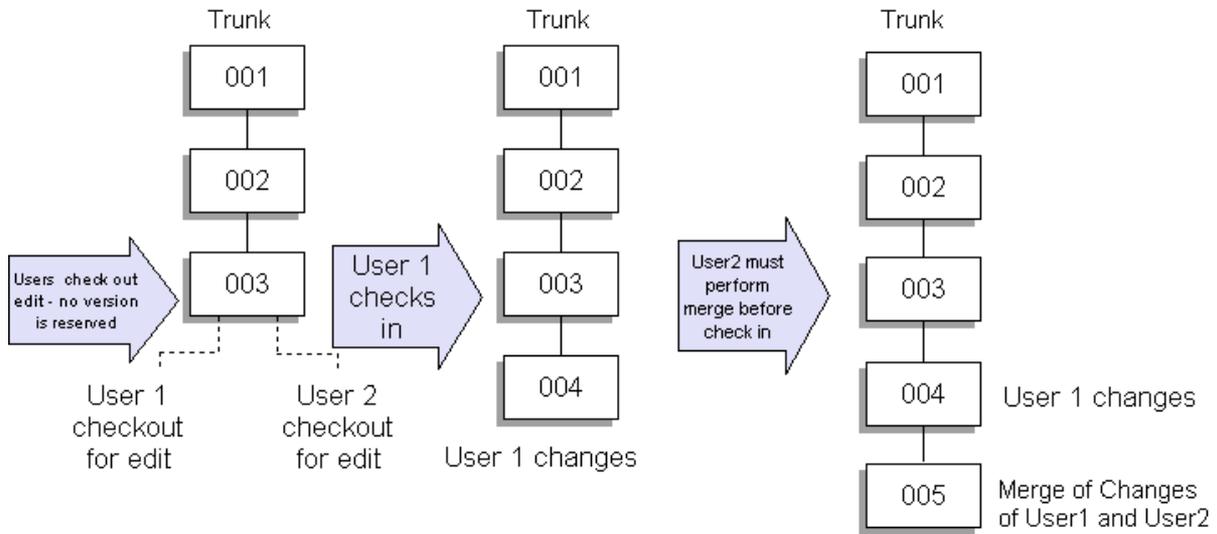
- Users need to consider in advance whether they will be developing in parallel and create a branch
- The merging process requires an additional step

Optimistic Locking

The optimistic model allows any number of users to check out the same version of a part for edit at the same time. On checking in AllChange performs a check to see if there have been any new versions since the part was checked out. If there have, then a merge with these changes must be performed before the part may be checked in. This is known as optimistic locking as at the time the part is checked out for edit the user does not know what new version will be created nor whether any merging will be required, it is *optimistic* in the hope that this will not be the case.

Users may merge repeatedly as they develop and others check in their changes.

The diagram below shows this model of parallel development.



With this model users must merge others changes with their own before checking in and will need to ensure that they are working with the latest versions of files they have not yet modified. They may perform repeated merges or a single large merge when their developments are complete. The [Update Workspace](#) and [Update](#) functions supports updating workspaces with latest versions for unmodified workfiles and merging modified workfiles.

Optimistic Locking requires both **Joint Editing** and **Optimistic Locking** are enabled in the Configuration Options.

This approach to parallel development has the following advantages:

- Users can easily develop in parallel without initially needing to consider the changes that others may be making
- The number of steps to merge is minimised

The disadvantages of this approach are:

- There is no record (version) stored for each users changes prior to merge. It is therefore harder to backout the merge if required
- The amount of merging required is more.

Workspaces

About Workspaces

Workspaces are simply operating system directories that are known to **AllChange**. They are used to hold local copies of particular versions of parts for examination, editing and building purposes. The parts are held in a workspace in the form of operating system files.

Note that the term workspace may have been mapped to site specific nomenclature (e.g. sandpit) in which case all references to workspace will show using the mapped nomenclature instead.

Workspaces may be defined as *ftp workspaces* in which case the files will be transferred to a remote workspace directory using FTP. This allows files to be managed for remote platforms including deployment to a web server.

In addition workspaces may be defined as *web workspaces*. These may be used with the web browser interface to AllChange (as well as the Windows interface if this has been enabled) See [Web Workspaces](#) for more details.

The files that exist in a workspace may be put there by *checking out* a version of a part to a workspace, or they may be derived objects resulting from a *build* which do not have a corresponding part. Files in a workspace are referred to as *workfiles*.

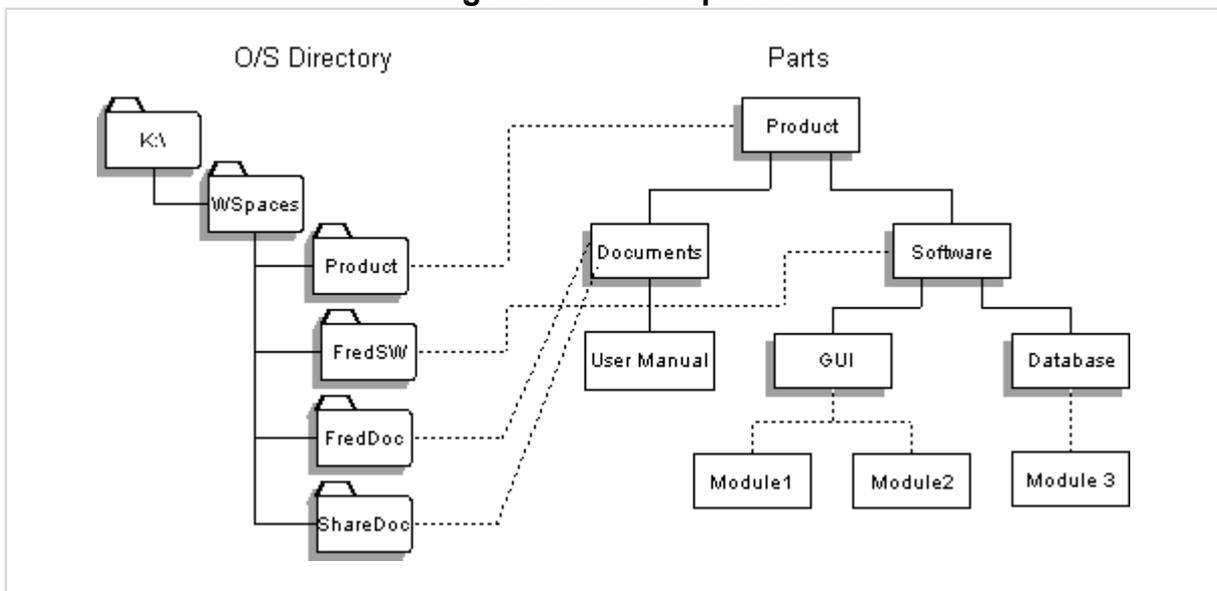
It is recommended that all editing and system building takes place in a workspace. Other workspaces are isolated from changes made to files while the files reside in a workspace. Different users may have several workspaces that they may use and workspaces may be shared by several users.

A user must **Attach** to a suitable workspace before invoking any action which accesses a workfile, such as **Check Out** or **Build**. This allows you to ensure that only authorised users access files in a particular workspace. The current workspace attached to is shown in the status bar.

A workspace has a single subsystem associated with it — the *workspace part* — and a corresponding operating system directory — the *workspace directory*. The intention is that all local work undertaken for descendants of this subsystem by users of the workspace will take place in the workspace.

Depending on the system a workspace may be associated with the top level part of a whole project, or there may be several workspaces each associated with some subset of the project such as source code and documentation. Equally there may be just one workspace associated with a particular part which multiple users share, or there may be separate workspaces all associated with the same part for different users.

Figure 6.2: Workspaces



In [Figure 6.2](#) several different possible workspaces are shown for a parts tree. The workspaces are simply directories and could reside, for example, on a network drive (k: in the diagram). Four workspaces are shown with their corresponding part and directory. All the workspace directories are within `k:\wspaces` for convenience.

Product

This workspace is to be used for the whole of `/product`. When using this workspace all work on any components in the tree would be performed in the *workspace directory* `k:\wspaces\product`

FredSW

This workspace is intended for user *fred*'s use only. Only parts within the subtree */Product/Software* may be worked on in this workspace. When using this workspace any workfiles created for component versions would be placed in the *k:\wspaces\fredsw* directory.

FredDoc

This workspace is intended for user *fred*'s use only for development of */Product/Documents*. When using this workspace any workfiles created for component versions of the documents would be placed in the *k:\wspaces\freddoc* directory.

ShareDoc

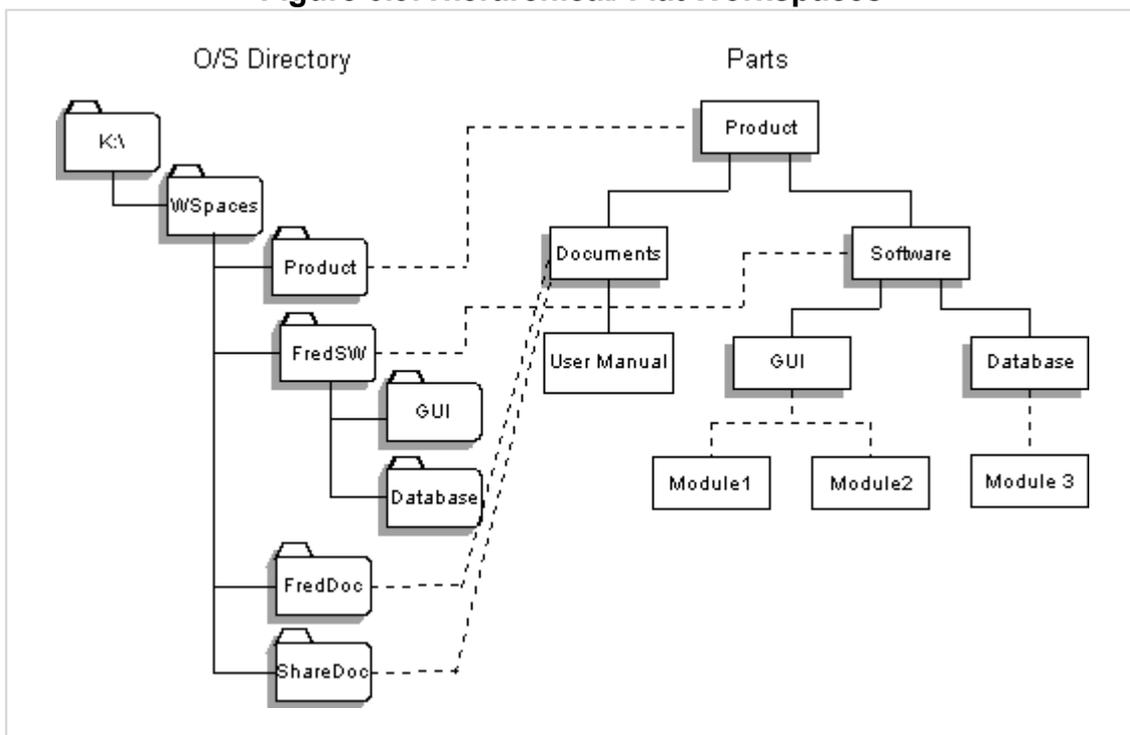
This workspace is intended as a shared workspace amongst several users for the product documentation. When using this workspace any workfiles created for component versions of the documents would be placed in the *k:\wspaces\sharedoc* directory. Several users would have access to this directory and so could access each others files.

All files in a workspace will reside in the *workspace directory*.

A workspace may be defined as *flat*, as shown in [Figure 6.2](#). This means that all files in the workspace will reside directly in the workspace directory.

Alternatively, a workspace may be *hierarchical* in which case the workspace should comprise a directory tree matching the part tree for the workspace part. Workfiles will be placed in the correct directory corresponding to the part.

Figure 6.3: Hierarchical/ Flat Workspaces



[Figure 6.3](#) shows what the directory structure should be if *FredSW* workspace were to be hierarchical (i.e. it mirrors the part tree for the *workspace part*). All files will be placed in the appropriate subdirectory depending on whether it is a *GUI* component or a *Database* component.

Workspace definitions may be created and updated from within ACE (see [Creating and Modifying Workspace Definitions](#)) or by the **AllChange** administrator using the configuration tool **ACCONFIG** (See [All-Change Administrator Manual](#)).

As described in [Workspace Registrations](#), workspaces may also have associated *registrations* which are used to control what pools ([Pools](#)) and default versions of parts are to be used while attached to the

workspace. With the aid of this facility multiple workspaces may be employed conveniently to develop and maintain different versions of a system simultaneously.

In general all workspaces should have a unique directory associated with them, however it is possible to have situations where workspaces have overlapping directories. *This is not recommended.* When All-Change is trying to determine what part corresponds to a file (e.g. when checking in a file from a third party tool or the file browser), it does so by determining which workspace the file is in. If more than one workspace is possible then the following rules apply:

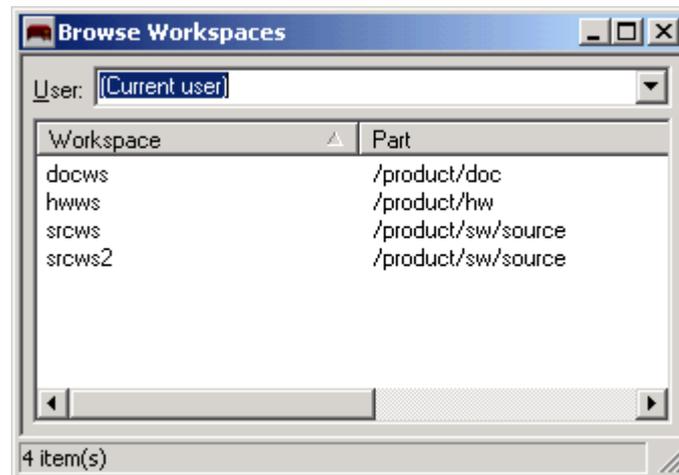
- If the **Try current workspace** configuration option is selected then use the current workspace if this is valid for the file
- If the **Ask if more than one match** configuration option is selected then if more than one workspace is valid for the file prompt the user for which one to use
- If more than one workspace is valid for the file use the first one encountered that matches.

If no workspace matches a file being checked in then 2 scenarios are possible:

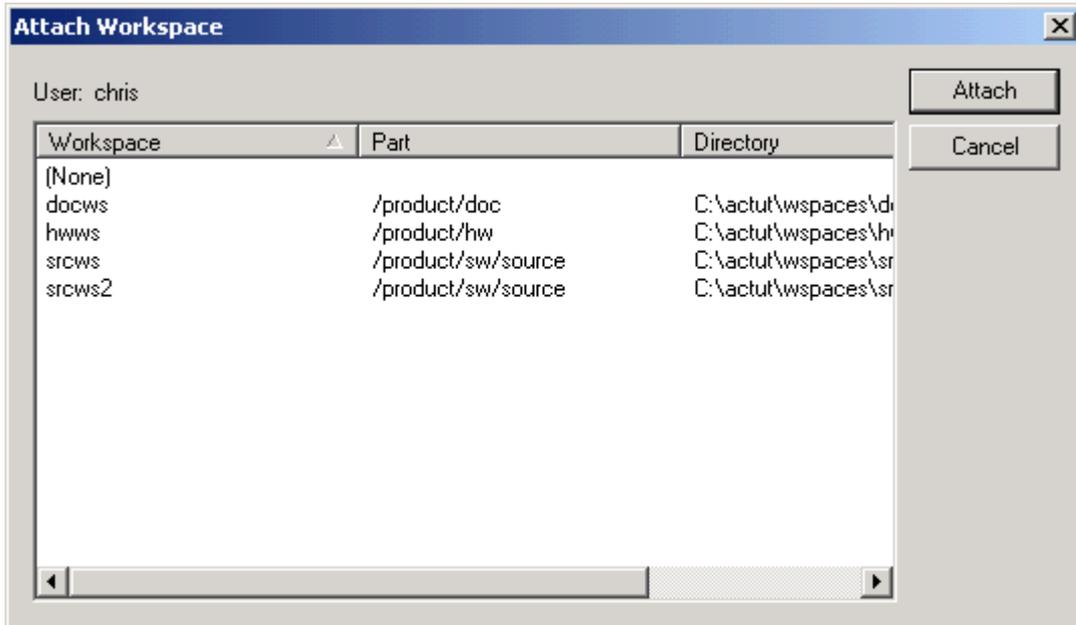
1. If the **Allow check-in with no workspace** configuration option is not selected (default) then an error will be issued.
2. If the **Allow check-in with no workspace** configuration option is selected then a dialog will be presented allowing the selection of the part subsystem into which the files/directories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

Attaching to a Workspace

In order to attach to a workspace use the **Browse Workspace** window accessible from the **Workspace** menu,



or the **Attach Workspace** dialog also accessible from the **Workspace** menu or by double clicking on the **Workspace** status bar.



The window will show those workspaces that the **User** shown may attach to (by default you will be the **User**). A user may attach to a workspace if he is associated with it either explicitly or is a member of a group which is associated with the workspace. The users/ groups that are associated with a workspace are defined by the **AllChange** administrator when the workspace is defined.

Select the required workspace and select the **Attach** button in the dialog (or double click on the workspace), or **Attach** from the context menu in the browser.

The workspace attached to will be shown in the status bar. Furthermore, the current part and current directory will also be changed to reflect the part and the directory for the workspace.

(Attaching to a workspace is the equivalent of changing your current directory in an operating system to the one where you wish to do work.)

On attaching to a workspace, if it is defined as being an *autoupdate* workspace you may be prompted to update the workspace to any new versions of parts checked out to the workspace.

Having attached to a workspace, many operations previously unavailable (greyed out menu items and toolbar buttons) will become available, such as **Check In**, **Check Out** for parts/ files.

Creating and Modifying Workspace Definitions

Workspaces (but not web workspaces) may be defined or modified from within ACE or from the administrator tool ACCONFIG (See [AllChange Administrator Manual](#)). Here the facilities from within ACE will be documented.

From the **Workspace** menu **Add** or **Edit** may be used to invoke the **Workspace Editor**. In order to modify the definition of an existing workspace or delete a workspace you must open the **Workspace Browse** window accessible from **Workspace | Browse Workspace**. This allows you to select the workspace that you wish to modify. A selected workspace may also be deleted using **Workspace | Delete**.

Each workspace definition has the following properties:

Name

An arbitrary name for the workspace. The name must be comprised of alphanumeric characters and the `_` character and may not include a space character.

Part

A part in the parts tree with which the workspace is associated: on attaching to the workspace this will become the current part. The part should be specified as a full part name e.g. `/product/software/source`. This may **not** be a uses type part.

Directory

The name of the physical directory associated with the workspace. This directory *must* be unique for each workspace defined, but may be on a local `C` drive or on the network. The directory may be expressed as either a drive mapped directory or as a UNC.

Hierarchical

If this is selected then the workspace is *hierarchical* otherwise it is *flat* (see [Figure 6.3](#))

Autoupdate

If this is selected then on attaching to the workspace **AllChange** will display the [Update Workspace](#) dialog allowing the components in the workspace to be updated to the required parts/versions

Users

A list of users or groups associated with the workspace. This is typically used to restrict who may attach to the workspace. If a group is specified then any member of the group may attach to the workspace. The users may be selected using the **Add** button, or removed using the **Delete** button.

In addition a workspace may be defined as an *FTP* or *Web* workspace for use when managing files on remote machines (e.g. Unix, VMS) and Web servers.

All the above information still needs to be supplied for an *FTP* workspace including the **Directory**, as this is used as a temporary transfer area before/ after FTP transfers occur.

In addition the following **FTP** information needs to be supplied. **Note** that if FTP Workspaces are not enabled then this part of the dialog is disabled.

FTP Workspace

Host name: this should give the name or IP address of the host computer

Host OS: this should specify the host operating system/FTP server

Remote path: this should specify the path to the directory on the remote machine that is to be the workspace directory.

Web deployment

Host name: this should give the name or IP address of the Web server

Host OS: this should specify the host operating system/FTP server

Remote path: this should specify the path to the directory on the Web server that is to be the workspace directory to/ from which deployment will occur.

For a fuller explanation of workspace definitions see the [AllChange Administrator Manual](#)

Checking Out

About Checking Out

When a copy of a particular version of a component is required in a file for viewing or editing purposes it should be *checked out* to the required workspace.

Ensure that you are attached to the workspace where you wish the file to be placed. Note that if the directory into which the workfile is to be placed does not exist then **Check Out** will create the required entire directory hierarchy as necessary.

The part(s) that are to be checked out may be selected from:

Part Browser/ Viewer: subsystems or components may be selected. If a subsystem is selected then the *default version* of all the parts in the subsystem tree will be checked out. If a component is selected without any specific version then the *default version* will be checked out. Any browser list which displays parts may be used.

File Browser: the directory into which the workfiles are to be placed may be selected, or specific files which are to be checked out may be selected. **Check Out** will then prompt for which parts to check out. If **AllChange** cannot find a workspace that matched the directory an error will be generated.

Check Out is available from the **Part** or **File** menu or from the toolbar and context menus where files/ or parts are listed. Also parts may be **dragged** from the part browser and **dropped** onto the file browser to cause a **Check Out**.

An optional toolbar button is also available for invoking the **CheckOut** dialog with the **For Edit** box ticked. The button is called **Check Out For Edit** and can be added by right-clicking the toolbar, selecting **Customise** and dragging it from **General** list to the desired toolbar location. The button does not appear on any toolbar by default.

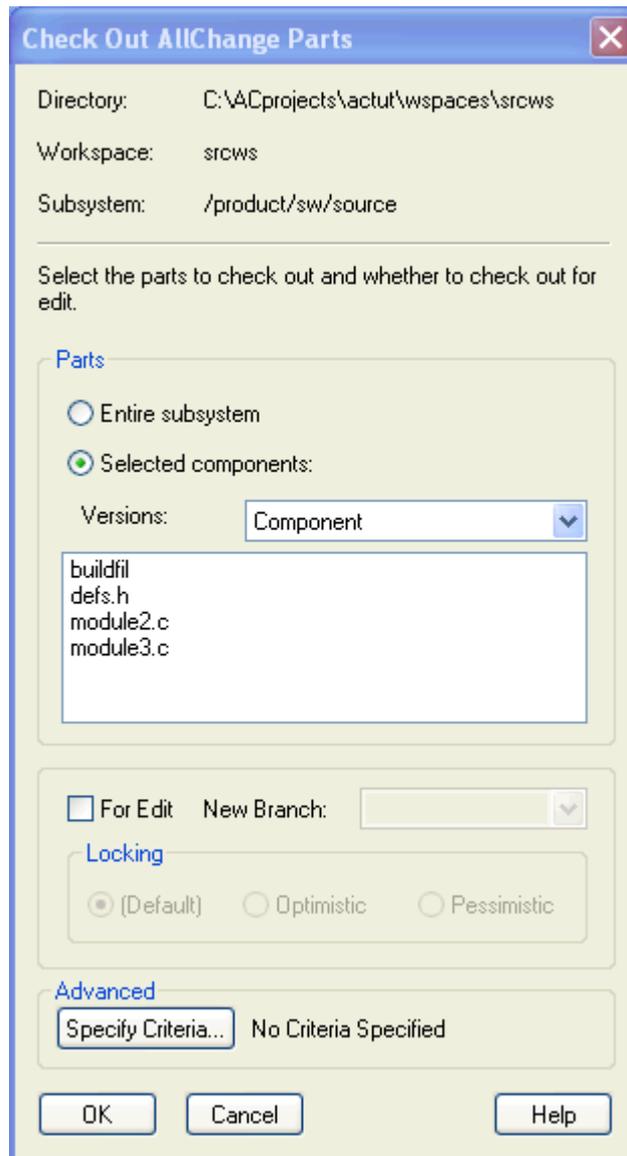
Furthermore parts may be checked out from the [Cycle View](#) by double clicking on a Check Out status.

Note that only component versions are actually checked out and logged, subsystems are not logged. It is important to ensure that you check out the correct version of the component which may be selected in a variety of ways:

- Selecting a version explicitly from the part browser or viewer or check out files dialog. The version selection may be filtered according to Default, Registered or Top, see [Part Browse Filters](#) for a definition of these
- Selecting the component which is cause the Default version to be checked out

In general a component may only be checked out if there is currently no version of the component already checked out to the same workspace. The exception to this is that if a component is checked out for read only purposes, then a checkout for edit is permitted (i.e. the read only check-out may be converted to an edit check-out).

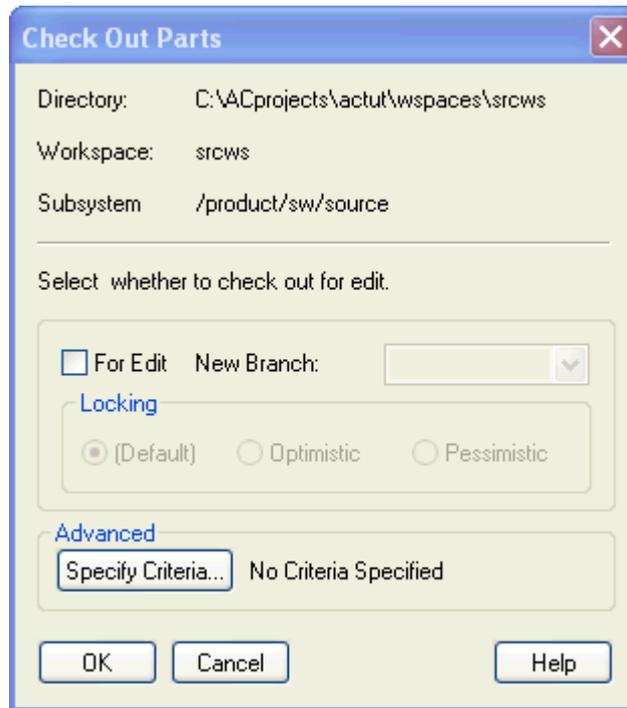
If a directory was selected the **Check Out Parts to Directory** dialog shown below will be presented.



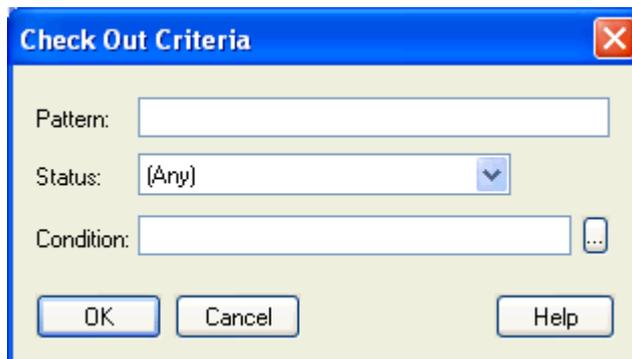
Select **Entire Subsystem** if the *default version* of all parts in the subsystem tree are to be checked out and **Selected Components** if only certain components are to be checked out. The **Versions** combo box may be used to modify which versions of components are shown in the component list, see [Part Browse Filters](#) for details of these options. Only components which are not already checked out for edit to the workspace will be shown as these are the only ones that may be checked out.

Select the required components from the list.

If parts were selected then the **Check Out Parts** dialog will be presented.



Check Out Criteria may be specified to define which parts are to be checked out.



If a **Pattern** is specified then the part/file name being checked out must match the pattern specified.

If a **Status** is selected the version to be checked out must be in that status for the part to be checked out.

The **Condition** allows any arbitrary criteria to be specified to further define which parts are to be checked out.

When checking out a subsystem or directory and the **Show Obsolete Items** option is not selected then any obsolete subsystems or components encountered will be skipped. If an obsolete item is explicitly selected or **Show Obsolete Items** is selected then obsolete items will be checked out.

There may be two reasons for wishing to **Check Out** a component:

1. For read-only purposes in order to examine a document or create a reference copy for others to examine.
2. In order to edit that version and create a new one.

Check Out Read-Only

If it is desired to check out for read only purposes then the **For Edit** and **New Branch** options should *not* be selected. **Check Out Criteria** may be specified to define which parts to check out.

The requested version of the **Parts** specified will be extracted from the archive (i.e. VC file) and placed in a file (the *workfile* in the current workspace directory).

Note that if the part was already checked out to the workspace then the checked out version will be updated to the one requested, if it is already the requested version then it will be skipped. The output window will give an indication of any parts skipped.

The extracted file may be viewed using **Part** or **File | Edit in AllChange** or using any other tool to view a file from outside **AllChange**.

The version checked out becomes the default version for future references to the component when attached to the workspace to which it is checked out.

When you have finished with the file you *must* perform a **Check In** — *not* delete the file. This allows **AllChange** to remove the check-out log from its database.

Check Out Criteria may also be specified to define which parts to check out.

Checking Out for Making a Change

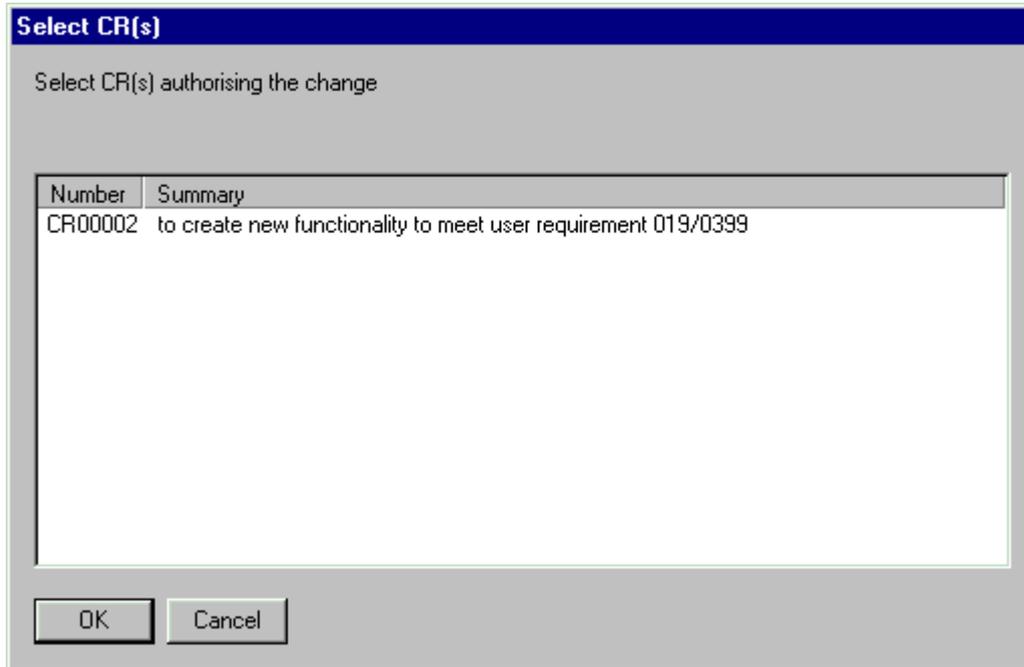
If it is required to check out for the purposes of making a change then the **For Edit** option should be selected. The locking type for the check out should also be selected. This may be one of:

- **Default:** this is the default locking specified for the class of part being checked out - this is determined by the AllChange administrator in the class definition (see Classes in the AllChange Administrator Manual)
- **Optimistic:** this is the optimistic locking model for parallel development (see [Optimistic Locking Model](#))
- **Pessimistic:** this is the pessimistic locking model for parallel development (see [Pessimistic Locking Model](#))

*Note that the Optimistic locking type may only be selected if both Joint Editing and Optimistic Locking are enabled in the configuration options by the **AllChange** Administrator.*

Check Out Criteria may also be specified to define which parts to check out.

If any of the parts being checked out are of a class which is defined as *requiring a CR to authorise a change*, then **CheckOut** will prompt for you to select the CR(s). The CRs which are valid for change must match the criteria as defined in [Valid for Change CR Criteria](#).



If there are no CRs which match *all* the above criteria an error will be issued to indicate this.

As each part is checked out the CRs selected to authorise the change are checked for validity for that part, i.e. the part to be checked out must be within the part tree defined by the CR **TopPart** field.

The CRs authorising the check out for edit is stored in a **Comment** field in the check-out log record.

The requested version of the **Parts** specified will be extracted from the archive (i.e. VC file) and placed in a file (the *workfile*) in the current workspace directory.

If the part was already checked out for read only purposes to the workspace, it will be automatically checked back in prior to the check out for edit.

A new version of the component will be created in the parts database as a place holder for when the changes are checked back into the system. This new version becomes the default version for future references to the component when attached to the workspace to which it is checked out.

The **Flags** will show **NoVersion** to indicate that this version is a place holder and is (in *AllChange* terminology) *checked out for edit* purposes. The user that performed the check out will be shown as the **User** and the **Date** will show the date and time of the creation of the version. If the part is of a class which has a life-cycle, then the status of the version will have been progressed to the status marked as the *check out status*; for example, in the supplied configuration **controlled_source** parts have a life cycle whose *check out status* is **Editing**.

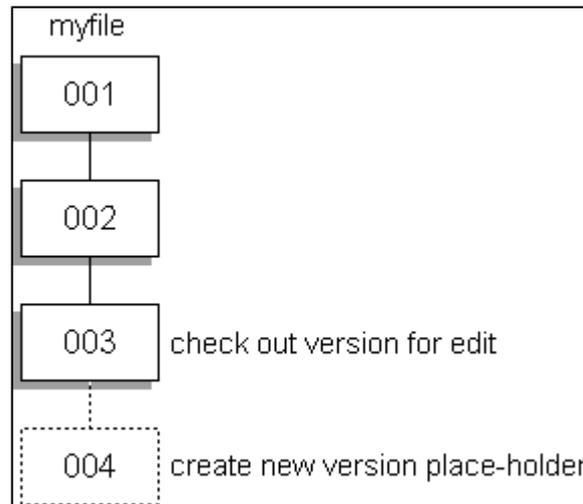
If the part viewer is open showing the checked out component, then the new version will be shown in the version tab on completion of the check out for edit. The *workfile* may be edited as required (either using **Edit** within *AllChange* or using any editor from outside *AllChange*).

When you have finished with the *workfile* then you *must* **Check In** the part.

Pessimistic Locking

If the part was checked out using *pessimistic locking* the new version created determines the version identifier that will be used when the part is checked back in.

Figure 6.4a: New Version Created on Pessimistic Check Out for edit



Unless joint editing has been specifically enabled at your site then if the version of the component being checked out is currently checked out to any other workspace then an error will be raised. If joint editing has been enabled then a branch must be created as the new version placeholder.

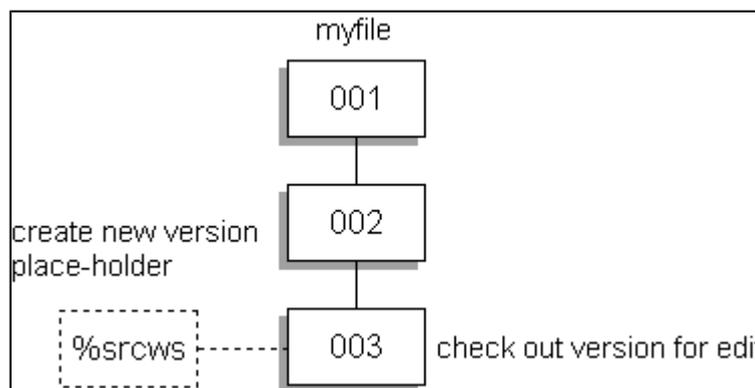
Optimistic Locking

If the part was checked out using *optimistic locking* the new version created has a generic version identifier based on the name of the workspace in the form:

[<branch>] %<workspace name>

The version that will be created on check in is determined at the time of the check in.

Figure 6.4b: New Version Created on Optimistic Check Out for edit



If the part is already checked out using pessimistic locking to another workspace then this must be checked back in before an optimistic check out for edit may take place. In this case an error will be issued.

Creating Branches

If the **New Branch** option is used a new branch from the version checked out will be created with the specified branch name if the version checked out is not already on that branch. The branch name may be selected from a list if these have been defined by the **AllChange** administrator or a user with permission to

define branch names using **Part | Edit Branches**. A branch name may also be typed in instead of selected from the list if this has been enabled as a configuration option by the **AllChange** administrator.

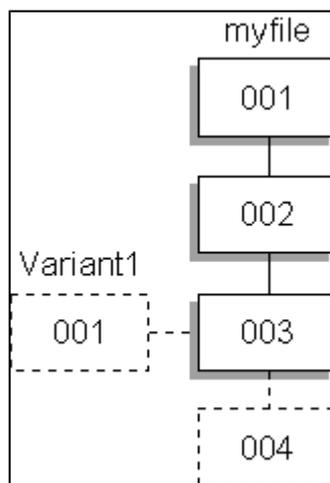
With pessimistic locking version 001 of the new branch will be the new version created. With optimistic locking the new version will be identified with the branch name and the workspace checked out to.

If joint editing is enabled and pessimistic locking is used, then if the version to be checked out for edit is already checked out for edit to another workspace then a branch *must* be created otherwise an error will be raised.

Facilities are also provided to allow a part checked out for edit to be moved to/from a branch using **Workspace | Move Check-outs to Branch**. If a branch is named incorrectly then it may be renamed for a component or an entire subsystem using **Part | Rename Branch**

By the use of branches you are assured that users do not overwrite each others changes and different variants of a file may be accommodated for different product variants. Branches may also be used to manage parallel development with each line of development using a different branch.

Figure 6.5: Parallel Development Using Branches



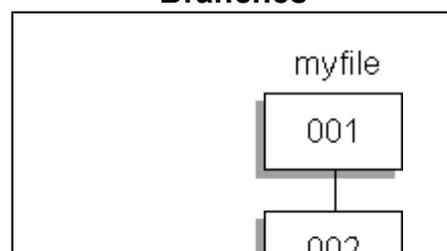
In [Figure 6.5](#) version 003 is checked out for edit to two different workspaces with version 004 reserved for one and the branch `variant1` created and version 001 on the branch reserved for the other.

With optimistic locking branches do not have to be created and in order to prevent users overwriting each others changes, AllChange verifies that no new versions have been checked in since the part was checked out for edit before checking in. If new versions have been created then a merge must be performed before the part can be checked in.

Merging Parallel Developments

It is sometimes necessary to incorporate the changes made in one branch (e.g. a bug fix) into the changes made in another (e.g. the main line of development). The process may be automated for text files using the merge tool.

Figure 6.6a: Merging Branches



[Figure 6.6a](#) illustrates merging the changes made on a `BugFix` branch into the main line of development, to create a new version.

The version to be the result of the merge should be checked out for edit. In the example shown in [figure 6.6](#) version 004 should be checked out for edit to create version 005 with the **NoVersion** flag.

If optimistic locking is used then it may also be necessary to merge the changes another user has made and checked in since the version was checked out for edit.

Figure 6.6b: Merging Changes into workfile

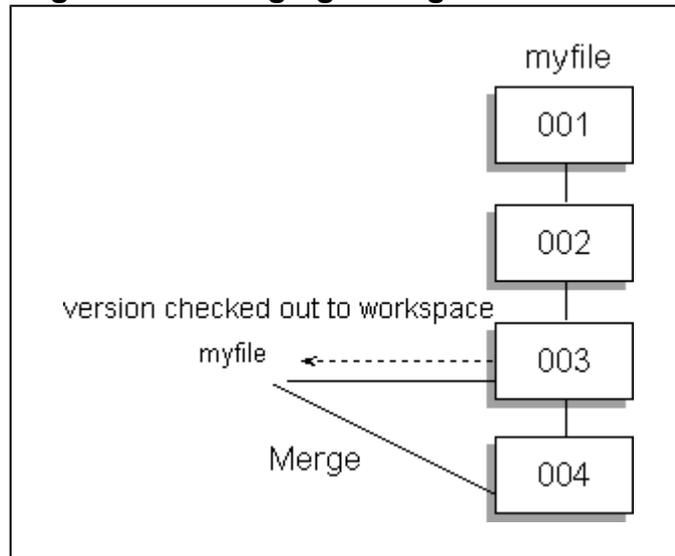
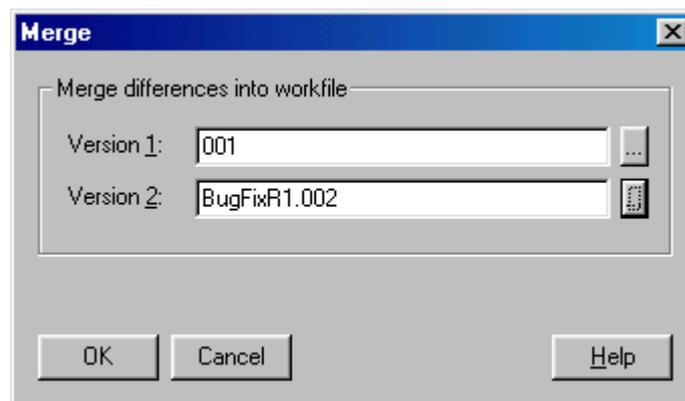


Figure 6.6b illustrates merging the change made from version 003 to version 004 into the workfile which originated from version 003

Merging Versions in a File/Part

The merge tool is available from **Part | Merge**: it will merge the differences between the two specified versions into the current workfile. To achieve the merge example shown in [figure 6.6](#), **Version 1** should be version 003 and **Version 2** should be `BugFix.001`.



The results of the merge will be displayed in the visual merge tool **VisMerge** and this may be used to resolve any clashes and save the resulting file — see Merging for details of the tool. If an arbitrary field named **Merge From** is defined for the part then details of the versions merged will be stored in that field.

If the **Differencing Application** in [Misc | Options](#) is set to Araxis Merge then this will be used instead of the built in VisMerge tool.

It is also sometimes desirable to perform a merge based on 3 existing files rather than versions of parts. **Merge** is available from the **File** menu and allows the merge tool to be invoked with up to 3 files selected. If less than 3 are selected then the remainder will be prompted for allowing files to be specified from different directories.

Merging is only available for Text files.

If attached to an FTP workspace, then if the file is marked as text (a **Binary** field is defined and this is set to the value **No**), then the workfile is transferred to a temporary location prior to the merge. After the merge is complete (the merge tool is exited), the file is then transferred back to the remote location. Note that built in merge tool will be used in all cases regardless of the **Differencing Application** setting in [Misc | Options](#). For files which are not marked as text (either the Binary field is not defined or the value is Yes or blank) the merge tool cannot be used, but the files required to perform the merge will be placed in the workspace to allow the merge to be performed manually or using another merge tool on the remote platform - a message to this effect is issued.

Merging Subsystems/Directories

If the **Differencing Application** in [Misc | Options](#) is set to Araxis Merge then 3 way directory or sub-system merging may be performed.

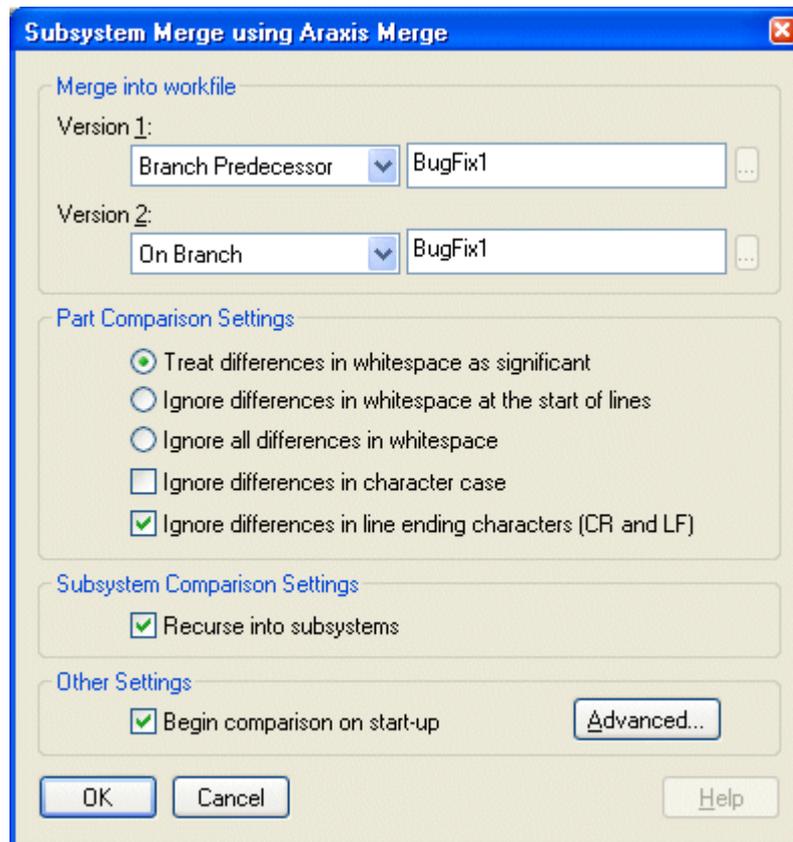
For 3 way directory comparison, simply select 3 directories in the file browser and select **File | Merge** or right click and select **Merge**. This will invoke Araxis Merge with the 3 selected directories.

Note that any copying of files between directories within Araxis will not be reflected in **AllChange** parts.

In order to merge a subsystem, select a single subsystem in the parts browser and select **Part | Merge** or right click and select **Merge**. This will merge the differences between the two specified versions into the current workfile for each component in the subsystem.

A dialog is displayed allowing the selection of the versions within the subsystem that are to be merged. Versions may be any of:

- Default - the [default version](#)
- Top - the top version on the main line
- Registered - the [registered version](#)
- In Baseline - specify the baseline and the version used will be the version in the selected baseline if it exists in the baseline otherwise the top version on the main line
- On Branch - specify the branch name and the version used will be the top version on the branch if it exists otherwise the top version on the main line
- Branch Predecessor - specify a branch name and the version used will be the version from which the branch was started (i.e. the predecessor of the first version on the branch) if the branch exists otherwise the top version on the main line
- At Date - specify a date and the version used will be the highest version that existed at the specified date. Note that if a version was created *on* the date specified this will be the version used.



Version 1: This should be the version from which changes have been made, i.e. this is the common ancestor.

Version2: This should be the version containing the changes which are to be merged with the workfile

Part Comparison Settings: These determine what changes should be treated as significant

Subsystem Comparison Settings: Determines whether to recurse into subsystems in the subsystem tree automatically

Other Settings: Provide other Araxis merge settings

For each component in the subsystem the specified versions are made available to Araxis merge to allow the changes between **Version1** and **Version2** to be merged into the workfile if the part is checked out for edit to the current workspace or into the default version if it is not. In the example above for each component the changes between the version from which the BugFix1 branch was created and the top version on the BugFix1 branch may be merged into the workfile.

In order to save the results of any merge the parts must be checked out for edit to the current workspace, the file name to save to is passed on to Araxis as a part of the URL.

To all intents and purposes once in Araxis subsystem merge looks just like a 3 way directory comparison.

Note that you may not perform file Copy operations when within Araxis.

Checking In

About Checking In

When you have finished with a work file/wish to place a file under **AllChange**'s control it should be *checked in* using the **Check In** function.

The **Check In**function in **AllChange** may be used in a variety of different situations:

- When you have finished with a checked out component version then the part should be *checked in*, regardless of whether it is checked out for read or edit purposes. If it is checked out for edit then the

modified file contents will be placed under **AllChange** control as the new version of the component. If it is checked out for read purposed then the workfile needs to be removed. In both cases the check-out log needs to be removed so that the part is no longer logged as checked out.

- When files are to be placed under **AllChange** control as new parts they may be *checked in*.
- When new versions of files have been created without first checking them out (e.g. a new release of a third party application) and they are to be placed under **AllChange** control they may be *checked in*

Any combination of the above may be performed in a single check in on selected files or a directory as **Check In** will determine what action is appropriate for each file/part encountered.

In order to perform a **Check In** either select the files/directory to be checked in, or select the parts to be checked in. **Check In** is available from the **Check In** option from the **Part** or **File** menu or context menu or from the toolbar. Files may also be dragged from the file browser to the part browser to invoke a check in. Files/Directories may also be checked in from Windows Explorer if the interface for this is installed on the workstation, files may be dragged from Explorer and dropped onto AllChange, or the **Check In** option on the **AllChange** context menu may be used.

Parts may also be checked in by double clicking on a Check In status in the [Cycle View](#).

When checking in files/directories the **Check In** 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.

Note that unless the **Show Obsolete Items** option is selected, then any components or subsystems encountered which are obsolete will be skipped (unless the obsolete item was explicitly selected).

Following a check in, if a component has a class which has the **Keep checked out** attribute, then if it is checked in from edit it will automatically be checked out again read only.

Check In Actions

Check In presents dialogs requesting information as appropriate to the files/ directories/ parts submitted. 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 and corresponding action are shown in the table below:

Criteria	Action
Workfile does not exist for part and part is not checked out	No action — i.e. ignore
Workfile exists as a part which is checked out for edit	Check in part
Workfile exists as a part which is checked out read only	Check in part
Workfile exists as a part which is not checked out	Create new version of part
Workfile exists but there is no corresponding component part	Add new component
Directory exists but there is no corresponding subsystem part	Add new subsystem

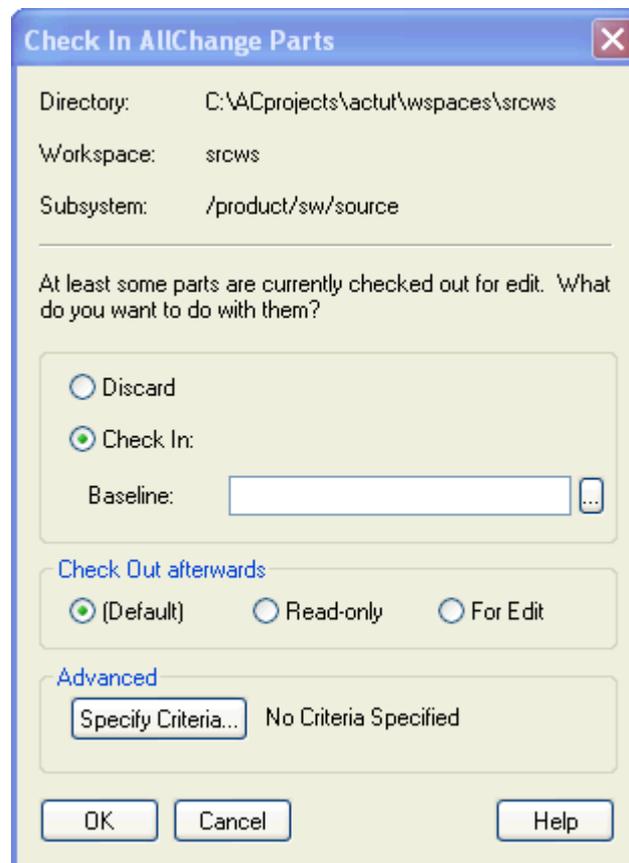
In all cases if a part is checked in which has a class with the **Keep checked out** attribute, then it will automatically be checked back out again read only on completion of the check in action.

Checking In Parts Checked out Read-Only

When a component which is checked out for read only purposes is checked in the check-out log and the workfile will be removed

Checking In Parts Checked out for Edit

When a component which is checked out for edit purposes is checked in the **Check In AllChange Parts** dialog is presented.



If any changes made are to be stored in **AllChange** then **Check In** should be selected. Any changes to the workfile will be stored.

If the part was checked out with pessimistic locking the *reserved/place-holder* version in the parts database will be changed to a full version (i.e. the **NoVersion** flag will be removed) (if no changes were actually made the **AllChange** system may have been configured to detect this and remove the redundant version).

If the part 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.

The *workfile* will also be removed, as will the check-out log record. If the part has a life-cycle then its status will be progressed to the *checked in* status as defined in the life-cycle definition. For example in the supplied **source cycle** the status will be progressed to *Complete* and then on to *Idle*

If a **Baseline** is selected then the new version checked in will be added to the baseline. The baseline must already exist and must be a *release* baseline.

If it is desired to recheck out the part after the check in then the appropriate option may be selected for **Check Out afterwards**

(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

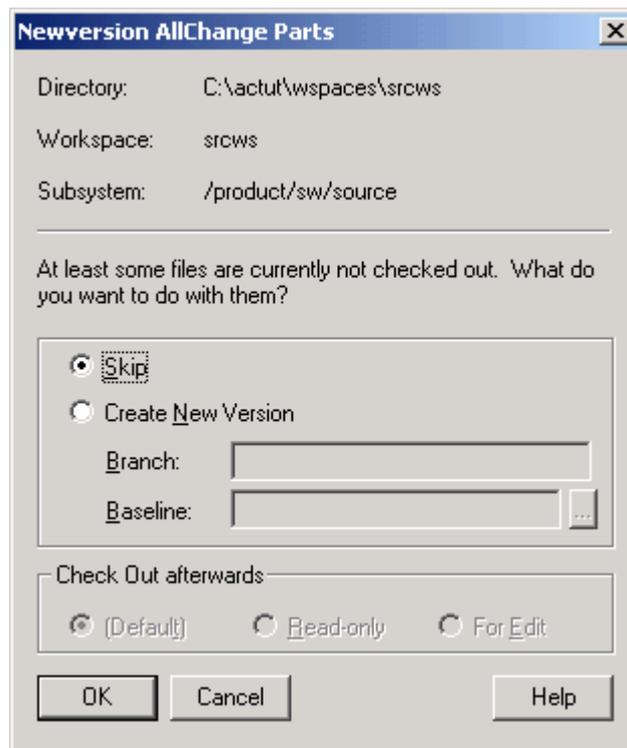
Any version arbitrary fields which have been defined as *compulsory* will be prompted for. In the **out-of-the-box** configuration parts which are of a class which do not **Require a CR** in order to be checked out for edit have version arbitrary field specified as a compulsory **Comment**. This means that a comment will be prompted for on check in.

If the component was checked out for edit purposes and this was a mistake, or you do not wish to retain any changes made then select the **Discard** option and the reserved version and the workfile will both be discarded and the check-out log removed.

If desired [Check In Criteria](#) may be specified which must be matched for the part to be checked in.

Checking In Parts Not Checked out

If a part is checked in which is *not* checked out (for either read only or edit purposes), but a workfile exists for the corresponding part, the **Newversion AllChange Parts** dialog is presented.



If you do not want to import workfiles as new versions of parts then select **Skip**.

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

If **Branch** is specified then the new version will be version 001 on the new branch.

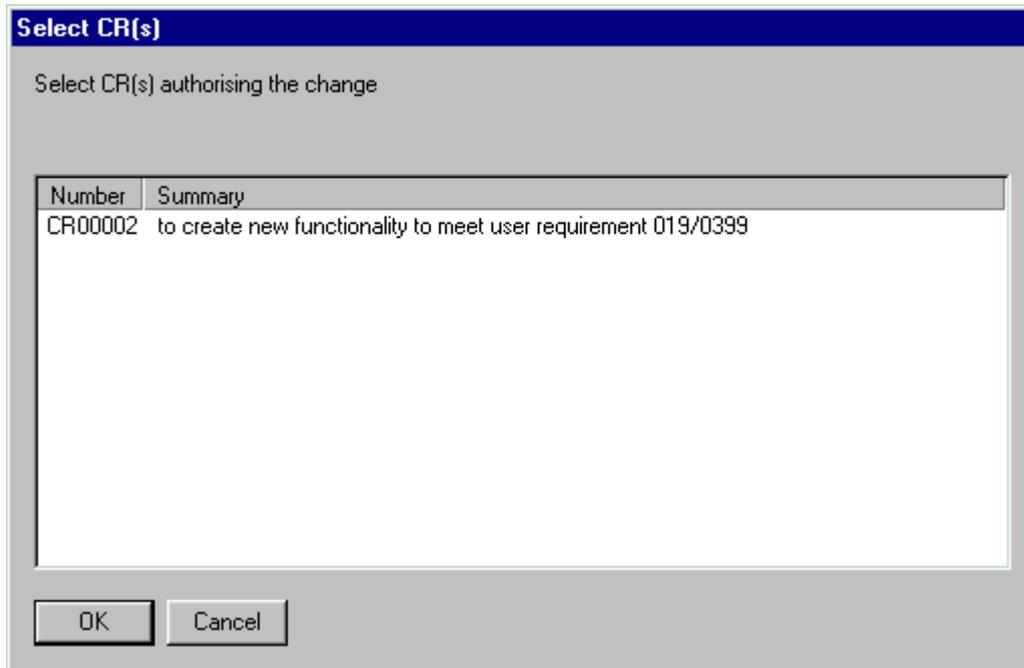
If a **Baseline** is selected then the new version will be added to the baseline. The baseline must already exist and must be a *release* baseline.

If it is desired to recheck out the part after the check in then the appropriate option may be selected for **Check Out afterwards**

(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

If the part is of a class which is defined as *requiring a CR* (e.g. **controlled_source** in the supplied configuration), then, in the same way as when checking out for edit (Checking Out for Making a Change), you will be prompted for the CR(s) which authorise the change to the part.



You must select a valid CR for authorising the new version to be created.

Any version arbitrary fields which have been defined as *compulsory* will also be prompted for.

Checking In New Files/ Directories

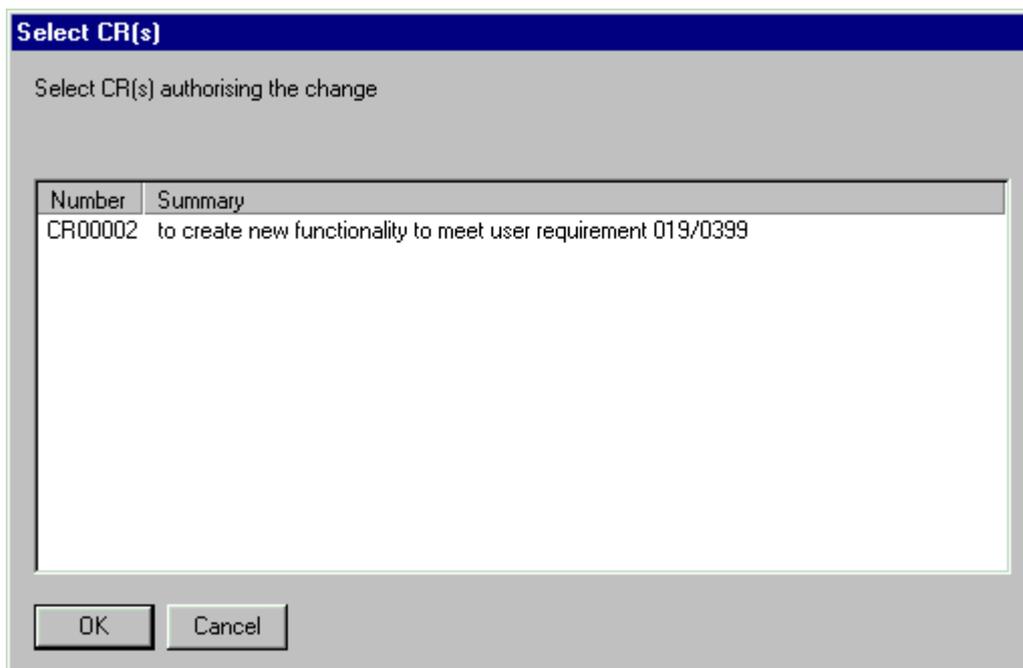
If a file/ directory is checked in for which there is no corresponding part the **Add AllChange Components/ Subsystems** dialog is presented.



If you do not want new parts to be created with the contents of the file as the first version then select **Skip**.

If **Create New Component/ Subsystem** is selected then a new part will be created. For components the contents of the workfile will be taken as the first version.

Select the **Class** that the new component/ subsystem should have. For components if the class is defined as *requiring a CR* (e.g. **controlled_source** in the supplied configuration), then, in the same way as when checking out for edit (Checking Out for Making a Change), you will be prompted for the CR(s) which authorise the new part to be created.



You must select a valid CR for authorising the new part to be created.

Any arbitrary fields which have been defined as *compulsory* will also be prompted for.

If a **Baseline** is selected then version 001 of the new component will be added to the baseline. The baseline must already exist and must be a *release* baseline.

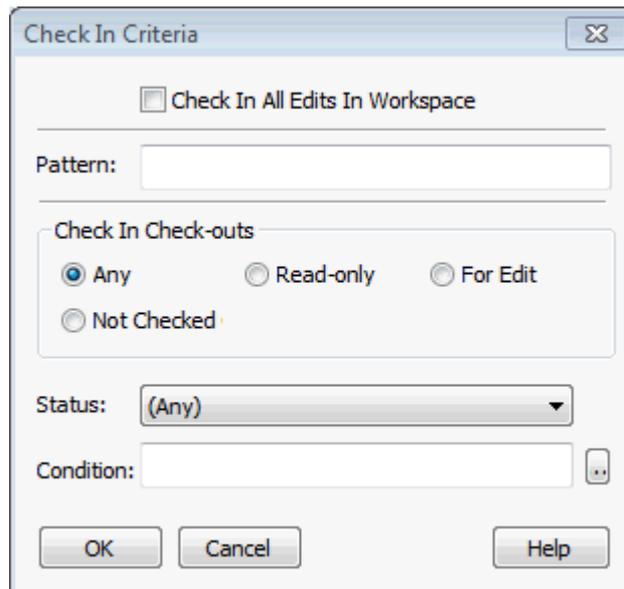
If it is desired to recheck out the part after the check in then the appropriate option may be selected for **Check Out afterwards**

- (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

Check In Criteria

On check in of parts checked out, **Check In Criteria** may be specified. Each part must match these criteria in order to be checked in, otherwise they are skipped.

The Check In Criteria is available from the **Specify Criteria** button on the Check In dialog.



If **Check In all Edits In Workspace** is selected, all other check in criteria are disabled. This will cause all parts checked out for edit to the current workspace to be checked in.

If a **Pattern** is specified then the part/file name being checked in must match the pattern specified.

Check In Check-outs selections allow specification of whether only parts checked out for edit, only parts checked out read only, all checked out parts or parts not checked out should be checked in.

If a **Status** is selected the version checked out must be in that status for the part to be checked in.

The **Condition** allows any arbitrary criteria to be specified to further define which parts are to be checked in.

Check In Parts

If parts are selected these will be checked in from the *current* workspace, you must therefore ensure that you are attached to the workspace to which the parts are checked out.

If a **Subsystem** is specified then the subsystem tree will be descended with every component encountered being submitted for check in.

Check In of Files/ Directory

If files are selected **Check In** 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 no files are selected and the current 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 checked in may then be selected.

If a **Directory** is specified then you will be prompted for patterns that files in the directory tree must match in order to be checked in. In determining which files to check in **AllChange** will include *hidden* and directories only if the Explorer setting to **Show hidden files** is selected.

Simply leave this blank if all files are to be checked in or enter a space separated list of patterns using * to denote *0 or more of any character*, e.g. *.c will only match files with the extension .c.

The directory tree will be descended with every file encountered being submitted for check in, provided it matches any pattern specified. Files which have a .ac or .bt extension 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.

Using Life-cycles to Check In and Out

When parts are checked in and out of **AllChange** this may be performed in one of 2 ways:

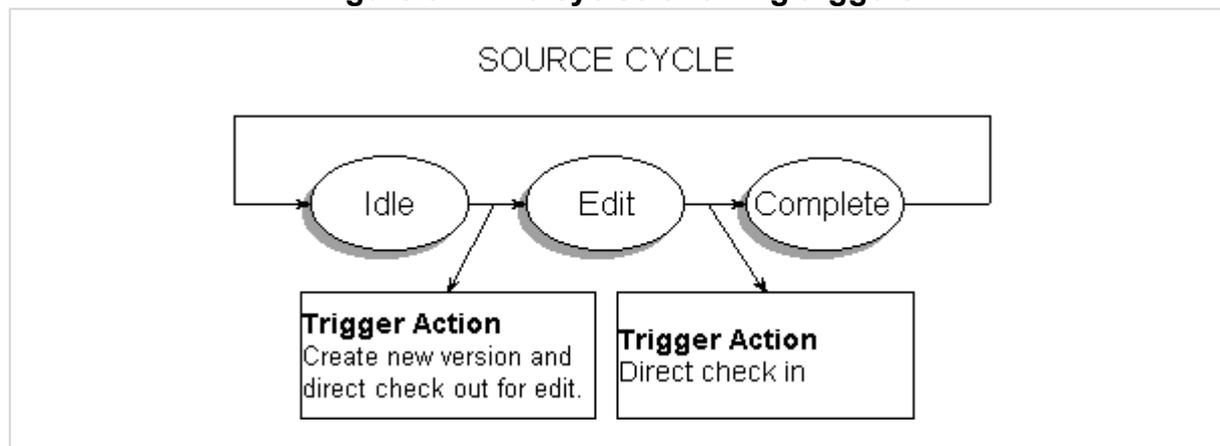
1. Direct check in and out
2. Life-cycle driven check in and out

If a part has a life-cycle associated with it (which is determined by the parts class) then the check in and out is driven by the life-cycle, otherwise it is performed directly.

The actions in the life-cycle of the part may trigger the *check out* and *check in* of parts on entry to certain life-cycle states, these are known as action *triggers*.

In [Figure 6.7](#) a possible source code life-cycle is shown

Figure 6.7: Life-cycles showing triggers

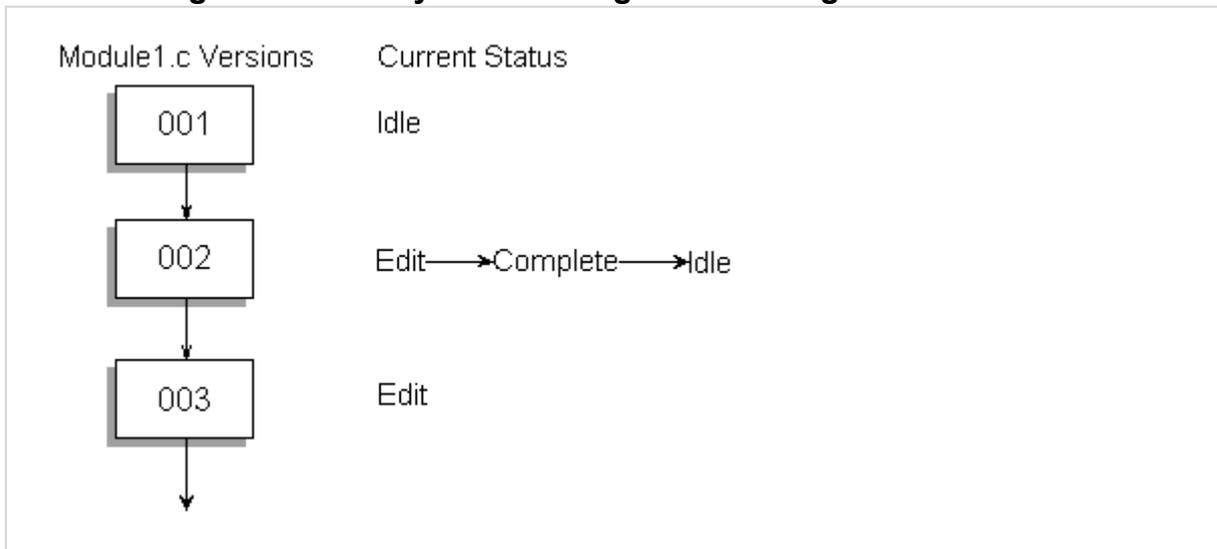


Upon entering the status of **Edit** for a version, a new version will be created and the original version **direct checked out** for edit. On entering the status **Complete**, the version is **direct checked in** from edit. i.e. the trigger in the life-cycle causes the same actions as a **direct** check in/ check out.

The status in a life-cycle which triggers a check in action is known as the **Check In status** and the status which triggers a check out action is known as the **Check Out status**.

Note that each version progresses through its own life-cycle and has a current status as shown in the Part Viewer. [Figure 6.8](#) shows how the status changes on each version as it is checked in and out using its life-cycle.

Figure 6.8: Life-cycles showing status changes on versions



When a **Check In** is performed this will progress the status of the version to its **Check In status** and when a **Check Out** is performed it will progress the status to its **Check Out status**.

There may be some life-cycles which have multiple check in or check out statuses which are valid at any particular point in the life-cycle - if this is the case then you will be prompted for which status to progress to.

The status of a version should not be changed to a Check In Status or a Check Out Status directly using the **Status** dialog as this does not provide the full Check In/Out functionality. However, a CheckIn/Out may be performed from the [Cycle View](#) by double clicking on a CheckIn or CheckOut status.

Checking In and Out from Microsoft Development Environments

AllChange provides facilities to conform to the Microsoft Common Source Code Control Interface (MCSCCI).

To make use of this facility you need to select it in the Workstation Installation, see the *AllChange Administrator Manual*.

This means that files may be checked in and out of **AllChange** directly from the Visual Basic or Visual C++ environments, or any other environment which supports the MCSCCI, e.g. MS Access, Artisan Real Time Studio, I-Logix Rhapsody and PowerBuilder.

When an operation is performed in the IDE which requires a communication with **AllChange**, if it is not yet connected it will connect to an already running **AllChange** if there is one or start a new one if not. Note that you should not attempt to perform operations in the **AllChange** to which the IDE is connected directly and via the IDE at the same time. If you wish to interact with **AllChange** directly, then you should run another instance of ACE for this purpose.

The facilities provided, the dialogs etc presented and the actions these implement are defined by the interface provider and not Intasoft.

The default implementation for the following IDE menu options is:

Get	performs an AllChange Check Out of a (newer) version of a file for read purposes.
CheckOut	performs an AllChange Check Out prompting for any additional information required according to the AllChange configuration. By default this will check-out the default version, however this may be tailored to prompt for the required version by the AllChange administrator, see the All-Change Administrator Manual .
CheckIn	performs an AllChangeCheck In prompting for any additional information required according to the AllChange configuration. It is then checked out read only.
Undo Check Out	performs an AllChange Check In using the Discard option to discard any changes and then checks it out read only.
Show History	runs a report using the <code>sccihist.rep</code> format file

The integration is designed to prompt for necessary information as it goes, rather than from dialogs presented in the IDE before an operation commences. This is largely because **AllChange** can store a lot more information than the SCCI interface allows for, especially when different classes of component are involved. **AllChange** informs the IDE not to request such information in its dialogs. However, we have found that some IDEs (e.g. VC++) ignore the fact that we have told it that it should not allow a comment to be typed into its Add/ CheckIn dialog; accordingly, if the IDE does pass a comment to **AllChange** for Add/CheckIn and a file's corresponding component's class defines a *compulsory* arbitrary field named `Comment` for its versions then the comment passed in is used for this instead of prompting the user. We would still recommend leaving this blank if provided in the IDE and allowing **AllChange** to prompt.

When a new file is **Added** to the IDE you will be offered to add the file to source code control. There is also an **Add Files to AllChange** option from **Add-Ins | AllChange**. Adding a file causes an **AllChange Check In** which will recognise that a new part needs to be added and prompt for additional information as necessary (e.g. class of the new part, CRs authorising the addition and compulsory arbitrary fields). The new component will be added to the subsystem corresponding to the directory containing the file.

If a file is Removed from the IDE, an attempt will be made to remove the corresponding component if it is not present in any baselines. If any versions are present in any baselines then a list of such baselines will be offered and all versions of the component will be removed from any baselines selected. The component will not be removed in this case.

At the commencement of each **Get**, **CheckIn** and **CheckOut**, ACE attaches to a workspace containing the IDE's current working directory as specified by the IDE.

By default all files for a project for Visual C++ or Visual Basic are maintained *checked out* for read only purposes unless they are checked out for edit. This check-outs log in **AllChange** may therefore be examined to see what versions of each file are in use. If, however, it is not desired to maintain the check-outs log, this may be configured by the **AllChange** administrator, see the **AllChange Administrator Manual**.

Checking In and Out from Eclipse

Facilities are provided which allow for the checking in and out of files directly from the Eclipse IDE.

In order to make use of the Eclipse integration it must have been properly installed, see the Administrator Manual for details.

In order to use the integration a workspace must be defined for the Eclipse workspace and the project must have been shared. For general information about the interface see [Eclipse Interface](#).

The check in/out functions available are the same as for the [Microsoft Source Code Control interface](#). These functions are available from the **Team** menu in Eclipse.

Checking In and Out from MS Word

Word Macros are supplied with **AllChange** that allow files to be checked in and out of **AllChange** directly from the MS Word user interface.

To make use of this facility you need to select it in the Workstation Installation (note that this will also install the MS Excel interface), see the **AllChange Administrator Manual**.

When the macros have been installed you should then find an **AllChange** pulldown menu has been added to the Word menu bar. Also a toolbar has been added to the available toolbars and may be selected from the **View | Toolbars** dialog.

Check In and **Check Out** are available from the **AllChange** menu and invoke the normal **AllChange Check In** and **Check Out** functions, see [Checking In](#) and [Checking Out](#). These functions will connect to an already running **AllChange** or start up a new session if there is no current session in progress.

Check In will check in the current active document, closing it prior to the check in. If there is no current active document you will be prompted to open a document using the standard open file dialog. On closing the document, if there are changes which have not been saved, the file is saved. If the document is not a new one and was checked out of **AllChange**, then any changes must be saved into the same workspace directory that the file checked out to.

Check Out will open a new document for the part checked out.

The **Check In** and **Check Out** will prompt for additional information as required by the rules defined in the **AllChange** configuration. e.g. if the class of part required a CR to authorise a check out or an add then this will be applied and the CR's will be prompted for in the normal way

Checking In and Out from MS Excel

An MS Excel Add-In is supplied with **AllChange** that allows spreadsheets to be checked in and out of **AllChange** directly from the MS Excel user interface.

To make use of this facility you need to select the Word Interface option in the Workstation Installation and then the add-in must be enabled using **Tools | Add-Ins...**, where **AllChange Integration** should be offered, see the **AllChange Administrator Manual**.

When the add-in has been installed an **AllChange** cascade menu should be available from **Tools** menu.

Check In Workbook and **Check Out Workbook** are available from the **AllChange** menu and invoke the normal **AllChange Check In** and **Check Out** functions, see [Checking In](#) and [Checking Out](#). These functions will connect to an already running **AllChange** or start up a new session if there is no current session in progress.

Check In will check in the current active workbook, closing it prior to the check in. If there is no current active workbook you will be prompted to open one using the standard open file dialog. On closing the workbook, if there are changes which have not been saved, the file is saved. If the workbook is not a new one and was checked out of **AllChange**, then any changes must be saved into the same workspace directory that the file was checked out to.

Check Out will open a workbook for the part checked out.

The **Check In** and **Check Out** will prompt for additional information as required by the rules defined in the **AllChange** configuration. e.g. if the class of part required a CR to authorise a check out or an add then this will be applied and the CR's will be prompted for in the normal way

Checking In and Out from Real Time Studio

An [MCSCCI](#) interface has been implemented between Artisan Real Time Studio and **AllChange**.

To make use of this facility you need to select MCSCCI in the Workstation Installation, see the **AllChange Administrator Manual**.

The interface is fully functional except that it does not support the **Add Package From Another Model** facility in Real Time Studio's Configuration Management menu.

In order to use AllChange with Real Time Studio you need to [create a workspace](#) which points to the directory containing your models, this is known as the Artisan Models Neighbourhood directory.

Documentation as to the functionality of the Configuration Management facilities may be found in the Real Time Studio Users Guide.

In order to perform the same functionality as **Add Package From Another Model**:

- use **AllChange** to check out the desired Package to a folder
- then use the **From Directory** command in Real Time Studio to add the Package from that folder.

It is recommended that ACE is loaded before loading Real Time Studio. If it is not, then ACE could load and unload several times on loading a model into Real Time Studio. Running ACE first will eliminate this problem.

Checking In and Out from Dreamweaver

Facilities are provided which allow the files for a site to be checked into and out of **AllChange** directly from the Macromedia Dreamweaver interface.

To make use of this facility it needs to be enabled for your Dreamweaver installation, see the AllChange Administrator Manual for details.

Once enabled **AllChange** may be selected as the source control system for a site in Dreamweaver's Site Definition **Remote Info** page. Ensure that it is selected.

In order to use the interface a workspace for each site must also be defined - the workspace directory should be the site's **Local Root Folder**.

The first time an attempt is made to connect to **AllChange** for a Dreamweaver site the user will be prompted to choose the **AllChange** workspace for that site; picking an unsuitable workspace will make the interface appear to function incorrectly. The workspace for the site is then stored in the registry and automatically reused in future connections.

The menu items available from the Dreamweaver interface are defined by Dreamweaver, not by **AllChange**. Although **Get** and **Put** are supported, we recommend using **Check Out**, **Check In** and **Undo Check Out** in preference, especially in a multi-user collaborative environment.

Checking In and Out from TestDirector

Facilities are provided to allow **AllChange** to control the versions of HP TestDirector (TD) files. It requires TestDirector 7.5 and AllChange 6.0, or later.

The interface (menu items, functionality etc.) is defined solely by TestDirector, not **AllChange**; **AllChange** cannot change the menu items or what they do, nor can it display any messages to the user or in any way interact with him.

See the AllChange Administrator manual for details of installing the TD integration.

The top level subsystem to be used to hold the files for each TD project must be created before the integration may be used (e.g. `/TestDirector/TestProj`) – use [Part | Add](#) – and a workspace for this part must also be available.

You may now create projects in TD SiteAdmin with Version Control support and/or **Enable VC** for projects, and allow end-users to use them. You need to enable Version Control in TestDirector for each project (in TD SiteAdmin, **Site Config** tab, set Attribute **VC** to have value **Y**. You will now have an **Enable VC** icon back on the **Projects** tab. You must enable VC on existing or newly created projects to use the integration). Note that whenever you **Enable VC** on a TD project TD does not pass us a username; we log onto **AllChange** as user **admin** in this case, so this must be a valid user with a suitable workspace. In all other cases, the user is logged onto **AllChange** under the name he logs onto TD as.

End-users will then see a **Versioning** item in the TD menu bar and on right click when on the **Test Plan** tab inside TD, and will be able to use VC.

Checking In and Out from Rhapsody

Facilities are provided which allow the files for a project to be checked into and out of **AllChange** directly from the Rhapsody interface.

For versions of Rhapsody before 3.0 see below for details. For Rhapsody 3.0 the MCSCCI is supported and the standard facilities are supported, see [Checking In and Out from Microsoft Development Environments](#).

Interface for versions of Rhapsody earlier than 3.0

The facilities provided are determined by Rhapsody and implement only a very simple interface to **AllChange** which does not make use of the full power of **AllChange**. These *may* be tailored to provide more powerful facilities.

Ensure that the subsystem associated with your workspace for your Rhapsody work exists.

When invoking a CM operation from Rhapsody, if **AllChange** is not currently running it will start it for you. This will be a minimised ACE which is for use with the interface and not for interacting with. It may be closed by selecting **Close** on the right click menu on the entry on the system bar in Windows.

Having opened your project (.rpy file) ensure that the **Project Properties** have a **Subject of Configuration Management** and a **Meta Class of AllChange**.

The items which may be checked in and out may be accessed from **File | Configuration Items**.

CheckIn to Archive may be used to check items in which have already been **Added to the Archive**. Selecting **Locked** checks the file back out for edit after checkin, otherwise the file is checked back out read-only. All other controls are ignored (not relevant to **AllChange**). This performs an **AllChange Return** command.

CheckOut from Archive checks items out of **AllChange**. Selecting **Locked** checks out for edit, otherwise the file is checked out read-only. All other controls ignored (not relevant to **AllChange**). This performs an **AllChange Issue** command.

Note that you must have **Connected to the Archive** before you may check in or out.

As the Rhapsody CM definition requires no interaction (i.e. no dialogs from) with the CM tool provider, no classes are used. Parts may not have the **Require CR** attribute and there must not be any compulsory arbitrary fields. There is also no interface to baselines.

See [Rhapsody Interface](#) for details of the Rhapsody interface.

Checking In and Out from DOORS

Facilities are provided which allow DOORS modules to be checked into and out of **AllChange** either directly from the DOORS interface or from ACE. Note that **Link** modules may not be checked in and out.

When a module is checked out for read only purposes then the module will be set as read only for that user and will only be writable when it is checked out for edit.

In order to make use of the DOORS integration it must have been properly installed and set up — see the **AllChange Administrator Manual** for details.

All DOORS projects should be stored under a single subsystem within **AllChange** (e.g. /DOORS) and each DOORS project is represented as a subsystem in **AllChange** with the same name as the DOORS project. The modules of the project will be stored within the project subsystem as components with the module name with a .zip extension. The class of the **AllChange** components which hold the DOORS modules is `doors` by default unless otherwise specified in the **DOORSPartClass** configuration option. Only components with this class will be treated as DOORS modules.

Whenever checking in and out DOORS modules if DOORS or ACE are not currently running they will be started automatically. If the required DOORS project is not open, then **AllChange** will prompt you for your DOORS user name and password and will open the project. If the project is already open, but **AllChange** does not yet know your DOORS password for that project then you will be prompted for the password only.

All checking in and out of DOORS components takes place in a workspace called `DOORSWS` unless otherwise specified in the **DOORSWorkspace** configuration option.

From within ACE for components of the DOORS class:

Check In will check in a doors module.

Check Out will check out a doors module.

From within DOORS Check In and Out are available from an **AllChange** menu on the main DOORS window the Module windows (Check In only):

Check Out

This will bring up an **AllChange** parts browser, from which you can select the modules you wish to check out

Check In

If selected from the **AllChange** menu on an open Module then the current module is checked in. If selected from the **AllChange** menu in the main view then this will bring up a window with a list of modules to select from. Any modules selected will be checked in. Note that if adding a new module to **AllChange** be sure to select the doors class for the new components.

Checking In and Out from Explorer

Facilities are provided which allow files to be checked in or out of **AllChange** directly from Explorer.

The integration is implemented as a Shell extension which adds Entries to Explorer's context menu (the menu which pops up when you right-click on a file or directory) allowing files to be checked in or out of **AllChange**.

To make use of this facility you need to select it in the Workstation Installation, see the **AllChange Administrator Manual**.

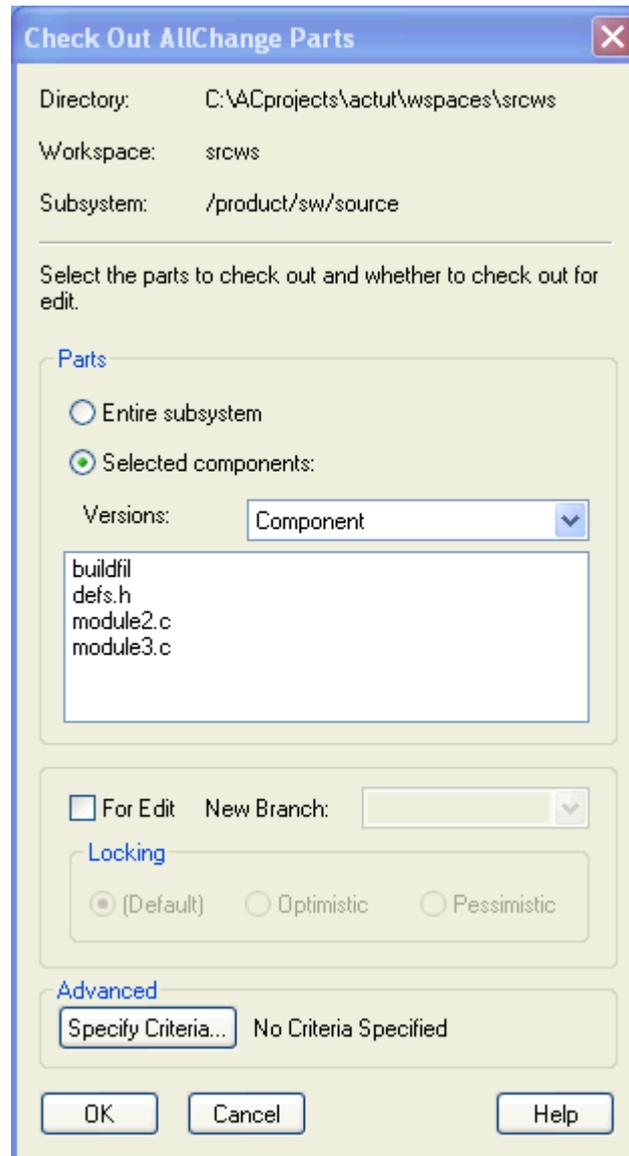
In order to check files *into* **AllChange** select one or more files or a directory in Explorer and right-click. Then select the **AllChange | Check In** entry which appears on the context menu. (Alternatively drag the files from explorer and drop them onto AllChange).

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.

This will then perform a standard **AllChange Check In** function prompting for various further information as necessary, see [Checking In](#). At the end of the operation the files which have been checked in will be removed from the directories.

In order to check files *out* from **AllChange** select, in Explorer, one directory to check out into — which must be a workspace directory — or select the files that you wish to check out if they are already checked out read only and right-click. Then select the **AllChange | Check Out...** entry which appears on the context menu. This will perform a standard **AllChange Check Out** function prompting for further information as required, see [Checking Out](#). At the end of the operation the workfile(s) will appear in the directory selected.



If ACE is not currently running it will be started (minimised) to service the requests. You cannot interact with this ACE; run a separate ACE if you want full **AllChange** interaction for other purposes.

Note that *hidden* and directories are only checked in if the Explorer setting to **Show hidden files** is selected.

Checking In and Out from SolidWorks

Facilities are provided to allow SolidWorks files to be checked in and out of **AllChange** from within SolidWorks.

In order to make use of the SolidWorks integration it must have been properly installed and set up — see the [AllChange Administrator Manual](#) for details of setting up to use the integration.

Check In and Out operations are available from an **AllChange** tab on the task pane and command manager and from an **AllChange** menu on the menu bar or from context menus.

Files to be checked in an out must reside in an **AllChange** workspace.

For details of the integration see [SolidWorks Interface](#).

Finding out What is Checked out

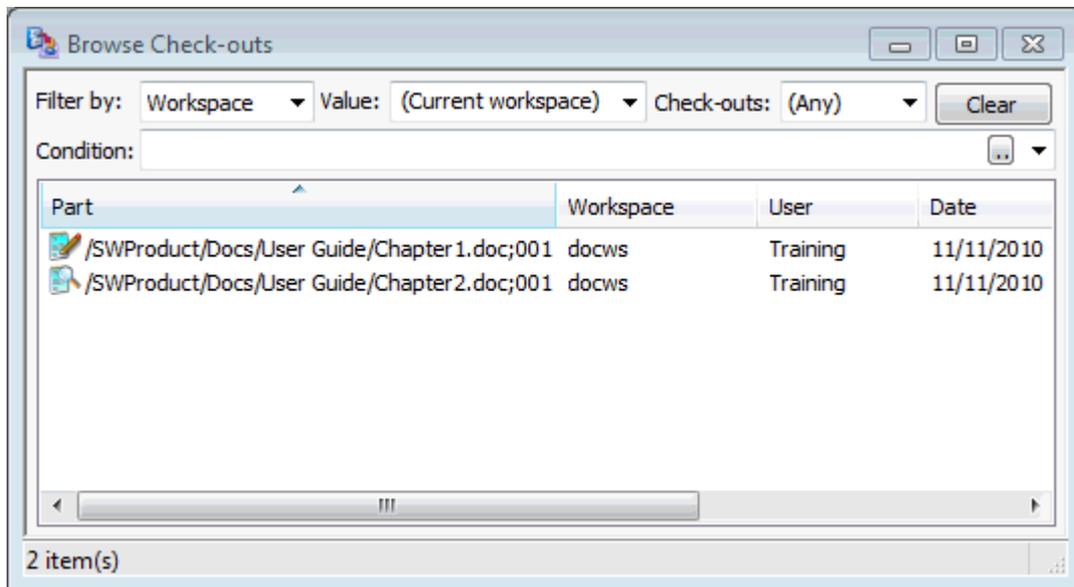
There are two primary methods of determining what is currently checked out.

Note that the term check-out may have been mapped to site specific nomenclature (e.g. issues) in which case all references to check-outs will show using the mapped nomenclature instead.

1. Use the part browser with various **Filters** and/or the **Show Check-out icon in parts list** option
2. Use the check-outs browser

This section covers the check-outs browser, for details of the part browser see [Browsing the Parts Hierarchy](#).

The check-outs browser is available from **Workspace | Browse Check-outs**.



The check-outs browser includes the name of each part that is checked out, the workspace to which it is checked out and the user to whom it is checked out; the icon will also show whether it is checked out for edit or read only.

Icon	Description
	Checked out for read only
	Checked out for edit

The list shows what versions are currently checked out.

The list may be filtered to restrict what is shown according to the **Filter By** selected and the **Value** specified. If a value is selected then only those check-out records which match the value selected for the field filtered by will be shown. The value may be **(Any)** if all items are to be shown but indexed by the selected filter. The **Value** may be terminated with * to denote *any other characters* and the values matched will be those which *start* with the text specified. If no * is used then an exact match is required.

For some filters (e.g. **Workspace**) other special values may be available such as **(Current Workspace)**, this is interpreted as the workspace that is currently attached to if any.

For example, selecting **Part** as the **Filter by** and then entering a value of `/project/mysub/*` will show all check-outs of anything below `/project/mysub/*`.

The **Check-outs** drop-list allows the items listed to be limited to those checked out for edit, checked out read-only, or not limited.

The entries shown, as well as matching the specified **Filter** and **Value** must also match any condition specified in the **Cond** edit control. This may be any valid ACCELexpression.

To find out what parts are checked out to you (for example), select the **Filter by** value of **User** and a **Value** of your user name and only parts checked out to you will be shown. Note that if there are a large number of check-out records then applying the above **Filter** is *much* more efficient than specifying a **Condition** of "User Equal to *username*".

Double clicking on an item in the check-outs browser will show the details of the check-out in the check-out viewer. This will show additional details such as the new version created which may not be displayed in the browser columns.

The check-out browser supports a [folding view](#).

Finding out What has Changed

There are two aspects of finding out what has changed:

1. What lines/contents of files have changed
2. Which files have been changed

Information about both of these is available from **AllChange**.

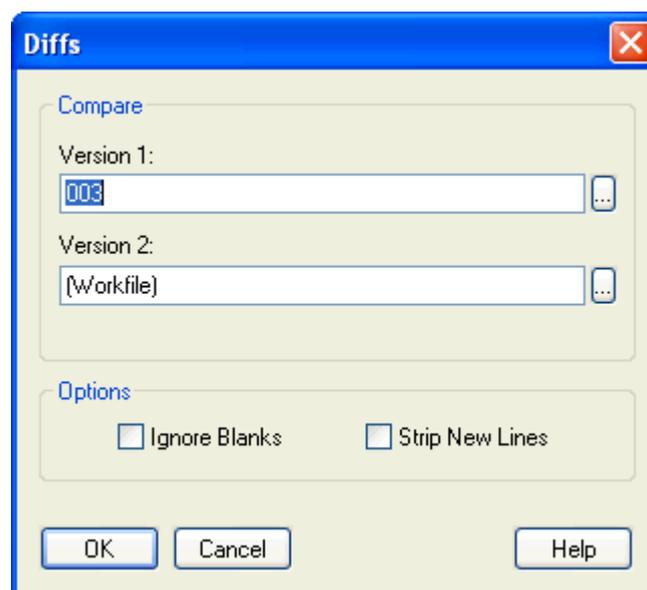
What File Contents have Changed

If you wish to view the changes that have been made between two part versions or 2 different parts, select **Part | Diffs** or **File | Diffs**, or from the context menus, or from the **AllChange | Differences** from Explorer or from **AllChange | Compare Versions** from MS Word.

By default if comparing Word documents, then MS Word will be used to perform the comparison and display the results. If comparing binary files then a simple comparison for the same or different will be performed. Otherwise the supplied **VisDiffs** tool is used. This may be used effectively for comparing text files.

This default behaviour may be modified to use the file comparison tool of your choice. Use the [External Tools](#) option in ACE.

In all cases if a single part or file (which corresponds to a part) is selected, a dialog will be displayed allowing you to select which versions of the part you wish to compare.



If 2 parts are selected a similar dialog is displayed allowing you to select the version of each part that you wish to compare.

If attached to an FTP workspace and comparing against a workfile then the file will first be transferred to a temporary location before invoking the comparison tool.

If you wish to compare 2 parts in different subsystems then select **Part | Diffs** with no parts selected, this will display a dialog allowing you to select the 2 parts you wish to compare.

The two versions of interest should be entered and the **Visdiffs** (or third party) tool will be invoked to show the differences. If the part is considered to be a Word document (file has a .doc extension) Word is used to display the differences.

Various options are available for **Visdiffs**:

Ignore Blanks

If selected then any white space will not be considered when comparing the two versions

Strip New Lines

If selected then blank lines (i.e. lines consisting of just a newline) are ignored.

These options are ignored if a third party differencing application is used. If Araxis is the differencing application then the Araxis comparison options dialog is displayed.

If two files are selected then the differencing application will be invoked for the two files immediately.

See the Differencing for full details of the **Visdiffs** tool.

From within MS Word, select the **AllChange | Compare Versions** option when a part which has been checked out from **AllChange** is the currently opened document. **Version 2** in the **Diffs** dialog will default to **Workfile** representing the open document. A new document will be opened in word displaying the differences between the versions selected.

Which Files have been Changed

Check Check-outs Changes

In order to determine which files that are checked out to the current workspace have been changed, use **Workspace | Check Check-outs Changes**. This will compare the current file contents for each part checked out to the workspace to see if any changes have been made. If a change has been made then this will be reported in the output window. If no change has been made nothing will be reported.

This command is, by default, an **AllChange** administrator only function.

Comparing Subsystems

If using the Araxis differencing application and the full integration with Araxis is enabled, then comparison of versions of parts in a subsystem may be performed.

A subsystem may be compared with:

- different versions within the same subsystem allowing, for example, a branch to be compared with the main line, or the top version with a baseline
- versions within another subsystem
- a directory or workspace allowing, for example, comparison to see which parts have been changed from the checked out (default) version. Note that the directory may be a remote directory accessible via FTP using a plugin available from Araxis free of charge.

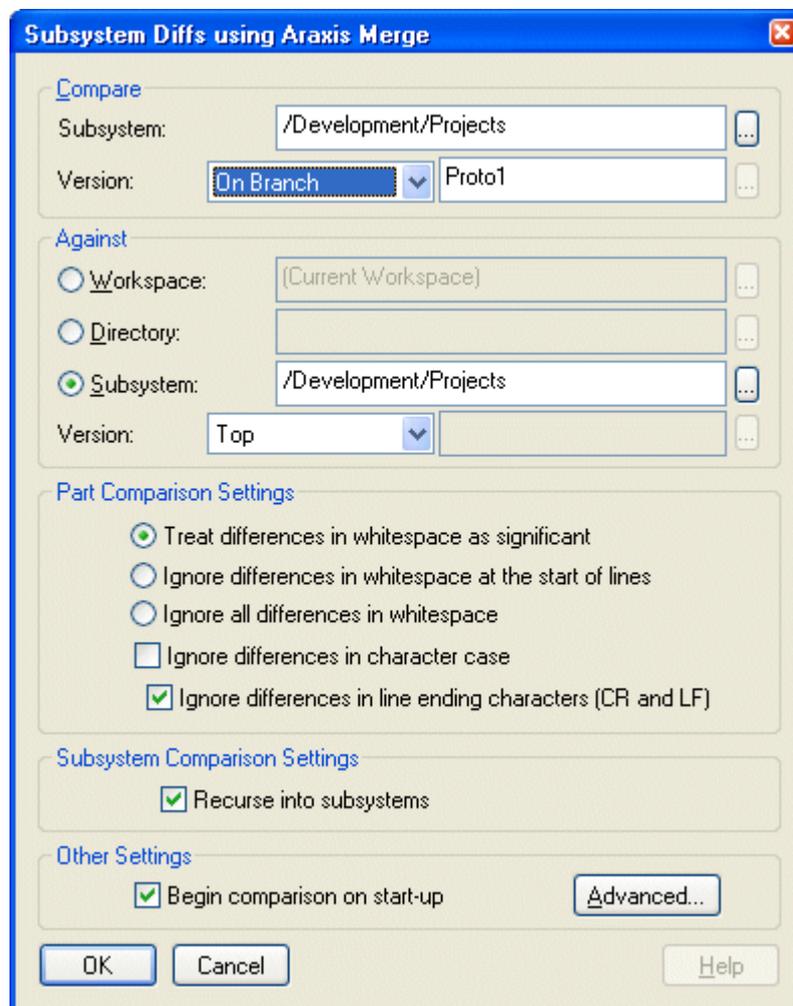
The versions that may be compared are:

- Default - the [default version](#)
- Top - the top version on the main line
- Registered - the [registered version](#)

- In Baseline - specify the baseline, and the version used will be the version in the selected baseline if it exists in the baseline otherwise the top version on the main line
- On Branch - specify the branch name, and the version used will be the top version on the branch if it exists otherwise the top version on the main line
- Branch Predecessor - specify the branch name, and the version used will be the version from which that branch originated, i.e. the predecessor of version 001 of the branch
- At Date - specify a date, and the version used will be the highest version that existed at the specified date. Note that if a version was created *on* the date specified this will be the version used.

To perform a subsystem comparison, select one or two subsystems and select **Part | Diffs**, or right click and select **Diffs**.

The dialog below will be shown allowing the selection of which versions are to be compared.



Compare:

Subsystem: specifies the subsystem to be compared

Version: specifies the version of the components within the subsystem to be compared.

Against: specifies what to compare the subsystem versions specified above with.

Workspace: specifies to compare against the files in a specified workspace

Directory: specifies to compare against the files in a specified directory

Subsystem: specifies to compare against the components in a subsystem. This may be the same subsystem as specified for the **Compare**, but selecting different versions, or may be a different subsystem altogether.

Version: specifies the version of components within the subsystem to compare against.

Part Comparison Settings: These determine what changes should be treated as significant

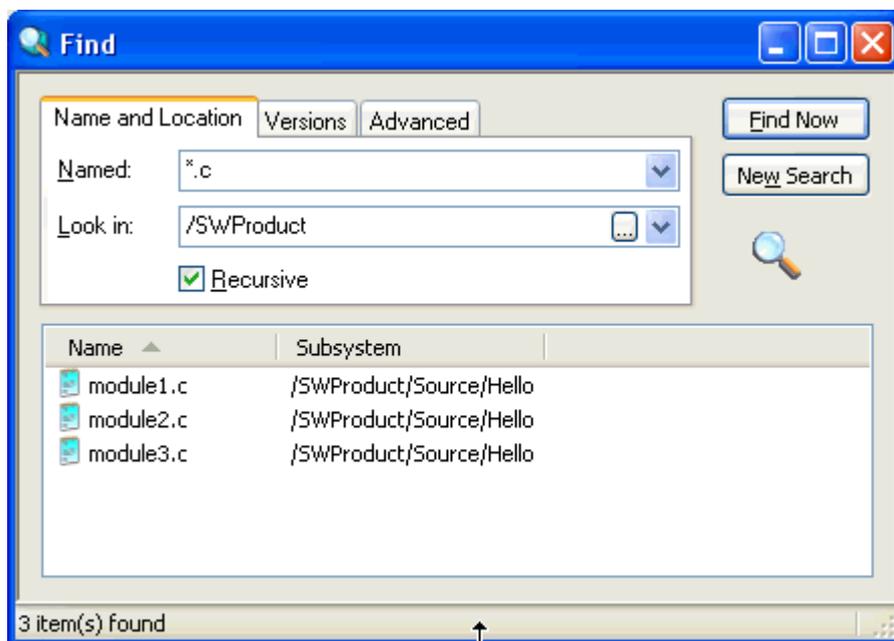
Subsystem Comparison Settings: Determines whether to recurse into subsystems in the subsystem tree automatically

Other Settings: Provide other Araxis merge settings

Finding Parts

The **Find** utility will allow you to search the parts database for all parts which have a particular name, match a specified condition or contain a reference to a particular string.

Find is available from the **Part** menu or from the Tools toolbar.



Specify in **Look In** the part of the tree that you wish to search, if left blank then the whole parts tree is searched (as if you had entered '/').

If you wish to search for parts of a particular name then enter the name (or the start of the name) in **Name**. All parts starting with this will be shown.

If **Recursive** is not selected then only the **Look In** subsystem will be searched and not the whole tree.

The **Versions** tab allows you to specify whether component versions are to be searched or just components. If only part versions are of interest then also select **Only list versions in results**.

The **Advanced** tab allows an arbitrary **Condition** to be specified that the parts found must match and/or a string which the parts must contain. If a string containing is specified then if a case sensitive search is required select **Match Case**. If a **Containing** value is specified then any **Versions** selection will be ignored, as the containing search looks for the string in any version of a part.

The **Find Now** button will start the search and all matching parts will be displayed in the list in the Find window.

Any parts found may be selected and used to perform any operation relevant to parts.

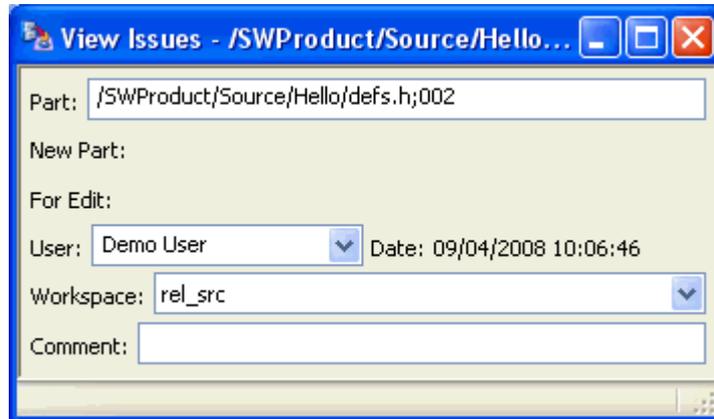
New Search clears all search criteria specified and allows a new search to be specified.

Moving Parts from one Workspace to Another

It is sometimes desirable to move a part (or file) from one workspace to another: perhaps in order for the part to be reviewed before checking it back in, or to pass the work on to another person.

This may be achieved directly using the Check-out viewer window or indirectly via the life-cycle.

In order to move a part directly, double click on the part in the Check-outs browser window and the details of the check-out record will be shown:



Select the new workspace required and then **Update**: the workspace that the part is checked out to will be changed and the corresponding file will be moved to the new workspace directory.

Updating Workspaces

It may be desirable to update the parts that are checked out to a workspace such that the workspace is maintained with certain parts always checked out to it.

The requirement may be to just update to the latest version of what is already checked out to it, or to ensure that the latest versions of all the parts which are in a subsystem are checked out to it, or perhaps the parts that are in a baseline is what is required.

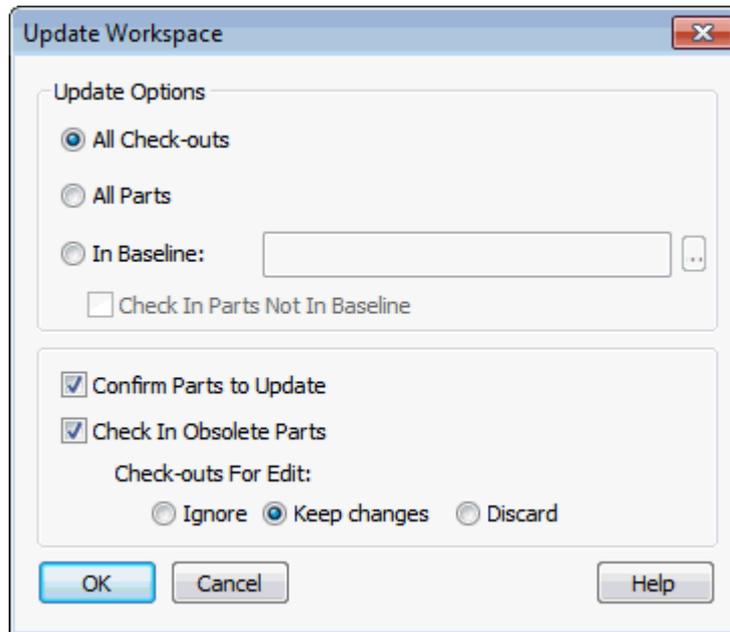
When using the [optimistic locking](#) model for parallel development it is also necessary to ensure that the workspace is updated with changes that others have made where there have been no changes in the workspace and to merge changes that others have made with changes made in the workspace before performing a check in.

AllChange supports all of these requirements.

There are three methods of updating the parts checked out to a workspace:

1. Automatically on attaching to a workspace
2. **Workspace | Update Workspace**
3. **Part | Update**, or **File | Update**

The first two of these invoke the same site modifiable function which will allow you to specify your criteria for updating the whole workspace.



The **Update Options** determine what parts are to be examined for update purposes and which versions of these are required in the workspace.

Any parts examined which are currently checked out to the workspace will be updated as follows:

- Checked out Read Only: the workspace will be updated to the required version
- Checked out for Edit with Pessimistic Locking: the workspace will not be changed
- Checked out for Edit with Optimistic Locking: if the required version is newer than the one which was checked out then:
 - if the workfile has not been changed since it was checked out it is updated to the newer version
 - if the workfile has been changed since it was checked out then a merge with the changes is offered.

All Check-outs updates the current workspace to the latest registered version of all parts currently checked out to the workspace.

All Parts updates the workspace to the current registered version of all of the parts in the workspace sub-system regardless of whether they are currently checked out or not. This means that for those that are checked out they will be updated to the latest registered version and any which are not currently checked out will be checked out.

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.

If **Check In Parts Not In Baseline** is selected this will force any part which is currently checked 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.

If **Confirm Parts to Update** is selected you will be presented with a list of the parts that **AllChange** proposes to update and allows you to make a final selection from these.

If **Check In Obsolete Parts** is 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 **If 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.

If automatic workspace updating is enabled, then when you attach to a workspace you will be prompted to update it. See the **AllChange Administrator Manual** for details of enabling auto workspace update.

The third option (**Update**) allows individually selected parts or files to be updated rather than the entire workspace. If a subsystem is selected then all components within the subsystem tree will be updated. If a directory is selected then the subsystem corresponding to the directory is used. Any parts which are checked out read only will be updated to the latest (registered) version. Any parts issued for edit with optimistic locking are updated accordingly as described above.

Extracting Versions Unlogged to a Directory

The normal method of extracting versions to a directory is to [check out](#) for read-only purposes. However, sometimes it is desirable to extract versions to a directory without creating a record of the extraction in the check-outs database.

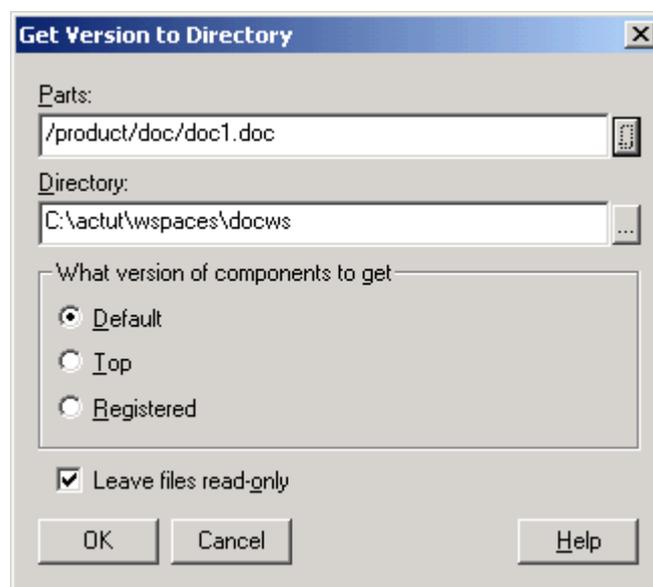
Two functions are provided for this purpose:

- **Get Version to Directory**
- **Get Baseline to Directory**

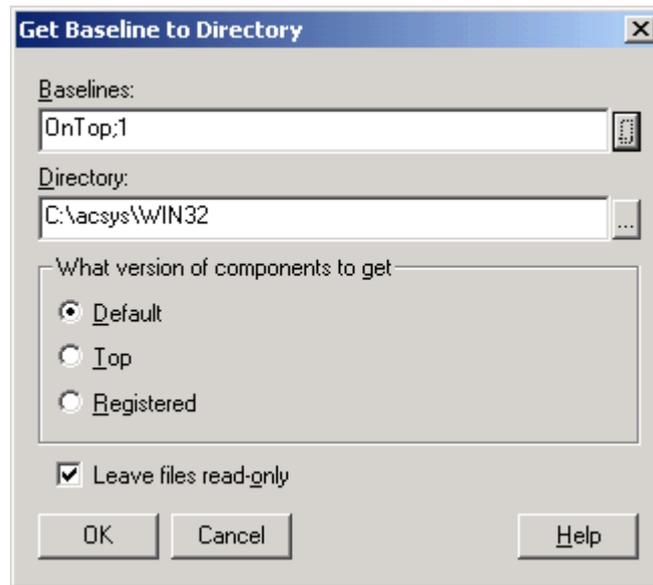
These allow parts to be extracted to any directory without being logged as a check-out and therefore without requiring the directory to be a workspace directory.

The files that are extracted are read-only and should be deleted when no longer required.

Get Version to Directory allows specified versions or a complete subsystem to be extracted to a named directory. If a version is not explicitly specified then the **Default, Registered or Top** may be selected.



Get Baseline to Directory allows a specified baseline to be extracted to a named directory. If the baseline is a release baseline then the baselined versions are extracted; if the baseline is a design baseline then **Default, Registered or Top** may be specified to define which versions of components are extracted. A Meta baseline may be specified in which case all baseline within the meta baseline are extracted (recursively).



Any components or versions which are obsolete will *not* be extracted unless the Show Obsolete flag is set.

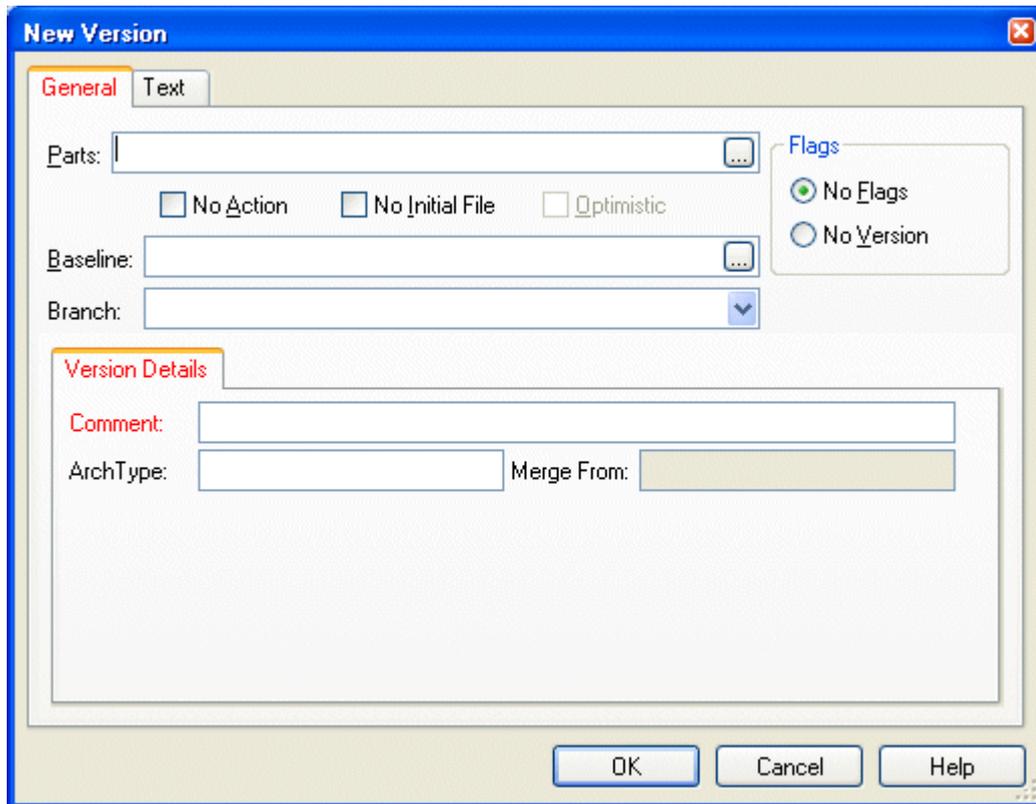
Importing New Versions

If a new version of a file has been created and it is desired to put this under **AllChange**'s control, but the part has not been checked out for edit purposes, then **New Version** may be used to create a new version of the part, storing the current file contents (in the current workspace) as the contents of that version.

New Version is available from **Part | New Version**, or by the use of **Check In**.

Check In will offer to use **New Version** for any files/parts encountered for which there is a part which it is not checked out and for which there is a workfile in the current workspace. You will be prompted for what action to take i.e. create a new version of the part using **New Version** or skip the file/ part, see [Checking In](#)

If using **New Version** directly then the version that is to be predecessor of the new version should be selected using the part browser or viewer, this will be entered as the **Parts** on the **New Version** dialog.



A **Branch** name should be specified if the new version is to be on a new branch.

The **Comment** may be used to specify a comment to be passed on to the underlying version control tool.

No Flags should be selected so that the new version does not have any special flags set on it.

If the **Varb1...Varb10** fields are in use then these may be specified in the dialog and will be associated with the new version created.

The new version will be created in the **AllChange** database and the file will be put under version control. The workfile will be removed as with **Check In**.

Managing Files on a Remote Machine

AllChange has the capability to allow files which reside on a remote machine to be managed transparently by the use of FTP.

A typical scenario might be **AllChange** running on a Windows platform and the files to be controlled reside on a Unix or VAX-VMS machine. The FTP integration permits the copying of files between different machines across TCP/ IP.

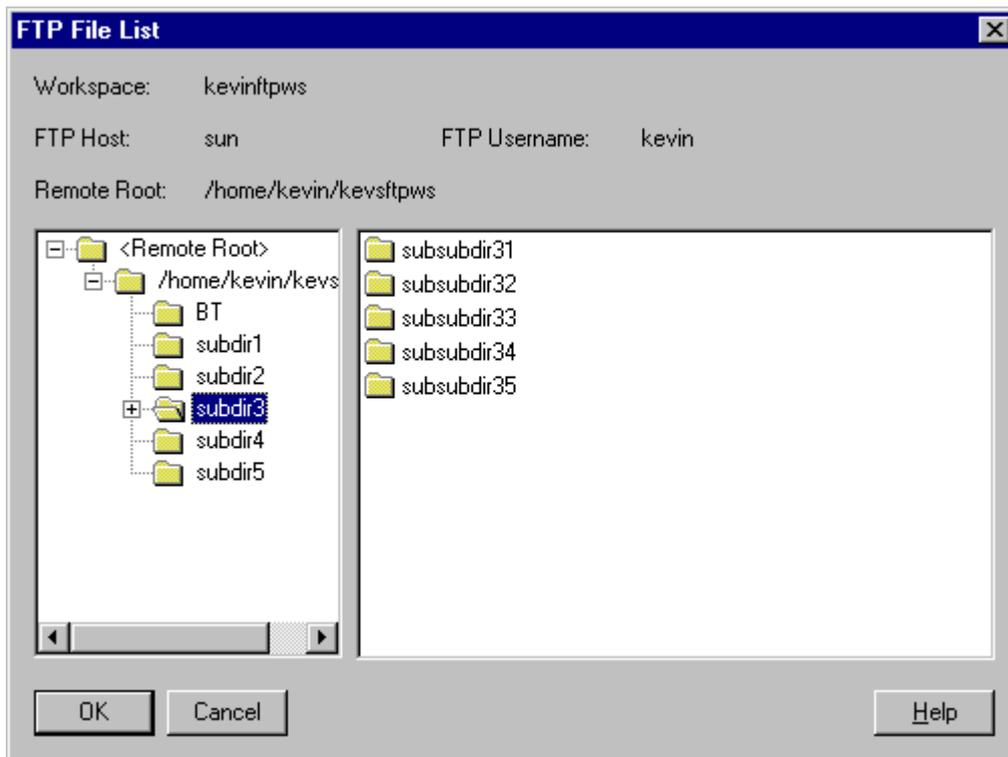
Certain workspaces and/or pools are declared as FTP workspaces/pools. When you use such a workspace/pool you may need to log onto the remote machine. Thereafter whenever that workspace/pool is used the required workfile will be transferred to/from the remote machine as necessary.

For example, whenever a part is checked out to an FTP workspace, the resulting workfile will be transferred to an appropriate location on the remote machine. Whenever a part is checked in from edit, the workfile is first transferred from the remote machine to the Windows machine and then stored under version control in the normal way.

All of this FTP activity occurs transparently to the user of **AllChange**.

When attached to an FTP workspace **Check In** will be available on the **File Menu** without the necessity to select any files to be checked in in the file browser. If **Check In** is invoked in this situation then an **FTP**

File Browser will be presented allowing the files/ and/ or directories to be checked in to be selected from the remote machine.



Similarly if promoting from an FTP workspace then a remote file browser will allow selection of the files to be promoted and these will be transferred to the local workspace directory before copying to the appropriate pool directory. If the pool is an FTP pool then the files will be FTPd to the appropriate remote pool directory.

It is possible to mix FTP and non-FTP workspaces with FTP and non-FTP pools in any combination.

For full details see the **AllChange** Administrator Manual.

Managing Web Development

ACCEL code has been provided to help support the management of Web development (i.e. the development of HTML pages which are to be viewed with a Web browser) from within **AllChange**. The scenario catered for is where HTML pages are stored as individual files which are managed as **AllChange** parts. Files are edited and tested on a local network from within **AllChange** in the same manner as any other type of file. However, special action must be taken when these files are released into a "live" Web environment. This process is called *deployment*.

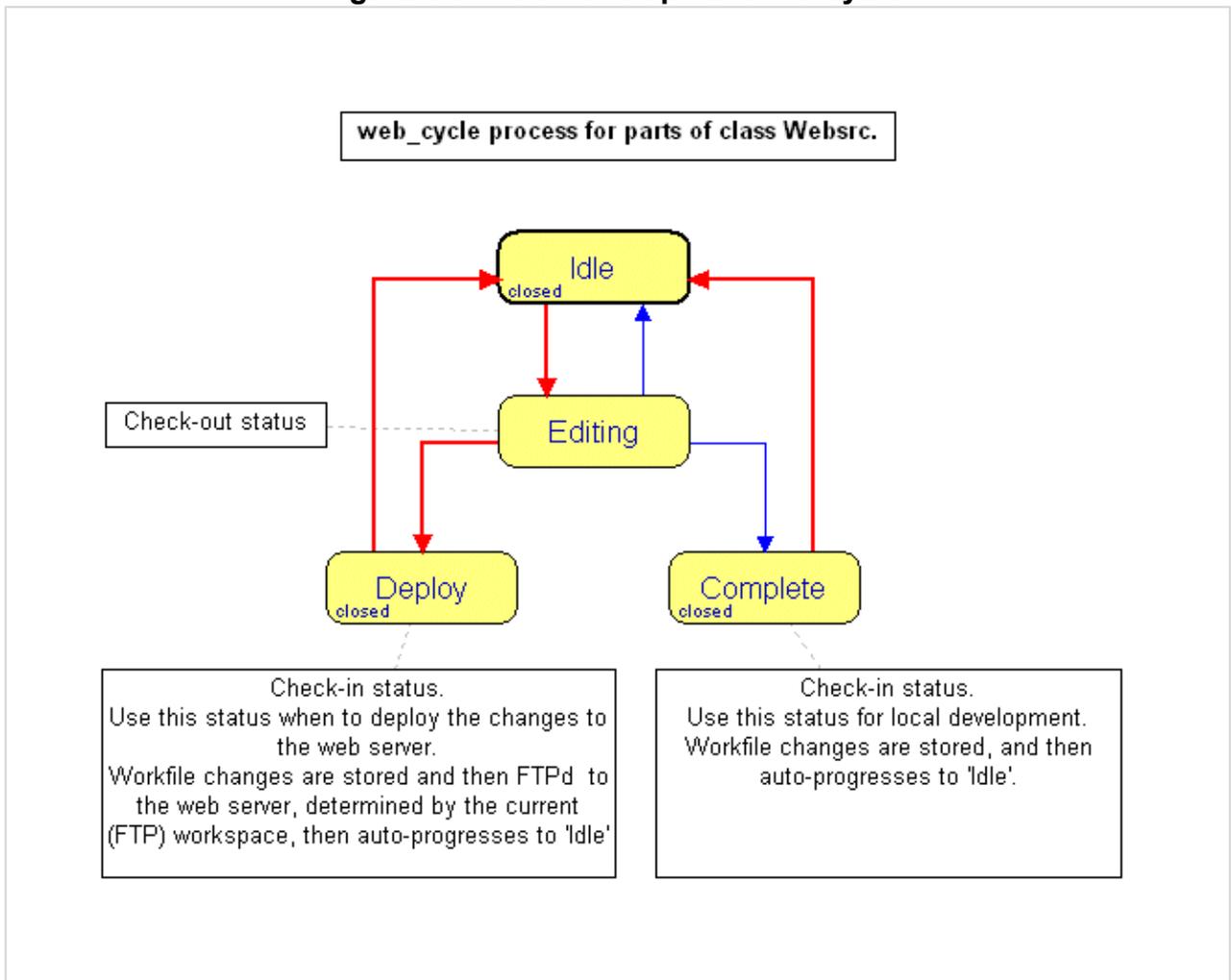
This support consists of two elements:

- Facilities are provided to allow the substitution of any links found which refer to the local development area — which is the state they need to be in during the editing phase — by links which refer to the "live" area — which is the state they need to be in for end users.
- Typically these HTML files must be copied to a remote Web server machine which may not be directly accessible from the local system. This is accomplished using the FTP functionality described in the **AllChange** Administrator Manual.

Web deployment will need to have been configured for use by your **AllChange** administrator.

The supplied `websrc` class and `web_cycle` life-cycle are intended to illustrate a possible life-cycle for Web source. This is illustrated in [figure 6.9](#).

Figure 6.9: Web Development Life-cycle



When a version enters the `Deploy` status any links in the workfile defined as requiring substitution will be substituted for the appropriate link on the web server. The file will then be deployed across FTP to the remote machine.

See the ***AllChange*** Administrator Manual for details on configuring ***AllChange*** to use these facilities.

File Operations

About File Operations

Most operations within **AllChange** which affect files are actually performed on *parts* (these are effectively files which have been declared to **AllChange**), see Sections [Creating and Managing Parts](#) and [Checking Files In and Out of AllChange](#).

These operations cover functions such as:

- Declaring new files to **AllChange**
- Checking files in and out of **AllChange**
- Editing files
- Deleting files

These operations may usually be performed on either the parts using the **Part** menu, or on files or directories which relate to the parts using options on the **File** menu in ACE, or from the various tool interfaces such as Windows Explorer, MS Word, Visual Studio.

Some operations, however, work directly on files which may or may not have corresponding parts. These include:

- Promoting files from workspaces to shared areas called pools
- Building derived files
- Releasing files
- Editing files
- Comparing files

These operations can be found on the **File** menu in ACE and are classified as **File** operations in the reference section. Some of these may also be available from the Explorer and Word interfaces.

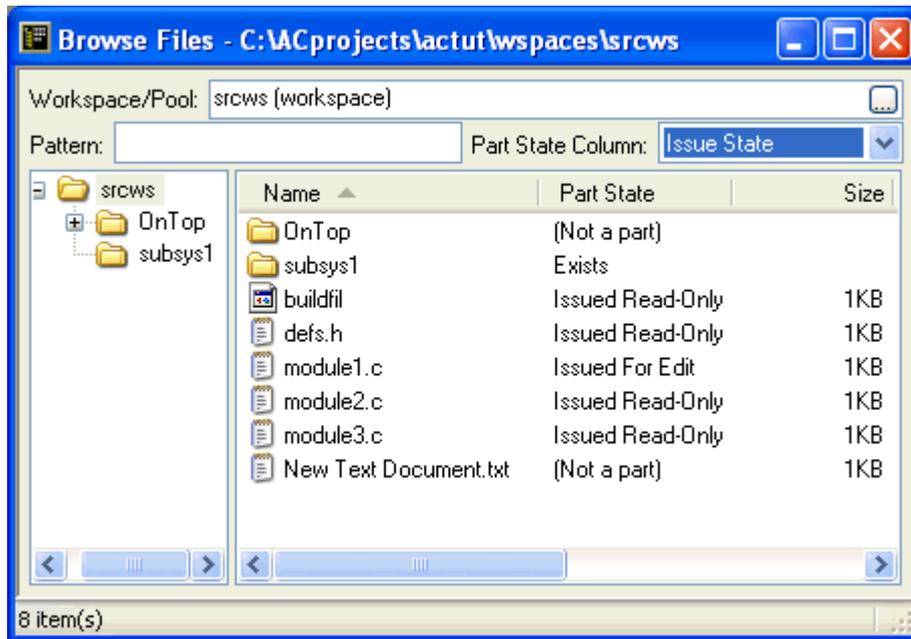
Promoting, building and releasing files are operations which centre around the files in *workspaces* and *pools*.

This chapter discusses the facilities which are available on files and which have not been discussed elsewhere such as in [Checking Files In and Out of AllChange](#).

File Browser

AllChange incorporates a built in file browser (like Windows explorer) to allow you to find and select files that you wish to operate on. Typical file based operations may include:

- Checking files in and out of AllChange
- Promoting files to pools
- Building derived objects from their source



The **Workspace/Pool** allows the selection of a workspace or pool to browse and will *root* the file browser at the top of the selected workspace or pool. This means that you may not access files outside this directory tree.

To view the files in another workspace or pool, simply select the required workspace/pool and the files in this directory tree will be shown and the browser newly *rooted* at this point.

Selection of **(None)** will cause the browser to be unrooted and it will display files in C:\, the entire filing system is now available for browsing (according to network access control) including the entire network.

Pattern may be used to limit the files shown to those matching a specified pattern, e.g. *.c to show only files with a .c suffix.

Part State Column may be used to specify what information is to be shown in the **Part State** column. Possible values are:

(None): no part state information is shown

Existence: shows whether a part exists which corresponds to the file

Check-out State: in addition to existence, displays whether the part is checked out

Changed State: in addition to the check-out state, checks to see if the workfile has changed

It should be noted that, depending on the selection, the operation required to determine the state can be quite expensive, and may slow down refreshing of the file browser. Selecting '(None)' does not require any processing. The other options become more expensive as you move from **Existence** to **Changed State**.

The **View** style may be changed using **View | View** in the normal way.

When a workspace is attached to, the file browser will automatically re-root to that workspace.

Editing/Viewing Files

The content of parts or files may be viewed or edited from within ACE. If the part has been checked out then a workfile will reside in the workspace directory to which it was checked out. This may be viewed/edited from outside the **AllChange** system if so desired.

Parts which are checked out may be viewed or edited from within ACE by use of **Part (or File) | Edit**. This, by default, will cause the Explorer **edit** action to be performed which will *edit* the file. Where required there should be a distinction between **edit** and **open**, for example for .bat files.

Parts do not have to be checked out in order to view their contents, **Part| Edit** may be used and if not checked out will extract the requested part version to a temporary file and then perform the **Edit** action as described above. If the part is not checked out, or is checked out for read only purposes, then the file will have read only attributes and should not be altered in your editor.

A file may also be opened after selection in the file browser using **File | Open** which will cause the Explorer **open** action to be performed if there is one.

A file or part may also be *viewed* using **File or Part | View File**. This will allow the user to *view* the selected 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 as described above.

Changing the Default Editor

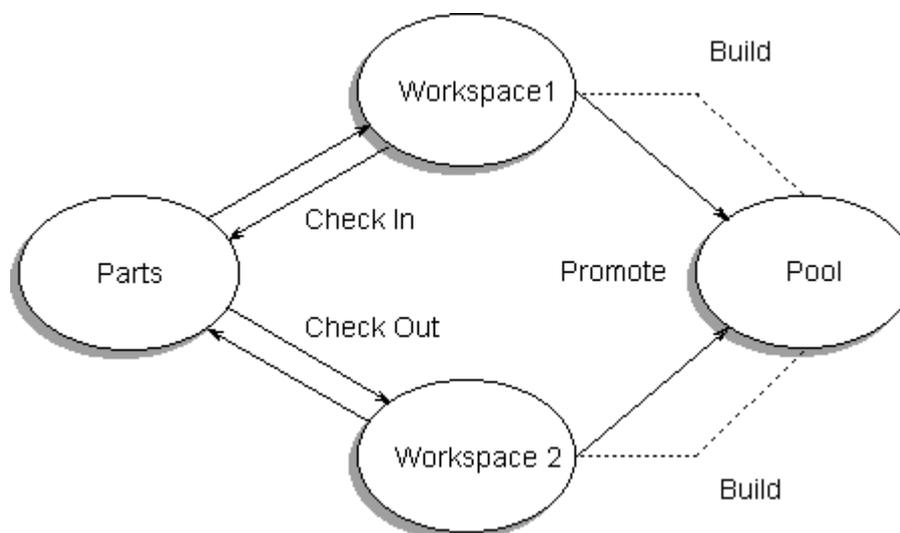
The editor that **AllChange** invokes may have been tailored by your **AllChange** Administrator to depend upon the part suffix, class or some other convention.

As initially supplied the editor invoked under Windows will be that determined by Explorer associations for the **edit** action if there is one, otherwise the default editor will be used.

The default editor may be specified on the **Misc | Options** dialog on the **Misc** tab, otherwise **Notepad** will be used.

Pools

Pools are simply operating system directories that are known to **AllChange**. They are used to hold copies of particular versions of parts and derived objects which are to be shared amongst different users for build purposes. The parts and derived objects are held in pools in the form of operating system files.

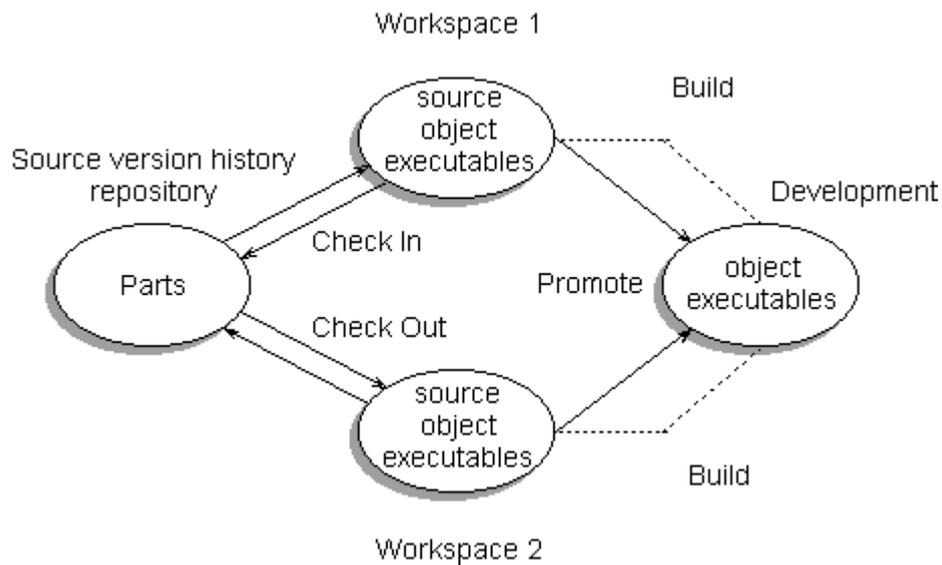


Pools are closely connected to workspaces — see [Workspaces](#). Like workspaces, pools have a *pool-directory* and hold workfiles — files which may be examined with an editor, compiled, printed etc. The files that exist in a pool may be put there by *promoting* a file to a pool. Unlike objects in workspaces, however, objects in pools are intended to be globally accessible, or at least shared by a possibly wide group of users: while changes made to files in a workspace affect only the user(s) of the workspace, a change made to a file in a pool affects all users sharing that pool. Accordingly, an object should only be placed in a pool if it is an acceptable state for all users of the pool.

Pools, like workspaces, may be local or remote: i.e. a pool may represent a directory on the local PC network or may represent a directory on a remote machine which is not directly accessible via the PC net-

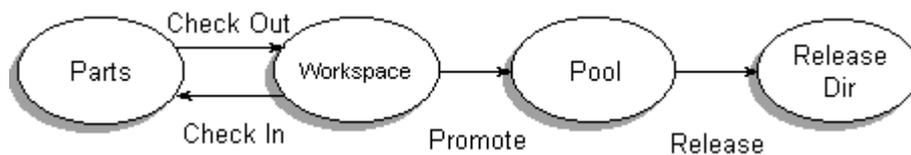
work. Files are placed in remote pools using FTP in the same way as for ftp workspaces, these are referred to as *ftp pools*.

Pools are of particular use in projects involving program source code compilation or a similar task. They provide a convenient repository for derived objects such as executable and object code files as well as compilable source code and include files. These may then be shared among developers and maintainers. Different pools may hold different versions of these files so that work is not restricted to just the current version. The **AllChange build** command is able to combine objects in local workspaces with objects in global pools in a highly efficient and intelligent fashion.



Objects are *promoted* to pools (once they have been built if they are derived objects) from workspaces. The *registrations* facility, described in [Workspace Registrations](#), links pools to workspaces for build purposes. If the workspaces to be used with a pool are *hierarchical* then the pool must also reflect the same directory hierarchy. If a pool is an ftp pool then the files will be transferred to the remote directory using FTP. Files may be promoted to local or ftp pools from local or ftp workspaces.

Objects may also be released from pools into release directories.

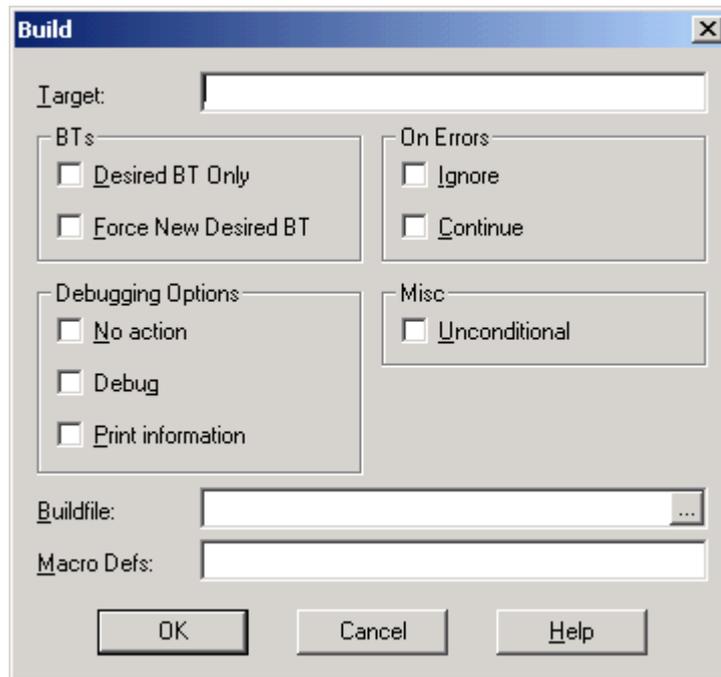


Build

About Build

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.

Build may be invoked from the **File | Build** menu or from the toolbar or context menu when the file browser is the current window. It may also be invoked from the **AllChange** menu from the Explorer context menu if the Explorer interface is installed.

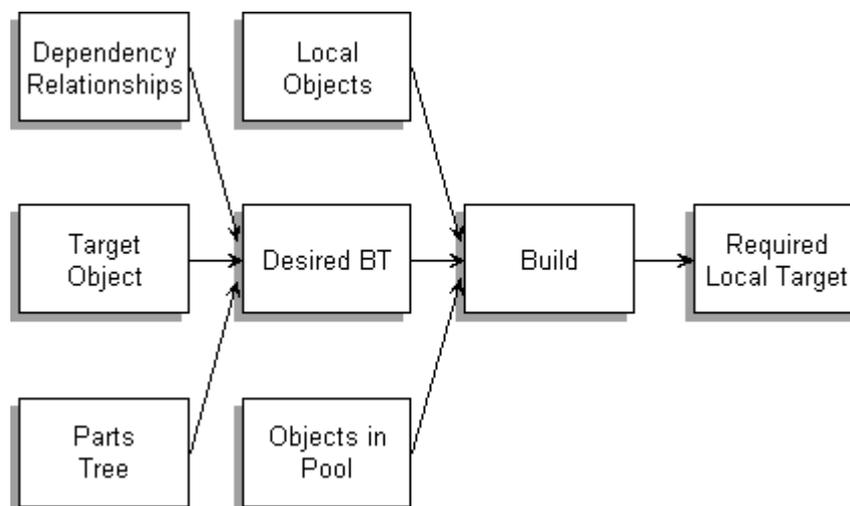


No options need be specified. Any files selected will be copied as the **Target**.

If a **Target** is specified then only that object will be built. If no **Target** is specified then the default **Target** will be built.

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 version defined for the current workspace , for each part involved in the build process (see [Default Versions](#)).

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.



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

The external build utility is similar to a traditional *make* type tool but with a very important extension: it 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 issued for edit and are therefore being changed without changing the version .

Note that for the BT aspects of build to work, build threads must be enabled, see the ***AllChange Administrator Manual***.

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.

The external build utility may be used outside of the **AllChange** system but will re-use the last generated desired BT. The desired BT is only generated when the **build** command in **AllChange** is issued.

In order to perform an **AllChange build** a description of the dependency relationships between the objects to be built must be supplied. This is supplied in the form of a buildfile, this must be present in the directory in which the build is to take place. The buildfile should be maintained under **AllChange** control as a part and should therefore be checked out along with any source code prior to doing a build. Full details of buildfiles and how to construct them may be found in [Building](#).

Full details of the **AllChange Build** facilities may be found in [Building](#).

Build Options

About Build Options

Various options may be specified which modify **Build**'s behaviour.

BT Options

The following options affect the use of build threads:

Desired BT Only

If this is selected, then only the desired BT will be built.

Force New Desired BT

If this is selected, then full regeneration of the desired BT will occur. In the interests of speed the desired BT will normally only be updated if it already exists, however, under certain circumstances a full regeneration may be desirable.

Debugging Options

The following options may be useful to understand the behaviour of the build tool, e.g. how it decides which objects need to be rebuilt:

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 build-file'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.

Error Handling Options

The following options affect how build acts on errors occurring as the result of a translation action (e.g. compilation errors):

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.

Miscellaneous Options**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.

Buildfile

If this option is not used, **Build** will look for a file called `buildfil` in the current directory, expecting it to contain a description of dependencies.

This option may be used to specify an alternative name for the buildfile **Build** is to use.

Macro Defs

This option should specify a (comma-separated) list of macro assignments (see [Build Macros](#)). This allows macros to be set in precisely the same way as in the buildfile or the environment but from the command line.

A typical use is to temporarily override a macro assignment situated in the buildfile — see [Macro Priorities](#). For example, while developing a C program it is inefficient to have optimisation switched on, so the buildfile may well contain the line:

```
CFLAGS =
```

to ensure the optimisation flag is not on. To test out optimisation without altering the buildfile, issue the command:

```
build -mCFLAGS=-O
```

(or equivalent for the particular C compiler). The `-m` preceding command line macro assignments may be omitted if desired, e.g. `build CFLAGS=-O` is acceptable; a macro assignment is distinguished from a filename by the `=` symbol.

Build Threads

About Build Threads

Every part checked out, and every derived object created by **AllChange build**, will have an associated *build thread* (BT) (if **Use build threads** is enabled, see the *AllChange Administrator Manual*).

BTs are used primarily for build purposes, but also provide a very valuable document as to how an object was built and may be included in a release if required.

BTs are external files created by **check out** and by **build**; they are required by **check in**, **promote** and **build**. These files will reside in a subdirectory of the object they are associated with called `BT` if the directory exists, otherwise in the same directory as the object (under Windows they will always reside in a subdirectory). They will have the same name as the object and may have an additional file extension of `_bt` (not under Windows).

BTs are text files and so useful functions such as comparing them by using the **Visdiffs** utility may be performed, in order to determine the difference between two configurations of the same object.

The structure of a BT file for a derived object is:

```
Target: filename
Header: date time user
Depends:
filenamelocation          [P]
|
End Depends.
Actions:
command line for translation actions
End Actions.
BTs for each dependent
End Target: filename
```

The **Target** is the name of the derived object (file), that this is the BT for.

The **Header** contains the date, the time and the user who built the target.

This is then followed by the dependents of the target. This is a list of the names of the dependents and the location of the dependent used to build this target. If the dependent was found in a pool, then the location will be followed by the letter `P`. If the dependent gives an absolute pathname, then the location will be empty. This may arise when objects are used which are outside the **AllChange** system.

This is followed by the actions that were used to construct the target and followed by the BTs for each of the dependents.

The actions are output with all macros expanded unless they have been marked as *non-critical*. Thus any *non-critical* macros will not affect whether an object needs rebuilding even if the macro expansion changes. Any *critical* macros are output expanded and hence any change in the expansion will cause the object to be rebuilt, (e.g. compiler flags changing should cause a rebuild since the derived object will be different with different compiler flags, however the pool from which an object is used should not affect the rebuilding of the object).

The structure of a BT file for a non-derived object is:

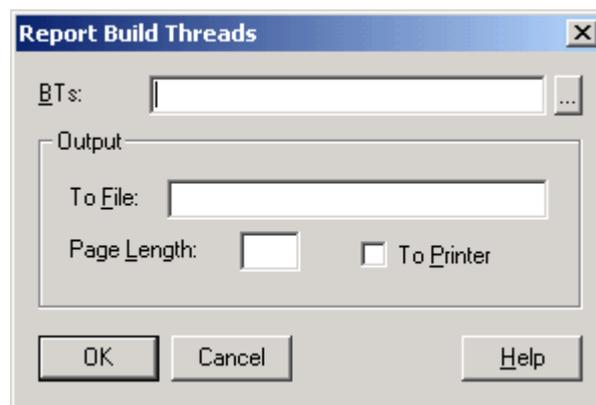
```
Target: filenamepartname [EDIT]
```

This simply gives the name of the dependent and its full part name. The `EDIT` is only present if the part is checked out for edit.

Viewing a Build Thread

A BT is simply a text file and so may be viewed using any text editor. However, **AllChange** provides a report format which may be used to *pretty print* a build thread and show the result in the Command Output window, send the result to a file or the printer.

The report may be invoked via **File | Report_BTs** which will report on the BT for any selected files.



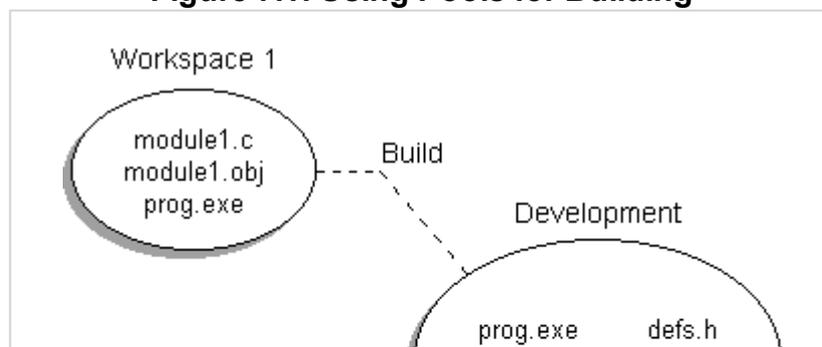
Alternatively the report format is `gibt` and is available from the **Reports | Report | Report General Info** menu option. The **Item** should be the full path to the file whose BT is to be reported on. **Format** should be selected as `gibt`.

Using Pools for Building

When **Build** looks for an object it looks first in the current workspace and then in each of the registered pools (see [Workspace Registrations](#)).

In this way users may share the same pools amongst different workspaces for the objects that they are not working on, and use the objects in their own workspace for those that they are working on, but these will not interfere with other users.

Figure 7.1: Using Pools for Building



If a program `prog.exe` is made up of two source modules `module1.c` and `module2.c` and a single include file `defs.h` which is used by both source modules, then a pool may be used to hold the source, derived object code and executable program which is *current* (the **Development** pool).

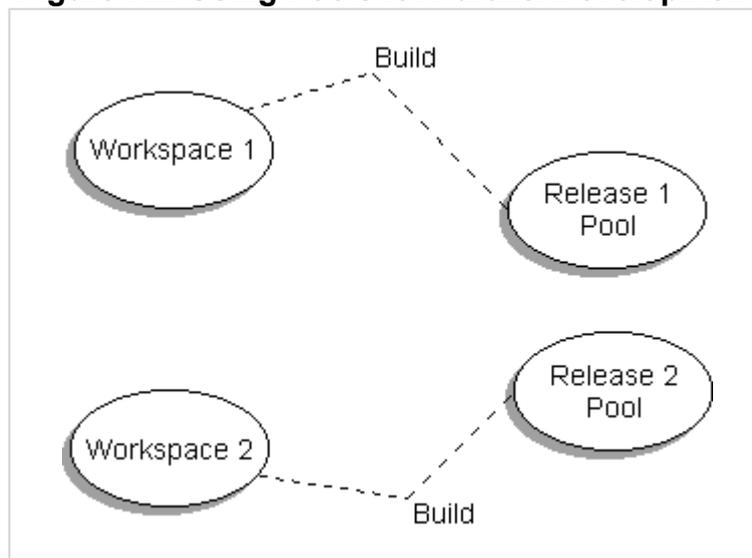
In [Figure 7.1](#) the user of **Workspace1** is modifying the code of `module1.c`. When wishing to compile and test the changes **Build** will use the local copy of `module1.c` and the copy of `defs.h` from the **Development** pool in order to build `module1.obj` which it will place in **Workspace1**. In order to create the `prog.exe` in **Workspace1** it will then use the `module1.obj` from **Workspace1** and `module2.obj` from the pool.

In the meantime the user of **Workspace2** can be altering `defs.h` and creating local copies of derived objects without being interfered with by the user of **Workspace1** and the changes that have been made to `module1.c`. This is because `module1.c` will be obtained from the pool in order to create `module1.obj` in **Workspace2** using the **Workspace2** changes to `defs.h`.

In this way the users of each workspace can work together but without interfering with each other.

Another use for pools is in the maintenance of multiple releases simultaneously. The current configuration for each release can be placed in a pool for that release and users may have workspaces for each release, configured to use different default versions and different pools using registrations. The correct system will be used for building any changes according to the workspace currently attached to. This is illustrated in [Figure 7.2](#)

Figure 7.2: Using Pools for Parallel Development



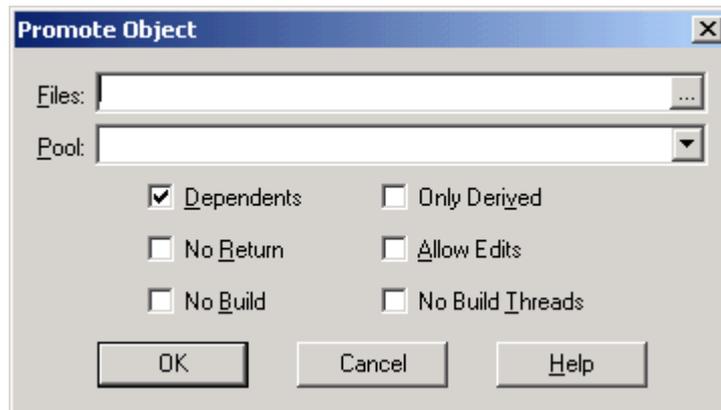
Promoting Files to a Pool

Promoting files to a pool moves files from the current workspace to the *pool directory*. Files in pools are set to be *read only* to prevent accidental modification.

Pools may be defined as *ftp pools* in which case the files will be transferred to the remote pool directory using FTP. This allows files to be managed for remote platforms.

If promoting from a local workspace, select the files required in the file browser and then select the **Promote** dialog available from the **File** menu or from a toolbar button. Alternatively the files may be selected from Explorer and **Promote** selected from the **AllChange** context menu.

If promoting from a remote workspace, then ensure no files are selected in the file browser and select **File | Promote**, you will be presented with a remote file browser for the workspace. Select the required files using this.



Any files selected will be entered as the **Files** to be promoted.

The **Pool** should be the pool that you wish to promote to.

If no options are selected then the files selected will be *moved* to the pool directory together with their *build threads* if BTs are enabled (see Build).

By default (i.e. unless modified by options) the following rules apply:

- objects which are derived from parts checked out for edit may not be promoted.
- objects which are 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.
- if BTs are enabled then 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, see [Using Pools for Building](#).

Various options may be used to modify the behaviour of promote:

Dependents

Specifies that all dependents of each object should be promoted. This option is not applicable if BTs are disabled. The BT for the object is used to determine what the dependents are. This option may therefore be used to allow, for example, the promotion of `prog.exe` and all the objects that were used to create it which reside in the workspace.

Only Derived

Only promotes derived objects. This option is not applicable if BTs are disabled. i.e. it prevents the promotion of source objects. Used in combination with **Dependents** this causes all the derived objects used to create an object to be promoted, but not the underlying source files.

Allow Edits

Allows objects checked out for edit, or derived from objects 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**. 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.

No Return

Prevents objects from being checked in. Use of this option, therefore, causes files to be *copied*, rather than *moved* when they are promoted.

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.

Updating Pools Automatically

It may be useful to maintain pools automatically up to date with the latest version of parts as changes are made. This may be achieved if the configuration option **Auto-update Pools** is selected.

In this case whenever a part is checked in from edit and a new version created any pools registered to the workspace from which the part is checked in will be updated with the new version created.

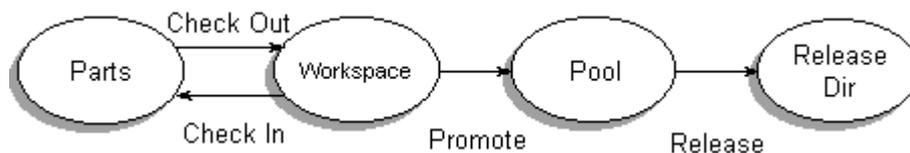
In order to enable pool updating you must therefore ensure:

1. The **AllChange** administrator has enabled **AutoUpdatePools**
2. Any pools to be automatically updated are registered for all workspaces that affect the pools

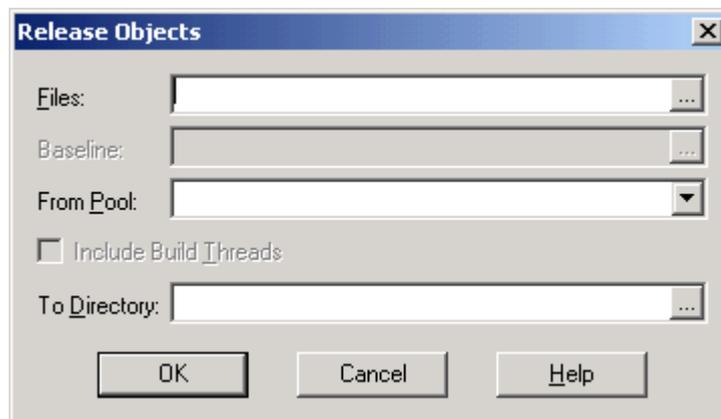
The pool updating functionality is implemented via an ACCEL function and the behaviour may therefore be modified to site/ project requirements. The default behaviour is to update the first registered pool containing the file which has just been checked in from edit. If the file does not exist in any registered pool then you will be prompted for which pool to add the file to.

Releasing files from a Pool

Release Objects copies releasable objects from a pool to a release directory.



The objects are **Files** and may be selected in the file browser. The **Release Object** dialog is available from the **File** menu.



If a **Baseline** is specified then each file released will be checked to ensure that it is composed of elements which have been included in the baseline to ensure that the release is repeatable. This is done by examining the BT files for each file released. If BTs are not being used then a baseline may not be specified. Furthermore, if the pool for the release is an FTP Pool it is not possible to examine the BT files and so again, a baseline may not be specified.

The **Pool** from which the **Files** are to be copied must be selected.

Each **File** will be copied from the **Pool** to the **Release Dir**.

If **Include Build Threads** is selected and BTs are enabled then the BTs associated with each object released will also be copied to the release directory. This can be useful in providing a document for each object as to how that object was built and of what it is composed.

Comparing files

If you wish to view (textually) the changes that have been made between two files which may or may not correspond to parts in **AllChange**, differencing tool may be used.

This may be invoked by selecting one or two files in the **File Browser** in ACE and then **File | Diffs**, or in **Explorer** if the Explorer interface is enabled, select **AllChange | Differences**.

By default if the files selected are Word documents then Word is used to compare the files, if the files are binary then a simple file comparison for equality is performed, otherwise the supplied **VisDiffs** is used.

If two files have been selected the **VisDiffs** tool will be invoked to compare the two selected files. If just one file is selected, the **VisDiffs** tool will be invoked and it will prompt for the second file to be compared. This is useful when the two files reside in different directories.

Alternative third party file comparison tools may be used by specifying them in the [External Tools](#) option in ACE.

If using Araxis merge then folder and [subsystem comparison](#) are also supported. Note that file sizes with Araxis will always be shown as 0.

File Stamping

About File Stamping

Facilities are provided in **AllChange** to "stamp" files that are managed by **AllChange** with useful information from the AllChange database such as part version, author and date.

This facility is supported for the following file types:

- [text](#) (ASCII) files using the *keywords* facility
- [MS Word documents using word fields and macros](#)
- MS Excel workbooks using formulae in cells

Stamping Text Files

The facility to "stamp" text files is provided by a **Keyword** mechanism.

Whenever possible, keywords should be included somewhere in the text files which are subject to **AllChange**'s version control facilities. When the files are then later retrieved, date, time, author etc. stamping will then automatically be performed.

To include a keyword in a file, *\$keyword\$* (note the two \$ characters) must appear somewhere in the file, where *keyword* is one of the keywords known to **AllChange**, e.g. *\$ACheader\$*. When a version of the file is extracted (checked out or by using Get Version) and a *\$keyword\$* sequence is recognised in the resulting workfile it replaces it by:

\$keyword:substitution\$

where *substitution* is the appropriate substitution sequence for that keyword, e.g.

\$ACversion: 3\$

Furthermore, if a new version of the file now containing

\$keyword: substitution\$

is subsequently stored as a new version then when this new version is retrieved the keyword will again be recognised and will produce:

\$keyword:new substitution\$

e.g., continuing the above:

\$ACversion: 4\$

This means that once a keyword has been inserted into a file it will automatically be updated as necessary whenever a new retrieval is performed.

Keywords may be placed somewhere in a file where they do not cause any interference with the file's purpose (e.g. inside comments), or they may be placed in such a way that they will actually be output (e.g. inside a programming language's *write* statement), or they may be found and displayed by the **Keywords** facility. This might be used, for example, in order to identify from which versions of which modules an executable program was compiled.

The keywords recognised by **AllChange** and their substitutions are listed in [Figure 7.3](#).

Figure 7.3: Keywords

Keyword	Substitution
\$ACheader\$	Equivalent to: \$ACpart\$ \$ACversion\$ \$Date\$ \$Author\$.
\$AClog\$	The log is \$ACheader\$ plus the version comment if an arbitrary field named Comment is defined, otherwise the CRs Solved by the version. It is placed <i>after</i> the \$AClog\$. This Comment part of the log may be customised to contain additional or alternative information, see the AllChange Administrator Manual for further details.
\$ACpart\$	The full partname of the component.
\$ACversion\$	The version number.
\$ACident\$	The substitution value is taken from a dynamic value defined during Check Out. This may/will be customised for site specific requirements by your AllChange administrator, see the AllChange Administrator Manual for further details.
\$Author\$	The name of the person who created the version.
\$Date\$	The date and time when the version was created. This will be the date/time in UTC unless the Keywords use local time configuration option is set in which case it will be in the current user's local time.
\$Ident\$	No substitution performed, but recognised by Keywords . Arbitrary text may be placed between the \$Ident and the \$ to be output by Keywords .
\$Line\$	The current line number in the workfile. The user may wish to use this in generating error messages to identify the offending line in the source file.
\$Locker\$	The name of the person who checked out for edit the version; empty if not checked out for edit.
\$Revision\$	The VCid of the version.
\$Today\$	Today's date and time, i.e. when the version was checked out.
\$VCfile\$	The name of the VC file.
\$VCpath\$	The full pathname of the VC file.
\$Workfile\$	The name of the workfile file.

The \$AClog\$ keyword differs from other keywords in that its substitution appears *after* the second \$ rather than inside it. This means that previous substitutions are not lost; instead it grows larger on each successive **Check In from Edit** and **Check Out**, thus building up a complete history including information on all the previous releases which make up the current release. It is intended to be placed inside a programming language's comment. If the log grows too large for comfort, lines containing information about old version which are no longer of interest may be deleted before putting away the next release: they will not then appear in future **Check Outs**.

Each line generated by \$AClog\$ follows precisely the same layout as the line on which the AClog\$ keyword appears; this allows for programming languages which do not permit multi-line comments. Two examples:

```
/*
   $AClog$
*/
```

might be used in C to produce a *single multi-line comment*, e.g.:

```

/*
  $AClog: $
  /product/sw/source/file.c 1 99/07/13 22:34:46 demo
  My Comment
*/
while
  -- $AClog$

```

might be used in Ada to produce *multiple single line comments*, e.g.:

```

-- $AClog: $
-- /product/sw/source/file.c 1 99/07/13 22:34:46 demo
-- My Comment

```

The text substituted for the Comment part of the log is taken from the **Comment** field if one is defined as a version arbitrary field or the CRs Solved by the version if not, at the time the version was checked in. This may be changed by your AllChange administrator to show, for example, The baseline in which the version was released in addition to or as well as the comment, see the *AllChange Administrator Manual* for further details

Although keyword substitutions are enabled by default there are certain files on which they should not be performed and should be disabled; any files with one of the defined **Binary suffixes** will not have keyword substitution performed. See the *AllChange Administrator Manual* and *AllChange VC Tools Manual* for further details.

Stamping Word/Excel Documents

The *AllChange* integration with Word allows the "stamping" of Word and Excel document/workbook files — marking them with information about their corresponding *AllChange* version. This information is stored in the document's properties; it may be viewed from within Word/Excel and from Explorer, and it is also possible to actually make the information appear in the document/workbook.

This facility should be used in place of the keyword facility for text files as described in the proceeding section since Word document files and Excel spreadsheet files are *not* text files.

Document stamping requires Word 8 (Office 97) or later.

This facility will only be available if the **Word Integration** option has been selected during the workstation installation. In addition for the Excel interface the add-in must be enabled. If it has been installed there will be an *AllChange* menu available on the menu bar in Word and an *AllChange* cascade menu from the **Tools** menu in Excel.

Document stamping involves extracting information from *AllChange* and storing this in the Word document or cells of an Excel spreadsheet.

There are two ways to perform this action:

1. At any time the user may cause the current document's *AllChange* information to be updated selecting **Stamp Document** from the AllChange menu
2. Stamping will be performed automatically when a document version is *checked in* provided the **AllChange Enable Word document stamping** configuration option has been selected — see the *AllChange Administrator Manual* — and it is not explicitly disabled for the current document. The document need not be the current document in Word/Excel. Stamping may be disabled for a specific document by selecting the **No_Doc_Stamp** field for the part. This field must be defined to enable this facility. This can be useful, for example, for large documents which have not stamp fields in them and which take a considerable amount of time processing searching for these fields on check in.

Whenever document stamping is performed several things occur:

1. **AllChange** determines the version corresponding to the document/workbook. This means that **AllChange** will have to be able to calculate which component corresponds to the file. In order to do this the file must be saved into a directory which is in a workspace known to **AllChange**.
2. Custom document properties are created/updated to reflect current **AllChange** information. Custom properties are treated as corresponding to **AllChange** information if their name starts with **AC**. Some custom properties are created automatically for all documents, while others are only maintained if they have been explicitly added to the document.

The following properties are created automatically:

ACComponent (the name of the component)

ACDate (the date the version was created - This will be the date/time in UTC unless the [Keywords use local time](#) configuration option is set in which case it will be in the current user's local time)

ACUser (the user who created the version)

ACVersion (the version number)

Additional properties which are of interest may be created by adding a custom property to the document. The custom properties must be named starting with **AC** and then the name of an **AllChange** field (e.g. **ACStatus**). The fields which may be used are those which are exported from **AllChange** for use with the document stamping feature. These are customisable, but as supplied include the following (in addition to the properties automatically created):

ACActualDate

AC<Arbitrary field new name> for all arbitrary fields which are defined. Any spaces in the name must be replaced with _(underscore).

ACClass

ACCompObsolete

ACCompStatus

ACCRAffecting

ACCRSolved

ACDupVer

ACFlags

ACLocation

ACLocationRaw

ACLocked

ACName

ACNoFile

ACNoVC

ACNoVer

ACObsolete

ACParent

ACPart

ACPredVer

ACStatus

ACSymbname

ACType

ACUser

AC<Version Arbitrary field new name> for all version arbitrary fields which are defined. Any spaces in the name must be replaced with `_` (underscore).

Document properties are created from the **File | Properties** dialog on the **Custom** tab; it does not matter what you enter for the value when adding a custom property as **AllChange** will set the value anyway (but Word seems to require that something be typed in).

The current value of a custom property may be viewed anytime you have a document open in Word from the **File | Properties** dialog **Custom** tab.

3. The in-built **Comments** property is searched for any references to **AllChange** properties and appropriately updated. The **Comments** property appears on **File | Properties** dialog **Summary** tab. Any lines which begin with (there must be no leading spaces) the name of an **AllChange** custom property (as described above) immediately followed by a = (equals) are treated as such a reference: all text to the right of the = up to the end of that comment line is replaced by the current value of the corresponding property. So, for example, to include a reference to the associated **AllChange** version in a document's Comments simply place the following line (with no leading spaces) somewhere in the Comments:

```
ACVersion=
```

Thereafter this line will be updated automatically to read something like:

```
ACVersion=012
```

Like custom properties, the Comments can be viewed anytime you have a document open in Word. Unlike custom properties, however, the Comments can *also* be seen from Explorer by right-clicking on a Word file and selecting the **Summary** tab of the **Properties** dialog.

4. The document itself is searched for any references to **AllChange** properties and these too are updated.

To place an **AllChange** property in a Word document you must place it in a *field*: move to the desired location in a document, select **AllChange | Insert AllChange Field** (or the corresponding toolbar item), and choose from the list of available **AllChange** properties. You may have as many such fields as you please anywhere in a document, including in page headers and footers.

To place an **AllChange** property in an Excel workbook you must place it in a cell: move to the desired cell, select **Tools | AllChange | Insert AllChange Field**, and choose from the list of available **AllChange** properties. **AllChange** properties may only be inserted in cells as formulae and may not be placed in cell comments nor in printed page header/footers

Remember that these fields — like the custom properties — only get updated when the document is checked into **AllChange** or you explicitly run the **StampDoc** macro. Remember too that a custom property must first exist (as described above) before it can be selected to appear in a field.

5. Finally, if document stamping is being performed automatically as part of **AllChange CheckIn** the document is now stored away with all the foregoing changes. Whenever that version of the document is subsequently retrieved it will include all the stamps that have been made.

Here is a worked example you may wish to follow to see stamping in operation:

1. Ensure you have installed **AllChange** with the Word integration option selected for your workstation. It will also be necessary to ensure the **Enable word document stamping** configuration option has been enabled by your **AllChange** administrator (you can find out if this is enabled by running the `giconfig` report from **Reports | Report | Report General Info**).
2. Start **AllChange** and then minimise it
3. Run Word; create a new document.
4. Type in:

AllChange component:

and then select **AllChange | Insert AllChange Field** (or use the corresponding toolbar button) and select **ACComponent** from the available field names; the line should now look like:

AllChange component: <ACComponent>

Now on a new line, type in:

AllChange version:

and repeat as above to insert the **ACVersion** field. Your document now contains 2 lines as follows:

AllChange component: <ACComponent>

AllChange version: <ACVersion>

5. Save the document (as `test.doc`) in a directory for which you have defined an **AllChange** workspace to which you are allowed to attach. If you do not save the document in a directory that is known to **AllChange** then **AllChange** will not be able to find the document.
6. Check the document into **AllChange**. You could use **AllChange | Check In** from within Word or Explorer, or **File | CheckIn** from within ACE; see [Checking Files In and Out of AllChange](#). Note that if you do not **CheckIn** from within Word you will need to close the document before checking it in.
7. Now check the document back out again, for edit purposes and open it in Word (if you use **AllChange | CheckOut** from within word then the document will be automatically opened. The document will look something like:

AllChange component: /product/docs/test.doc

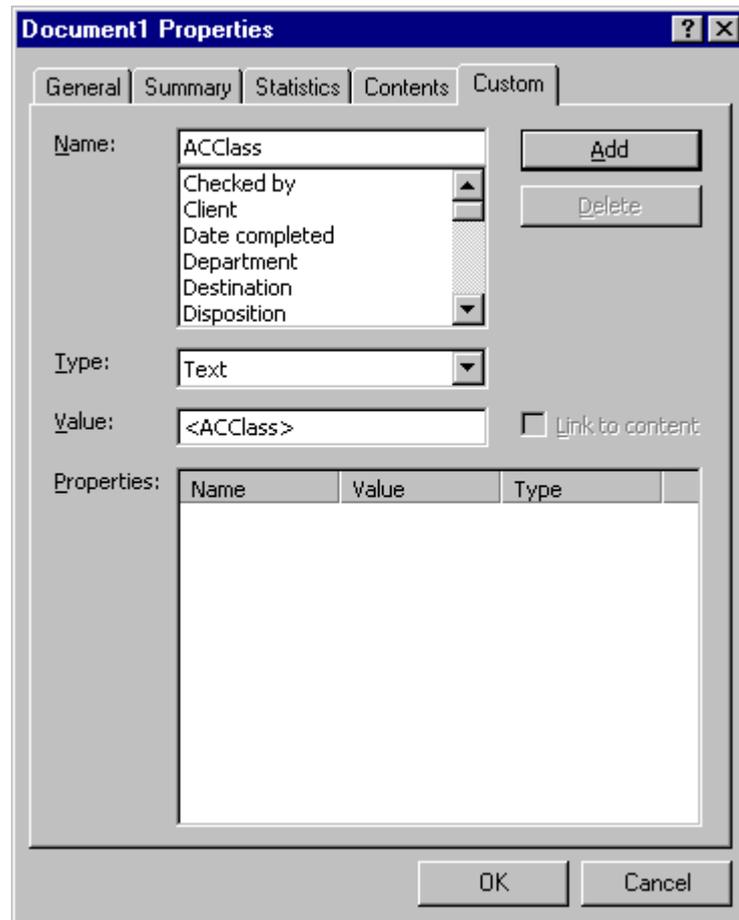
AllChange version: 001

The stamping has taken place!

Now let us try some of the more advanced features.

8. Select the **Custom** tab of the **File | Properties** dialog, type **ACClass** (with capitals as shown) into the **Name:** edit control and something like <ACClass> (it does not matter what) into the **Value:**

edit control.



Select **Add** and then **OK** to exit the dialog.

Now insert another line in the document and type:

AllChange class:

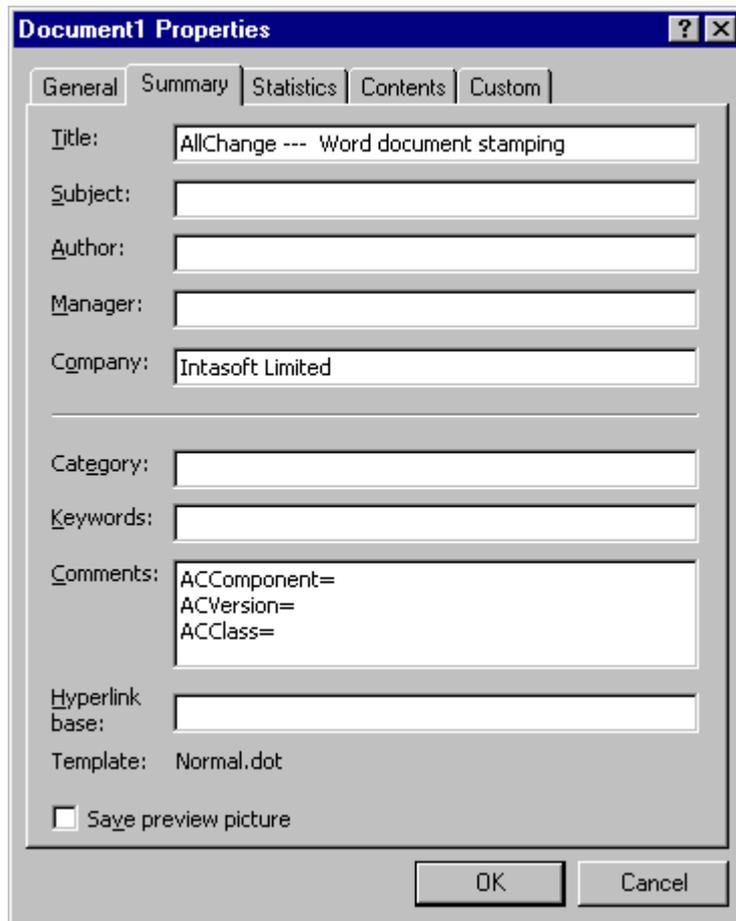
then select **Insert AllChange Field** as before and select **ACClass** which we have just defined as a new field.

9. While we are at it also select the **Summary** tab of the **File | Properties** dialog enter the following three lines into the **Comments:** edit control:

ACComponent=

ACVersion=

ACClass=



and select **OK**.

10. Save the document and check it in again.
11. Now check it out again. This time it should look something like:

AllChange component: /product/docs/test.doc
AllChange version: 002
AllChange class: document

Note that the version number has been updated to the next version and the class (as selected when you first put the document under **AllChange** control) is also now shown.

12. Finally, exit Word, and navigate to the `test.doc` workfile in Explorer, right click and select **Properties** from the context menu. Select the **Summary** tab; the **Comments**: edit control shows information about the checked out version.

Windows Explorer Interface

Facilities are provided which make various **AllChange** operations which are file based available directly from Explorer.

The integration is implemented as a Shell extension which adds an **AllChange** menu to Explorer's context menu (the menu which pops up when you right-click on a file or directory) or you can drag from Explorer and drop onto **AllChange**, providing access to the interface functions.

To enable this facility you need to select it in the Workstation Installation, see the *AllChange Administrator Manual*.

The facilities available include:

Check In:

Allows files to be checked into **AllChange** using the **Check In** function, see [Checking In](#).

Check Out:

Allows files to be checked out of **AllChange** using the **Check Out** function, see [Checking Out](#).

Differences:

Allows 2 files or 2 versions of a file which corresponds to a part in **AllChange** to be compared, see [Finding out What has Changed](#) and [Comparing files](#).

Report

Runs a predefined report on the parts which correspond to the files/ directories selected. As supplied this will report on the check-out status of each file selected (or each file in a selected directory). This report may be tailored to site specific requirements, see *AllChange Administrator Manual*.

Update Workspace

Updates the workspace that corresponds to the directory selected using the **UpdateWorkspace** function, see Update Workspace. Note that it is only available when 1 directory is selected.

Versions

Shows a list of the versions of the selected files together with the date that each version was created.

Autobuild

Invokes the **AllChange Autobuild** tool with the selected files as the **Source**, see [How to Set Up a Buildfile](#).

Build

Invokes the **AllChange Build** tool with the selected files as the **Target**, see [Building](#).

Keywords

Scans the selected files for any **AllChange** keywords, see [Stamping Text Files](#)

Promote

Invokes the **AllChange Promote** function with the selected files as the files to be promoted, see [Promoting Files to a Pool](#).

MS Word/Excel Interface

Facilities are provided which make various **AllChange** operations which are file based available directly from MS Word and MS Excel.

The integration supports MS Word/Excel 97 and later (reduced functionality is also available for MS Word 6). It is implemented as Templates and add-ins which add an **AllChange** menu (to the menu bar in Word and to the Tools menu in Excel) providing access to the interface functions.

To enable this facility you need to select it in the Workstation Installation, see the *AllChange Administrator Manual*. In addition for the Excel interface the add-in must be enabled.

The facilities available include:

Check In:

Allows the current file to be checked into **AllChange** using the **Check In** function, see [Checking In](#).

Check Out:

Allows a file to be checked out of **AllChange** and opened as the current document/workbook using the **Check Out** function, see [Checking Out](#).

Compare Versions (Word only):

Allows 2 versions of the currently active document which corresponds to a part in **AllChange** to be compared, see [Finding out What has Changed](#). If no document is open, then you will be prompted to check one out in order to proceed with the compare against the newly checkout component.

Version Information:

Runs a predefined report on the part which corresponds to the currently open file. As supplied this will report on the check-out status of the file. This report may be tailored to site specific requirements.

List Versions:

Shows a list of the versions of the selected files together with the date that each version was created.

Insert AllChange Field:

Provides facilities to insert information about the version of the document/workbook from the **All-Change** database, see [Stamping Word Documents](#).

Add Requirements (Word only):

Allows CRs to be imported into **AllChange** from a word document (which may be a requirements document), see [Creating CRs from a Word Document](#)

Exit AllChange:

Allows the currently running **AllChange** session to which Word is connected to be exited.

Rhapsody Interface

Facilities are provided which allow the files for a project to be checked into and out of **AllChange** directly from the Rhapsody interface.

For versions of Rhapsody before 3.0 see below for details. For Rhapsody 3.0 the MCSCCI is supported and the standard facilities are supported, see [Checking In and Out from Microsoft Development Environments](#).

Interface for versions of Rhapsody earlier than 3.0

The facilities provided are determined by Rhapsody and implement only a very simple interface to **All-Change** which does not make use of the full power of **AllChange**. These *may* be tailored to provide more powerful facilities.

See **AllChange** Administrator Manual for details of installing the Rhapsody interface.

In order to use the interface:

- Ensure that you have a workspace defined for use with your Rhapsody project and that the sub-system associated with it exists.
- Ensure that (in Rhapsody) the **Project Properties** have a **Subject of Configuration Management** and a **Meta Class** of **AllChange**.

When invoking a CM operation from Rhapsody, if **AllChange** is not currently running it will start it for you. This will be a minimised ACE which is for use with the interface and not for interacting with. It may be closed by selecting **Close** on the right click menu on the entry on the system bar in Windows. If you wish to interact with **AllChange** then you can invoke another (interactive) instance of ACE.

As the Rhapsody CM definition requires no interaction (i.e. no dialogs from) with the CM tool provider, no classes are used. Parts may not have the **Require CR** attribute and there must not be any compulsory arbitrary fields. There is also no interface to baselines.

The items which may be checked in and out from Rhapsody may be accessed from **File | Configuration Items** and the following operations may be performed.

Connect to Archive

This *must* be the first operation performed. Select **OK** on the resulting **Connect to Archive** dialog box; The **Archive path** is ignored (not relevant).

Add to Archive

All items *must* be added to the archive before they can be checked in and out. This creates a component in **AllChange** with no initial version.

CheckIn to Archive

Checks items in which have previously been checked out or just added to the archive. Selecting

Locked checks the file back out for edit after checkin, otherwise the file is checked back out read-only not for edit. All other controls are ignored (not relevant to **AllChange**). This performs an **AllChange Return** or **Newversion** command as appropriate (**Newversion** is used if this is the first version of a new component).

CheckOut from Archive

Checks items out of **AllChange**. Selecting **Locked** checks out for edit, otherwise the file is checked out read-only. All other are controls ignored (not relevant to **AllChange**). This performs an **AllChange Issue** command.

Show Items in Archive

Shows information about all files in the current project which have been added into **AllChange**, and the default version if it has at least one. **Version** and **Locked by** columns should be populated correctly.

In the main **Configuration Items** dialog the **Mode** column should be populated correctly. the **Version** column will be empty until source files contain **AllChange** keywords, these will be added automatically when new items are created. The Rhapsody documentation states that keywords will be embedded *when the item is first checked in to CM*. However, this does not seem to be the case when checking in existing files. This can be done on existing files (if wanted), by checking them out for edit from ACE editing them so that lines 3 and 4 read:

```
- _cmheader = "$ACheader$
$ACversion$
```

(instead of `$Header...` and `$Revision...`), check them back in, then back out again. You should now find the column is correct from now on in Rhapsody.

DOORS Interface

Facilities are provided which allow DOORS modules to be checked into and out of **AllChange** either directly from the DOORS interface or from ACE. This is achieved using the archive/ restore module facilities in DOORS to create a file which can then be checked into and out of **AllChange**.

Prior to archiving a module, information is stored in the module about all the links from that module so that on restoration of the module the link information can be restored. Note that **Link** modules themselves may not be checked in and out of **AllChange** as they cannot be archived and restored.

When a module is checked out for read only purposes then the module will be set as read only for that user and will only be writable when it is checked out for edit.

In order to make use of the DOORS integration it must have been properly installed and set up — see the **AllChange** Administrator Manual for details.

A DOORS project is represented as a subsystem in **AllChange** with the same name as the DOORS project. The modules of the project will be stored within this subsystem as components with the module name with a `.zip` extension. The class of the **AllChange** components which hold the DOORS modules is `doors` by default unless otherwise specified in the **DOORSPartClass** configuration option, this will be referred to as the *DOORS class*. Only components with this class will be treated as DOORS modules.

All DOORS projects should be stored under a single subsystem (e.g. `/DOORS`) as all checking in and out of DOORS modules takes place using a single workspace which must be named `DOORSWS` unless otherwise specified in the **DOORSWorkspace** configuration option, which should be associated with the top level DOORS subsystem (e.g. `/DOORS`), this will be referred to as the *DOORS workspace*. Each DOORS project is then a subsystem within the top level DOORS subsystem (e.g. `/DOORS/Project1` `/DOORS/project2`).

Whenever checking in and out DOORS modules if DOORS or ACE are not currently running they will be started automatically. If the required DOORS project is not open, then **AllChange** will prompt you for your DOORS user name and password and will open the project. If the project is already open, but **AllChange** does not yet know your DOORS password for that project then you will be prompted for the password only.

From within ACE if you are attached to the *DOORS workspace* then the following functionality is supported for components of class `doors`:

Add Part

If a component of the *DOORS class*(with a name of the form *module-name.zip*) is added using **Part | Add** then the module of the same name as the component (excluding the `.zip` extension) is archived from DOORS prior to being stored under **AllChange** control.

Check In

If a component of the *DOORS class* is checked in (e.g. using **Part | Check In**) then the module of the same name as the component (excluding the `.zip` extension) is *archived* from DOORS prior to being stored under **AllChange** control.

Check Out

If a component of the *DOORS class* is checked out (e.g. using **Part | Check Out**) then after the part is checked out of **AllChange** the module of the same name as the component (excluding the `.zip` extension) is *restored* into the DOORS project with any links being recreated.

From within DOORS an **AllChange** menu will have been added to your main DOORS window as well as to the Module windows. These menus provide access to the following:

Check Out

This will bring up an **AllChange** parts browser, from which you can select the modules you wish to check out. These will then be checked out of **AllChange** using the standard **Check Out** function and restored into the DOORS project removing any previous definition of those modules.

Check In

If selected from the **AllChange** menu on an open module then the current module is checked in. If selected from the **AllChange** menu in the main view then this will bring up a window with a list of modules to select from. Any modules selected will be checked in. The module will be archived to a zip file and then stored in **AllChange**. The standard **Check In** function is used, see [Checking In](#). Note that if adding a new module to **AllChange** be sure to select the `doors` class for the new components.

Report Issues

This will run an **AllChange** report to show DOORS modules are currently checked out.

Eclipse Interface

Facilities are provided which allow **AllChange** to be the eclipse Team Provider. This allows files to be checked in and out directly from within the Eclipse environment, deletion and moving of files and team synchronisation.

In order to make use of the Eclipse integration it must have been installed on the workstation, see the Administrator Manual for details.

In order to use **AllChange** with eclipse you will need an **AllChange** workspace defined which corresponds to your eclipse workspace.

To enable **AllChange** for an Eclipse project, right click on the project and select the **Team** menu. Then select **Share Project...** and pick **AllChange**. AllChange will now be set as the Team provider for that project. The Team right click menu will now contain the **AllChange** Team functions.

Ensure that the **AllChange** Decorators are enabled in your Eclipse decorators to see the **AllChange** decorators. They should be enabled by default.

File Decorators

The following symbols appear at the front of the resource name. These symbols indicate the status of the file in the current Eclipse workspace:

- > Checked Out For Edit
- + Checked Out Read Only

- X (grey) Not under **AllChange** control
- - (blue) Not Checked Out

Additional information can appear at the end of the resource name:

- [OUTOFDATE] (red) Checked Out Read Only, not latest version
- [OUTMULTIPLE] Checked Out to multiple workspaces
- [OUTOTHER] Checked Out For Edit by another user
- [OUTDIFFERENTWS] Checked out (by current user) to another workspace

The icon on a resource will indicate one of the following states:

- Checked out
- Checked in
- Conflict

Moving/Deleting Files

The plug-in is hooked into the move, rename and delete actions in Eclipse.

If a file is moved in Eclipse, it will prompt you whether to move the corresponding part in **AllChange** to the same place. `AllChange` will automatically check in the file and will create any new subsystems if necessary.

If you delete a file in Eclipse, the corresponding part in **AllChange** will be checked in and deleted.

Synchronisation Support

AllChange integrates into the **Team Synchronisation** facility in Eclipse.

Run the synchronisation wizard in Eclipse and pick **AllChange**. This will check any files that are checked out for edit for differences between the working file and the version that was checked out. Any files with differences are listed in the Synchronisation view. Clicking on the file will bring up the Eclipse diffs to show any any changes between the workfile and the **AllChange** version.

There is also the option to revert to the last version in **AllChange**.

Limitations

- Decorator information is saved with the Eclipse resource and will be accurate up to the last **AllChange** operation. On start-up and all **AllChange** operations update the decorators to the current status. The **Refresh Status** option on the team menu may be used to update the decorators if this is needed.
- The merge used in optimistic locking is the **AllChange** merge, not the Eclipse merge

SolidWorks Interface

The **AllChange** integration with SolidWorks allows the files manipulated by solidworks (assemblies, parts, drawings) to be subject to change and configuration management with **AllChange** from within the SolidWorks environment. Using the integration ensures that the correct versions of any dependents of an assembly or drawing are always used.

The following functionality is supported:

- Check In/Out
- Status progression
- Voting
- Synchronisation of SolidWorks Properties with AllChange arbitrary fields
- Comparison of versions of SolidWorks files

The Integration supports SolidWorks 2010 SP0 or later.

The integration is implemented as an Add-in to SolidWorks which must be installed for each SolidWorks user who wishes to use the integration with **AllChange**, see the [AllChange Administrator Manual](#) for details of setting up to use the integration.

An **AllChange** tab is added to the task pane, an **AllChange** tab is added to the Command Manager, an **AllChange** menu is added to the Menu bar, and an AllChange cascade menu is added to the context menu of the Feature manager.

AllChange Task Pane Tab

The **AllChange** tab on the Task Pane will show a tree view of the details of the currently opened SolidWorks document and its dependencies including AllChange Version, User who checked out the file, Status, and Description where Description is the AllChange version Description or SW-Description arbitrary field.

Files displayed in red indicate that this is not the latest version.

For each file the icon will show the check-out state of the file as follows:



File is not in AllChange and may be added to AllChange



File is checked out read only from AllChange



File is checked out for edit from AllChange



File may not be checked in/out of AllChange - this may be because the file is in a directory which is not an AllChange workspace

Selecting a file will show details of that file in the bottom half of the pane as follows:

Version	The AllChange version. If the file is checked out for edit it will show the version checked out and the new version that will be checked in
User	The AllChange user who created this version.
Workspace	The AllChange workspace that the file is checked out to.
Status	The AllChange status of the version
Other Fields	AllChange Arbitrary Fields which have the SolidWorks Show In Taskpane attribute . The Revision field in the Intasoft Standard configuration is shown by default

The toolbar and context menus provide access to the AllChange integration functionality as described below.

Synchronising Custom Properties

Any custom properties which have the same name as an **AllChange** field (built in or arbitrary) can be synchronised with **AllChange**. **AllChange** arbitrary fields which are to be synchronised must have the [SolidWorks Synchronise attribute](#) and this must define which direction the synchronisation is to occur in. Any properties which do not exist will be automatically created on synchronisation.

In addition any **AllChange** field whose name is prefixed with SW- and the rest of the name matches a SolidWorks custom property can be synchronised (e.g. SW-Description will synchronise with the SolidWorks Description custom property).

Properties are automatically synchronised on opening a document if the **Auto-synchronise properties on opening documents** SolidWorks option is set. Also whenever a custom property is updated in SolidWorks the corresponding **AllChange** field will be updated if synchronisation for that field is set to "**To AllChange**" and the **Update AllChange fields immediately on changing document properties** SolidWorks option is set.

On Check In of files the **AllChange** arbitrary fields will be updated from the SolidWorks custom properties, built in fields will not be updated in **AllChange**.

Custom properties can also be synchronised using the **Synch Properties** function from Menu/Tool-bar/Command Manager.

AllChange Functions

The following **AllChange** functions may be performed from within SolidWorks.

Open from AllChange

Allows a file which is controlled by **AllChange** to be opened in SolidWorks.

The workspace containing the file(s) you wish to open is prompted for. A dialog is then presented of the **AllChange** parts for the selected workspace, select the file to be opened. A specific version may be selected (including an old version). If no specific version is selected the default version will be opened.

If the file is already present in the workspace at the correct version (i.e. is already checked out - read only or for edit - from **AllChange**) then the file will simply be opened. If it is not yet checked out then the file and appropriate versions of all its dependents will be checked out read only into the selected workspace. Where a different workspace is required for a dependent then this will be used.

This allows a previous version of a file including its dependents to be opened in SolidWorks.

Check Out

Allows SolidWorks files to be checked out of **AllChange** for edit purposes.

A dialog is presented allowing selection of which SolidWorks files are to be checked out for the current open document. This allows all files which related to the current document to be checked out at the same time. The correct version of all dependents for an assembly or drawing will be checked out according to the parts affected relationships of the parent assembly/drawing. The files will be checked out to different **AllChange** workspaces as needed.

If any further information is required by **AllChange** according to the class this will be prompted for in the normal way, for example if a CR is required then this will be prompted for.

The new version created in **AllChange** will have the arbitrary field values from the predecessor version copied to the new version.

AllChange arbitrary fields which are to be synchronised with SolidWorks custom properties will be synchronised with the properties.

Get Latest

Allows the latest version of files to be checked out of **AllChange** for read only purposes.

A dialog is presented showing the files for the current open document. Only those which are not at the latest version may be selected to get the latest version.

Check In

Allows SolidWorks files to be checked into **AllChange**. This includes adding new files to AllChange as well as checking in files which are already controlled by **AllChange**.

A dialog is presented allowing selection of which SolidWorks files are to be checked in for the current open document. This allows all files which are related to the current document to be checked in at the same time.

On Check In each selected file will be checked into **AllChange**. If the file does not exist already in **AllChange** then a new **AllChange** part will be created. If there is a part class with the attribute **SolidWorksDoc** or **SolidWorksDrawing** then this will be used as appropriate. If no such class exists or multiple such classes exist the standard **AllChange** [Add AllChange Components](#) dialog will be shown allowing the class to be selected. The files will be checked in from different **AllChange** workspaces as needed.

If any further information is required by **AllChange** according to the class this will be prompted for in the normal way as if checking in from within **AllChange**.

AllChange arbitrary fields which are to be synchronised with SolidWorks custom properties will be synchronised with the properties.

The **AllChange** Parts Affected for the version checked in will be set to the appropriate version of any dependents, thus ensuring that there is a record of which versions of each dependent are applicable to a particular version of a parent (e.g. an assembly).

Undo Check Out

Allows a check out to be discarded.

A dialog is presented showing the files for the current open document. Only those which are checked out for edit may be selected.

Refresh

Refreshes the information in the **AllChange** tab on the task pane.

Status

Allows the status of the selected file or currently open document to be progressed in **AllChange** according to the life-cycle defined for the class of the **AllChange** part. Only valid progressions from the current status will be offered.

If the status is a Check Out or Check In status then the corresponding action will be taken as appropriate.

This allows SolidWorks documents to be submitted for approval and reviewed.

Cast Vote

Allows a vote to be cast to approve a document. This option is only available if the document is in a status for which a vote has been initiated.

This allows approvals of documents to be performed.

Compare

Allows the selected file which is checked out (for edit) to be compared for changes made to the checked out version (if it is checked out for edit) or with another version. If **Compare with other version** is selected then a dialog will be presented allowing the selection of the version to be compared against.

The SolidWorks comparison facility is used to display the differences.

Synch Properties

Allows SolidWorks custom properties to be synchronised with **AllChange** arbitrary fields.

A Synch Properties dialog will be presented. This shows the custom properties which correspond to an All-Change field which has the **SolidWorks Synchronise** attribute. The value of the SolidWorks property is shown on the left and the value of the corresponding **AllChange** field is shown on the right.

An **Action** is specified in between. The action may be:

- Synched - this field is already synchronised, no further action required
- Set - set the **AllChange** field to the value of the SolidWorks property
- Get - set the SolidWorks property to the value of the **AllChange** field
- Ignore - do nothing
- Choose - choose which action to take

Options

Various options may be used to determine the behaviour of certain aspects of the integration:

- **Dont prompt to check out read-only documents:** if unchecked then on opening a document which is read only and controlled by **AllChange** you will be prompted as to whether to check out the document for edit.
- **Get Latest versions on opening read-only documents:** if checked then on opening a read only document which is controlled by **AllChange** if it is not the latest version then the latest version will be retrieved from **AllChange** and opened.
- **Auto-synchronise properties on opening documents:** If set then whenever a document is opened the properties will be synchronised with **AllChange** fields.
- **Update AllChange fields immediately on changing document properties:** If set then when document properties are changed, they will be synchronised with **AllChange** fields.
- **Revert property value if corresponding AllChange field is not settable:** If set then if an attempt is made to update a property which is defined to be synchronised from **AllChange** and is not settable in **AllChange** it will revert to the value of the **AllChange** field automatically.
- **Reset options to default settings:** this will reset all options to the default settings
- **Reset the Command Manager AllChange tab:** this will reset any customisations to the **All-Change** tab on Command Manager to the defaults.

Building

About Building

The Building facilities within **AllChange** are related to the maintenance, updating and regeneration of files which *depend on* other files such as executable programs which depend on source files. This may mean the *compilation* of source modules to create object files which in turn may need to be *linked* together to create an executable program.

One of the important benefits of using build facilities is that the build tool only causes the regeneration of those files which are *out of date* and need rebuilding.

In order to calculate what files *need* rebuilding and how to build them the build tool requires information about which files depend on which other files and what action(s) to perform if a file is to be built.

This dependency information is extracted from a (user-supplied) file called the *buildfile*.

A file is judged as needing building if it is *out of date* with respect to any or all of the files on which it depends.

Filea is judged to be *out of date* with respect to *fileb* if:

- If an existing file exists which is not comprised of the required versions of its constituent parts. This is determined by comparing the *desired* Build Thread (BT) with the existing file's BT.
- If the BT's match but the file is a part which is checked out for edit then time stamps are used and *filea* is out of date with respect to *fileb* if the latter's "last modification" time is more recent than that of the former (i.e. *fileb* is *newer* than *filea*)
- or if either file does not exist.

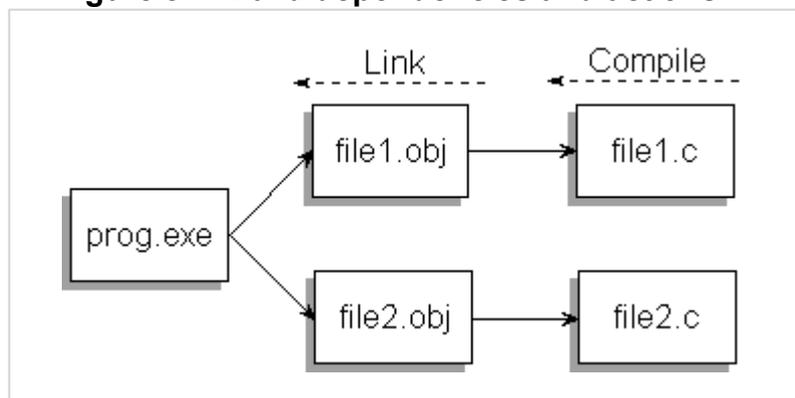
The time stamp that a file will have which has been **checked out** from **AllChange**, but not (yet) modified will be either the time at which the version was stored, or the time stamp that the file had at the time the version was stored; this depends on the **Putaway store file time** configuration option, see **AllChange Administrator Manual**.

Note that this is computed recursively, so that if *filea* depends on *fileb* which in turn depends on *filec* and *fileb* is deemed to be out of date with respect to *filec* then — assuming the actions performed to bring *fileb* up-to-date change its last modification time — *filea* will in turn be out of date with respect to *fileb*. **Build** maintains an internal record of file existence and modification times; furthermore, a file is marked as *up-to-date* once it has been built so it will not be rebuilt.

Build performs whatever actions are necessary to bring files back up-to-date, but it does not perform any actions it does not need to perform. The most frequent use of **Build** is to ensure that the correct modules are compiled and linked to produce an executable program. It can detect which module(s) have been changed since the last compilation was performed and recompile those *and only those* modules before relinking.

[Figure 8.1](#) illustrates the concepts of file dependencies and actions.

Figure 8.1: Build dependencies and actions #1



The example is for a simple software product written in C — the principles should hold for many other programming languages and applications. Here, source files `file1.c` and `file2.c` are *compiled* to produce object files `file1.obj` and `file2.obj` respectively, which are then *linked* to produce executable file `prog.exe`. Therefore `prog.exe` *depends on* `file1.obj` and `file2.obj`, which in turn *depend on* `file1.c` and `file2.c` respectively. If asked to build `prog.exe`, **Build** will first ensure that both `file1.obj` and `file2.obj` are up-to-date. If, and only if, the source file is *newer* than the object file (i.e. a different version of the source is required than the last time it was compiled or, if it is checked out for edit, it has been edited since it was last compiled) it will be (re)compiled. Then if, and only if, *either* object file is *newer* than the executable file (i.e. it has been (re)compiled since the last link) the objects will be (re)linked.

Build builds each of the *files* given as arguments. If none are specified, it builds the target in the first *dependency entry* encountered in the buildfile — see [The Buildfile](#).

The **AllChange** build makes use of an external tool to actually perform the operations necessary to do the build. The external tool is known as the Build Tool and **AllChange** provides it with the information about part versions that it requires in order to calculate what needs rebuilding and in order to create the derived object Build Threads, see [Build](#).

How Build Works

This section presents a description of how **AllChange build** works in general algorithmic terms. Building is a complex process; it is hoped that this information may clarify certain areas.

1. Call the external **Build** tool to generate a list of all non-derived (i.e. source) files which potentially might be involved in the build. The **Build** tool responds by placing the names of all entries in the buildfile — as they appear in the buildfile — which have no dependents or actions into a file named `depends.ac`.
2. Output the desired BT (`desired.ac_bt`), as follows (its contents are essentially lines giving a linked pair of *workfilename -version*):
 1. Read in filenames from `depends.ac` file.
 2. Read in lines from existing desired BT file, if present.
 3. For each filename in an existing desired BT file, if the filename is still among the required (non-derived) dependents use the existing information to provide the corresponding **All-Change** part and append the current default version .
 4. For each remaining filename, try to find a corresponding **AllChange** part by: descend the **All-Change** parts hierarchy from the current workspace 's associated part downwards; for each component compute the corresponding VC filename (location); generate a corresponding workfile name (taking into account whether the workspace is hierarchical); search the list of (non-derived) dependents for a matching workfile (making any relative names absolute and squeezing out any leading ". " or ". ."). If a corresponding part is discovered append the current default version.
 5. Output any remaining filenames with no corresponding part/ version.
3. Output the pools used by this workspace to a file named `pooldirs.ac`. This file will be read in by **Build**: it assigns `VPATH` to a list of the directories associated with the workspace's pools (as stipulated in the workspace's `register` file, if any).
4. Call the external **Build** tool to perform the build. This proceeds as follows:
 1. Read in the normal `buildfile` and `build.ini`, plus the `desired.ac` and `pooldirs.ac` files.
 2. For each target to be built, try to find an existing BT file for that object which satisfies the current build request in all respects. Do this by searching for the object and its BT (as a pair), first relative to the current directory, then relative to each pool directory in turn (in the order they appear in the `register` file). If an object is not found in one place in the search — or if its BT is not found or is not perfectly acceptable — proceed to the next possibility in the search.

- Derived objects — those having dependents and/ or actions in the buildfile — *must* be found with an acceptable BT to be used. The BT must contain all the same dependents, actions, and further dependents' BTs as would be generated by the current build; at the bottom level this includes the correct versions of all non-derived objects.
 - Non-derived objects — those having no dependents or actions in the buildfile — *must* be found with a BT if the `depends.ac` file shows a corresponding **AllChange** version was found; the BT must show the correct version. Conversely, a BT must *not* be found with a file if the `depends.ac` file shows that no corresponding **AllChange** version was found; such an object is used as a "black box", i.e. it is always acceptable.
 - The "correct version" of a non-derived object means the version-id in the BT must be the same as the desired version-id *and* the editing status must be the same (either both checked out for edit or both not checked out for edit).
 - Whenever an object depends on a non-derived object which is checked out for edit **Build** reverts to comparing the two objects' time stamps to decide whether the derived object needs building.
3. If found use that object and its BT. If not found, recursively build each dependent, then execute actions to build this object and output a new BT for this object including object's actions, dependents' BTs etc.

The Buildfile

About The Buildfile

The buildfile is a text file, either set up by the user with an editor or produced by the **Autobuild** tool — see **Autobuild** — which defines the dependency relationships and translation rules necessary to perform a build.

Entries in the buildfile must conform to one of 4 kinds:

Comment

Lines beginning with a # (hash) are ignored by **Build** up to the end of the line and may be used to insert comments.

Directive

Directives are used to which sections of the buildfile are used, see [Buildfile Directives](#). A line beginning with a ! (exclamation) denotes a directive.

Macro Definition

Macros are used to allow descriptive words to be used in place of other text, see [Build Macros](#). A macros is defined by a line of the form:

wordword or text = := text

Dependency

Dependency entries define the relationships between different files and the actions that are needed to bring a file(s) up-to-date.

A dependency entry has 3 parts which are specified in the form:

*targets : dependencies
actions*

Where:

- targets:* one or more filenames which are targets which may need (re)building. These should be specified words (filenames) separated by spaces preceding the colon. If more than one line is needed a \ (backslash) character must be placed at the very end (i.e. no trailing spaces) of all but the last line.
- dependencies:* zero or more filenames on which the target depends. These should be specified words (filenames) separated by spaces following the colon. If

more than one line is needed a \ (backslash) character may be used as above. If any of these files is found to be newer than the target then the target will be judged to be out of date and will be (re)built. There need not be any dependencies.

actions: zero or more lines which express an operating system commands which are to be invoked to bring the target *up-to-date* Each line should be indented with one or more spaces or tabs and should follow the targets and dependencies. These lines are sent to the operating system (without the spaces/tabs and after interpretation of any special characters — see [Action Line Modifiers](#)) whenever the target(s) are to be built. Each line is sent individually; the backslash character may be used as above for multiple lines. The first line which does *not* begin with a space/ tab or # (for comments) ends the dependency entry. There need not be any action lines.

You should note that:

- the : (colon) character is not followed by an =;
- the space before the colon is optional;
- the space before any dependencies is compulsory.

However, spaces are recommended for clarity anyway.

[Figure 8.2](#) gives a buildfile for the example of [Figure 8.1](#); note that [Figures 8.5](#) and [8.6](#) later in this section expand and improve on this.

Figure 8.2: Example buildfile #1

```
# This buildfile builds prog from file1.c and file2.c
prog : file1.o file2.o      # prog's dependencies
    cc -o prog file1.o file2.o
file1.o : file1.c          # file1.o's dependencies
    cc -c file1.c
file2.o : file2.c          # file2.o's dependencies
    cc -c file2.c
```

Comments may appear as actions and these will be output as such to the BTs. Comment actions may be specified with a # (hash) character as the first character of an action line. These will not be executed as actions, but will be output to the BT .

This facility may be used to cause any external information affecting how an object is built to be included in the BT and thus affect whether the object is to be rebuilt. An example of this might be a compiler which uses environment variables for determining compilation flags instead of/ as well as passing options on the command line. The environment variables used may be placed in a comment action as macros: these will then be output to the BT and hence have the same effect as if they were passed on the command line.

Suppose, for example, a C compiler uses an environment variable called `CFLAGS` as the flags to be used for compilation. It is therefore important that the value of this environment variable at the time of compilation should be included in the BT for future reference and comparison. The external **Build** utility makes environment variables available as macros with the same name, so the rule for compiling C source files could read:

```
.c.o:
    # $(CFLAGS)
    $(CC) $<
```

All filenames appearing in a buildfile should comply with the rules specified below if the buildfile is to be used by an **AllChange** build using Build Threads.

1. All filenames for which there is a corresponding part *must* be specified with relative path names. These will be taken to be relative to the part corresponding to the workspace in which the build is occurring; this in turn means that all workspaces that are to use a single buildfile must have the same part and hierarchical flag settings.
2. Any filenames with an absolute path will not be recognised as parts and will be taken to be files outside of the **AllChange** system. These will be accepted as a "black box" (i.e. no BTs involved); The **Build Tool** will revert to comparing time stamps for any objects which depend on such files.
3. Filenames *must* reflect the hierarchical or flat nature of the workspaces in which the buildfile is to be used and hence the corresponding locations of the files.

Special Targets

There are a few "special" (or "dummy") targets which may be defined if desired, either in a buildfile or in build.ini see [Build Builtin Rules and Macros](#). These do not name genuine targets; instead they influence **Build**'s behaviour. They are:

- . **DEFAULT** if this entry appears its actions will be used when a target must be built and there are no actions for it see later on.
- . **DONE** if this entry appears **Build** builds its dependencies and executes its actions *after* building any genuine targets (including if an error occurs).
- . **IGNORE** if this target is defined **Build** will not error when an action line returns a non-zero exit code see later on.
- . **INIT** if this entry appears **Build** builds its dependencies and executes its actions *before* building any genuine targets.
- . **SILENT** if this target is defined **Build** will not echo action lines as it executes them see later on.
- . **SUFFIXES** the dependents of this target specify the source suffixes used in *inference rules* see later on.

Build Macros

Macros are used in a buildfile to define abbreviations for longer text and may be used wherever the text is required.

Entries of the form:

word = *text*

are treated as a macro definition defining *word* to stand for *text*; the spaces on either side of the = are optional. *Text* continues up to a # character (for a comment) or until the end of the line. If more than one line is needed a \ (backslash) character must be placed at the very end (i.e. no trailing spaces) of all but the last line.

Macros may also be assigned at run time by completing the **Macros Defs** option.

Whenever

\$ (*word*)

is subsequently encountered in other macro definitions or anywhere in a dependency entry (target, dependency or action parts) it is replaced by its corresponding *text*, this is referred to as *macro expansion*. If the macro has not been defined it is replaced by an empty string. A macro reference is defined to be a \$ followed by a punctuation character (which normally has to be (). A \$ followed by a non-punctuation character is left unchanged, so as not to interfere with operating systems which allow filenames like \$disk1:.... \$\$ always expands to a single \$.

Macros should be used for items which may change over time or which are referenced more than once, and to increase readability of buildfiles, e.g.

CC = /bin/cc

```

CFLAGS = -O
TAR ET = prog
SOURCE = file1.c file2.c
$(TARGET) : $(SOURCE)
            $(CC) $(CFLAGS) $(SOURCE) -o $(TARGET)

```

To change what flags are passed to the compiler it is only necessary to change the value of `CFLAGS` once — either in the buildfile or from the command line — no matter how many times it is referenced.

Build Macro Substitution

If a macro reference of the form:

```
$(word:subst=repl)
```

is encountered the macro is expanded, but with all occurrences of *subst* being replaced by *repl*. Thus in the previous example `$(SOURCE:.c=.o)` would expand to `file1.o file2.o`.

If *repl* is omitted then all occurrences of *subst* are effectively removed. As a special case, if *subst* is omitted all occurrences of (one or more) spaces are replaced by *repl*. This is particularly useful under operating systems with commands expecting multiple filenames to be separated by something other than space, e.g.:

```

SOURCE = file1.c file2.c file3.c
printall : $(SOURCE)
          print $(SOURCE:=,)

```

would produce `print file1.c, file2.c, file3.c`.

If *subst* is just a single `^` (circumflex) then *repl* is inserted at the *beginning* of each word in the macro expansion. This is particularly useful for prepending an option string to each of a list of words to pass to a command. Suppose, for example, it is desired that a compiler search for include files in a number of directories and the compiler requires that this be specified by separate `-I` *directory* options. This could be achieved by:

```

INCLUDES = /dir1 /dir2 /dir3
file.o : file.c
        cc $(INCLUDES:^=-I) -c file.c

```

to produce

```
cc -I/dir1 -I/dir2 -I/dir3 -c file.c
```

Build Macro Expansion

Macros are expanded *recursively* and (normally) *when needed*, so

```

MACRO1 = $(MACRO2)
MACRO2 = file

```

works as expected. A normal macro assignment may thus not refer to itself, either directly or indirectly.

Using `:=` instead of `=` in a macro assignment causes the right-hand side to be expanded immediately, permitting:

```
MACRO := -a $(MACRO) -b
```

Internal Macros

There are several macros which have special significance to the build tool, these are referred to as *internal macros*

The internal macros are:

CWD is set (on initialisation) to the name of the current working directory

BUILDFLAGS is set (on initialisation) to the options passed to the **Build Tool** on the command line, excluding **Buildfile Filename** and **Print Information**, and is useful for invoking a new **Build** with the same options as the original one.

VPATH is used to define the directories the **Build Tool** should look in when searching for files — see [Search Path for Files](#) for further details.

NONCRITICAL is used to mark macros which are not to be expanded on output of actions to a BT.

The macros `$*`, `$@`, `$<`, `$?` and `$^` (see below) are automatically non-critical.

Further non-critical macros may be added by assigning the macro names to the `NONCRITICAL` macro as follows:

```
NONCRITICAL=macro-names
```

Since the `VPATH` macro is used to define the pools in which objects are to be found, this should normally be set as a non-critical macro as follows:

```
NONCRITICAL=VPATH
```

In addition to the macros above there are 5 internally maintained macros which are re-evaluated as each target is built and may be used in the actions part of dependency entries; they may be used in any kind of dependency entries, but are especially useful in inference rules — see below. The parentheses which usually surround a macro name may be omitted for these single-letter macros. To illustrate these macros, suppose a target and dependents line reading:

```
/path/file.o : file.c defs.h
```

is being evaluated. Then:

`$@` (at) stands for the filename of the current target, i.e. `/path/file.o`.

`$*` (asterisk) stands for the filename of the current target with the suffix deleted, i.e. `/path/file`.

`$<` (less-than) stands for the filename of the first dependent, i.e. `file.c`.

`$^` (circumflex) stands for the filenames of all of the current target's dependents, i.e. `file.c defs.h`.

`$?` (query) stands for the filenames of all of the current target's dependents which are *newer* than the current target, e.g. `defs.h` if it, but not `file.c`, had been altered more recently than `/path/file.o`.

The first three of these may have certain single letters appended to them (in which case the enclosing parentheses must appear, e.g. `$(@F)`) to stand for useful parts of the pathname they deliver. Suppose these are applied to `$@` (`/path/file.o`) in the previous example. Then:

- B** The basename (no path or suffix) part of the pathname, i.e. `file`.
- D** The directory part of the pathname, i.e. `/path`. (The current directory symbol is generated if the pathname has no leading path.)
- F** The filename part of the pathname, i.e. `file.o`.
- R** The rootname (all but suffix) part of the pathname, i.e. `/path/file`.

The sequence `$$@` used in the dependencies for targets is a *dynamic dependency* and may be used to refer to the current target as the dependency is generated for each target in turn. This allows simple dependency entries to be written like:

```
prog1 prog2 prog3 : $$@.c
      $(CC) $(CFLAGS) $< -o $@
```

which means that `prog1` depends on `prog1.c`, `prog2` on `prog2.c` and `prog3` on `prog3.c`.

Internal macros that are dynamically evaluated should not be assigned to explicitly in a buildfile.

Macro Priorities

The priority of macro assignments and rules from different sources in **Build** is as follows (lowest to highest):

1. Builtin macros and rules (i.e. those read from the `build.ini` file).
2. Environment variables (normal case).
3. Macros and rules encountered in the buildfile.
4. Environment variables (**Environment variables override** option used).
5. Command line macro assignments (**Macro assignments** option).

Search Path for Files

Build has a facility to search for files across a number of directories. Whenever a (non-absolute) filename is sought, **Build** first looks for it relative to the current directory; if it is not found then, if the internal macro `VPATH` exists, it is taken to hold a (space-separated) list of directories which are to be searched, in order, for the file. If the file is found along `VPATH` the new path to the file is used from then on; otherwise **Build** sticks to the filename with which it started. `VPATH` is searched for all kinds of files (targets, explicit and implicit dependents).

`VPATH` is used to indicate where the pool directories are located.

The use of pools is especially useful in multi-user environments where files may be shared in common pools, for example header or object files. Suppose common header files are held in a pool called `headers1` which is associated with the directory `\\server\share\pools\headers1`. If the pool is *registered* to the workspace (see [Workspace Registrations](#)) then `VPATH` will be exported to the **Build Tool** (via the file `pooldirs`) as:

```
VPATH = \\server\share\pools\headers1
```

a buildfile might then contain:

```
HEADERS = $(VPATH)
.c.o:
    (CC) $(CFLAGS) $(HEADERS:^=-I) -c $<
prog.o : defs1.h defs2.h
```

If a header file exists in the current (workspace) directory — presumably only the case if it is being edited — it will be found there; otherwise it will be found in the header pool.

Note the use of the `HEADERS` macro which is used to pass the appropriate include path command line option to the compiler, so that the compiler will also be able to find the header file.

Since `VPATH` is used to define the pool directories to be searched, this information must not be destroyed in your buildfile. If you wish to specify some directories for `VPATH` you must *add* them to the existing macro contents using the `:=` macro assignment operator.

When using `VPATH` it is particularly important to use **Build's** internal macros rather than an explicit reference when referring in an action to a file which might be found along `VPATH`.

Thus if `VPATH = c:\dir` then a buildfile:

```
DEPENDENTS = file1 file2
dummy : $(DEPENDENTS)
    echo $(DEPENDENTS)
```

could only ever echo `file1 file2`. The correct action line is `echo $^`; since `$^` is an internal macro, **Build** will ensure it contains `c:\dir\file1` and/ or `c:\dir\file2` if that is where the file was found.

Inference Rules

To avoid having to repeatedly spell out actions which are always the same for files which conform to a certain pattern — such as one stating the required action to translate a source file into the object file is compilation of the source file — **Build** allows *inference rules* to be stated giving a dependent file's suffix, a target file's suffix (if it has one) and required actions to bring the target up-to-date.

An inference rule is a dependency entry of the form:

```
source-suffix target-suffix :
    actions to produce target from source
```

except that there should be *no* space between the suffixes. An example for building object files from C source files is:

```
.c.obj :
    $(CC) $(CFLAGS) $<
```

If there is to be no target suffix it may be omitted. Note the use of the internal macros (\$< always stands for the first dependent when an inference takes place).

Build must know of all source suffixes involved in inference rules before it can infer properly. These suffixes are kept as the dependents of a dummy target called `.SUFFIXES`. To allow users to write rules containing new suffixes without having to deal with `.SUFFIXES`, whenever a rule is met the left-hand (i.e. source) suffix is automatically appended to `.SUFFIXES` if it is not already present.

Should it be necessary, to add additional suffixes, say `.z1` and `.z2`, to those already known by **Build** a dependency entry

```
.SUFFIXES: .z1 .z2
```

should appear in the buildfile. An entry of just `.SUFFIXES:` with no dependents empties the list of known suffixes. The order of `.SUFFIXES` determines the order in which a prerequisite is sought; only one prerequisite may be inferred.

If an entry for the dummy target `.DEFAULT` appears in the buildfile its actions will be used if a file must be built but it does not exist, there is no explicit entry and no inference rule applies; otherwise an error is issued.

Action Line Modifiers

The normal behaviour of **Build** as it encounters an action line may be modified by starting the line (*after* the indentation) with one or more special punctuation characters or by other means, as follows:

1. A line is executed just once, except if it has `!` (exclamation) as its *first* character after the indent, in which case it is performed *repeatedly for each filename in turn contained in a macro*. There are two possible forms for such a line:
 1. A line beginning with a *double* `!!` takes the first word following as the *name* of a macro (so it should not be enclosed in `$()` and should have a following space) and executes the rest of the line repeatedly for each (space-separated) filename in turn in the macro's expansion, replacing a reference to the macro by the filename each time around, e.g.:

```
PROGS = aprog.exe bprog.exe cprog.exe
install :
    !!PROGS copy $(PROGS) \PUBLIC
```

could be used under Windows to copy each executable file one at a time (since `COPY` does not accept multiple source files except via a wildcard).

2. A line beginning with a *single* `!` does not take the first word following as the name of a macro; instead the action must refer to either `$$^` or `$$?`, and the action is performed for each filename in that in-built macro's expansion. The previous example would copy all execut-

ables every time `install` is built; this could be improved to ensure the executables are rebuilt and then copy only those which have changed since the last install by:

```
install : $(PROGS)
        !copy $? \PUBLIC
        update $@
```

Any further special punctuation characters described below must come after the `!` or `!!` *macro-name*.

2. An action line is *not* executed if the **No execute mode**, **Question mode** or **Touch (Update) targets** options are used *unless*:

1. the line starts with a `+` (plus) character; *or*
2. the line contains a reference to the macro `$(BUILD)`.

This is very useful to see what would happen in recursive build situations with the **No execute mode** option, e.g. under Windows:

```
all :
      +cd \DIR1
      $(BUILD) $(BUILDFLAGS)
      +cd \DIR2
      $(BUILD) $(BUILDFLAGS)
```

3. A line is echoed to the screen before being sent to the operating system unless:
 1. the line starts with a `@` (at) character; *or*
 2. a dummy dependency entry of just `.SILENT:` appears in the buildfile; *or*
 3. the **Silent mode** option is used.
4. If the operating system returns a non-zero exit code from any action line then **Build** will exit immediately unless:
 1. the line starts with a `-` (minus) character; *or*
 2. a dummy dependency entry of just `.IGNORE:` appears in the buildfile; *or*
 3. the **Ignore error codes from actions** option is used; *or*
 4. the **Continue after error code** option is used.
5. A line beginning with a `%` (percent) is executed internally to the **Build** instead of being sent to the operating system.

Currently the only built-in action is `cd` (e.g. `%cd d:\tmp` in Windows). This is most useful because under Windows the operating system `cd` command has no effect on current drive/ directory of **Build** or subsequent actions. If a drive is included in the path specification it becomes the current drive.

Creating External Files

It is sometimes useful to be able to create external files from within **Build** containing information related to the build. Let us look at two examples which take advantage of action line modifiers and complex macro substitutions to achieve this.

First, suppose we wish to create a file containing the names of all source files, one per line (for use with, say, an external program). The following buildfile will do this:

```
CFILES = file1.c file2.c ...
HFILES = defs1.h defs2.h ...
SOURCES = $(CFILES) $(HFILES)
fetchfile :
    @-DEL -f $@
    !!SOURCES echo $(SOURCES) >> $@
```

`fetchfile` is first removed, then each file in `SOURCES` is appended to it. `fetchfile` could be made to depend on `buildfile` so that it is recreated whenever the buildfile is changed (which would be the case if the list of source files changed), or it could be built from time to time with the **Unconditionally build** option.

Second, suppose we wish to create a linker response file (i.e. a file containing the required responses to the linker's requests for filenames etc.), which is often needed when the operating system has a short command line length, like MS-DOS. Here the technique is complicated by the fact that we may pick up some of the object files along `VPATH` and by the need to place each object file on a line of its own with all but the last line terminated with a `+` symbol:

```
VPATH = \OBJECTS
OBJECTS = f1.obj f2.obj f3.obj f4.obj
LNKOBJECTS = $(^: =+ )
prog.lnk : $(OBJECTS)
    @-del $@
    !!LNKOBJECTS echo $(LNKOBJECTS) >> $@
```

The critical macro is `LNKOBJECTS`: it is set to `$(^` — which will evaluate to all the dependents of `prog.lnk`, i.e. all object files, with `\OBJECTS` prepended to those found there — with the space separator *between* each filename changed to a plus symbol followed by a space, so it might end up evaluating to:

```
f1.obj+ \OBJECTS\f2.obj+ \OBJECTS\f3.obj+ f4.obj
```

The `!!` line then appends, in turn, each (space-separated) word in this macro to the linker response file. If the linker requires other response lines these could also be echoed into the response file before or after the object files.

Buildfile Directives

indexdirectives

Build recognises *directives* while reading buildfiles. A directive has a `!` (exclamation) in column 0, optionally followed by whitespace, and then the (case-insensitive) directive word (cf. **Prep**). Recognised directives are:

!include *buildfilename*

Include the specified buildfile at this point in the current buildfile. Generally useful macros, rules etc. may thus be included in all buildfiles. Included files may themselves contain further `!includes`. *Buildfilename* may contain macros which are expanded. If the filename is relative it is sought first in the current directory; if not found there it will be sought along the (space-separated) directories named in a special macro called `INCPATH`, which functions like `VPATH`. A fatal error is raised if the file cannot be read; if *buildfilename* starts with a `-` (minus) just a warning is issued.

!else

Reverse the sense of the current `!ifdef` or `!iftarg`.

!endif

End the current `!ifdef` or `!iftarg`.

!ifdef *macro-name*

Starts a conditional section of the buildfile till the matching `!endif`. *Macro-name* should be specified without any `$`, `(` or `)`. If *macro-name* is defined then process lines as normal up to the matching `!else` or `!endif`, otherwise ignore them. `!ifdefs` may be nested within one another.

!iftarg *target*

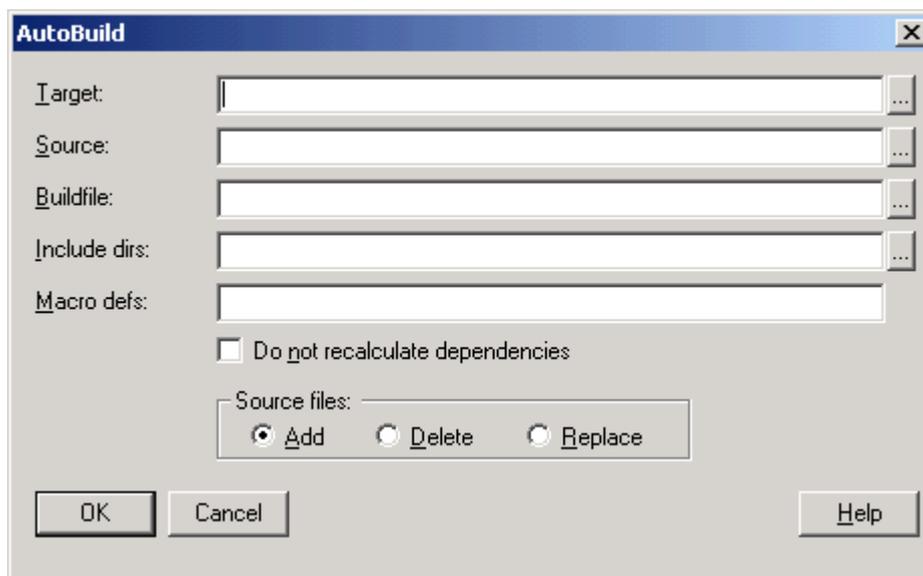
Like `!ifdef`, but tests whether *target* is among the top-level target(s) to be built, either passed to **Build** on the command line or the first genuine target in the buildfile.

How to Set Up a Buildfile

About How to Set Up a Buildfile

A *buildfile* for **Build** may be set up and maintained using the **Autobuild** tool.

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. **Autobuild** may then be invoked from **AllChange** using **File | Build | Autobuild**.



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

The **Target** option is used to specify 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.

The **Source** option should specify the source files on which the **Target** ultimately depends.

The **Buildfile** option may be used to specify an alternative name for the *buildfile* generated. If none is specified then `buildfil` will be created.

The **Include dirs** option 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.

The **Macro defs** option Allows macros to be defined in the same way as for the build tool, see Build

If **Do not recalculate dependencies** is selected, this 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.

The **Source Files** options determine how the source files should be acted upon. 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**.

Autobuild infers the required intermediate files, e.g. object files, that lie between sources and target; the Examples section illustrates what to do if the programming language does not have any intermediate stage.

Autobuild must first locate a *build template* file, by default called `build.tpl`. This file gives a template of what the final buildfile should look like, plus some information to help **Autobuild** calculate dependencies. When creating a new buildfile, **Autobuild** essentially copies it into the buildfile, adding further lines appropriate to the specified *target* and *files*.

When using **Autobuild** to create the buildfile, all target dependents (usually object files) are defined in a macro. The actions for each target dependent may then be defined once using these macros.

For example, for a target of prog:

```
$(PROGOBJECT) :
    $(CC) $(CFLAGS) $<
```

See [The Buildfile](#) for more information.

The Build Template

About The Build Template

The build template describes a template of what the final buildfile created by **Autobuild** should look like, plus some information to help **Autobuild** calculate the dependencies.

By default the template file is called `build.tpl`. **Autobuild** will look for the template file in the current directory or the project directory or system directory.

The template file is divided into four sections, separated by a line of the form `---` (3 minuses) as follows:

```
source_file_extension object_file_extension preprocessor_command
include_file_string include_dir include_file_extension
include_file_string include_dir include_file_extension
```

```
.
.
.
```

```
source_file_extension object_file_extension preprocessor_command
```

```
.
.
.
```

```
---
TARGET : $(OBJECT)
instructions for making TARGET from $(OBJECT)
```

```
.
.
.
```

```
---
lines to be copied to start of resulting buildfile
```

```
---
```

```
lines to be copied to end of resulting buildfile
```

Where the sections are:

1. Informs **Autobuild**, for each source file with a particular extension what extension the corresponding object file will have; e.g. a `.c` file will produce a `.obj` file.

It also informs **Autobuild** what strings to scan for in the source files of that type to determine the other files used by (files included in) that source file; e.g. for C source files, the include string might be `#include "`.

Up to 10 include file strings may be specified for each source file extension.

This information is used to generate the required dependency information in the resulting buildfile.

See [Source/ Include File Section](#) for further details

2. Informs **Autobuild** how *targets* are built from objects (or sources); e.g. how to link object files together to create an executable program.

A modified copy of these lines will be added to the buildfile immediately before the dependency lists each time a new target is added to the file via **Autobuild**. This section is only used when creating a new buildfile or adding a new target to an existing buildfile.

See [Target Translation Rules Section](#) for further details.

3. Defines any lines to be copied into the *start* of the resulting buildfile, e.g. macro definitions.
4. Defines any lines to be copied into the *end* of the resulting buildfile, e.g. generally useful rules, such as how to install, backup or delete files. These lines will come before any lines generated by the rules given in the second section.

When constructing the buildfile, **Autobuild** will insert macro definitions for `TARGETS`, `OBJECT` and `SOURCE` between the third and fourth sections, as well as macros giving the source and object file lists for each target being constructed.

The fourth section should give any lines to be copied into the *end* of the resulting buildfile, e.g. generally useful rules, such as how to install, backup or delete files. These lines will come before any lines generated by the rules given in the second section. A common first line for this section is:

```
all : $(TARGETS)
so that build all will build all programs.
```

After the fourth section, rules constructed from the template given in the second section of the `build.tpl` file, specifying how to construct the target from its object files, will be inserted.

Autobuild will also append lines for the dependencies of each object file at the end of the buildfile.

Source/ Include File Section

This is the first section of the template file which comprises entries of the form:

```
source_file_extension object_file_extension preprocessor_command
include_file_string include_dir include_file_extension
```

Each field within a line should be separated from the next by a tab character. Up to 10 lines of include file strings may be specified per source file extension.

A blank line must be inserted to separate one entry from the next.

The first two fields (*source_file_extension* and *object_file_extension*) are compulsory and should specify the extensions (i.e. suffixes) appearing at the end of source and object files respectively, e.g. `.c` and `.obj` for the C language.

If the compiler can produce object files with different extensions according to options used, multiple object file extensions may be specified separated by `;` (semi-colon): the first will be the one used in the `OBJECT` macro definitions but they will all be used on the left hand side of the dependency lists.

Autobuild generates object filenames from source filenames by replacing *source_file_extension* with *object_file_extension*. All files passed to **Autobuild** on the command line *must* bear extensions which appear as a *source_file_extension*.

Files found to be included in source files will appear in the dependencies for the source file's corresponding object file(s). Normally, source files are searched for include file directives. However, source files with *conditional* file inclusion, e.g. C file containing:

```
#if MACHINE1
#include "file1"
#else
#include "file2"
#endif
```

should be run through the preprocessor with the appropriate flags to determine which files are actually included. The output from the preprocessor will then be searched for the specified include file directives.

The *preprocessor_command* is optional: if specified it will be used to preprocess each source file before it is searched for any include files. It should specify the complete command to preprocess the source file and must contain %s in place of both the source and the preprocessor output filenames (in that order). For example, the command to execute the C preprocessor on each file might be:

```
cl -E %s > %s
```

The preprocessor output file will be deleted by **Autobuild**.

Autobuild exports macros to the environment prior to executing the preprocessor command in the same way as **Build**; the special macros `EXPORTENV` and `INIOVERRIDEENV` are also handled in the same way — see [Build Environment](#). This facility may be used to affect the behaviour of the preprocessor as it would be affected when called from **Build**.

The include pattern lines are optional and must immediately follow the source file patterns line. The *include_file_string* defines the string used in a source file in order to include another file. The *include_dir* is optional and is used to specify any directories (instead of the current directory) in which to search for any include files. The *include_file_extension* is also optional and specifies the suffix that include files have.

Autobuild will scan each source file (after preprocessing if appropriate) for the *include_file_string* and, if found, will take the word immediately following the string (ignoring whitespace and punctuation characters) as the name of the file being included (after possibly replacing any suffix with the *include_file_extension* specified). Any strings of space characters in the *include_file_string* will match any strings of whitespace (space or tab) characters in the file, and any occurrences of * (asterisk) will match any string of characters in the file.

The scan is case-sensitive, so it may be desirable to specify alternative include file strings which differ, say, in capitalisation if the source files may contain such variations. If no preprocessor is being used, **Autobuild** will scan included files for any *further* included files, recursively; it will also remember these to avoid having to rescan them, which will save a fair amount of time.

For example, in C, the include file string, if no *preprocessor_command* is being used, might read:

```
#include "
```

On the other hand, if the C preprocessor is to be run as the *preprocessor_command* it will replace such include lines by:

```
#line line-number"filename"
```

so the include file string should be:

```
#line * "
```

Some preprocessors and C compilers provide an option to produce just a list of the filenames included: if this is available then the corresponding *include_file_string* should be a single * (asterisk) which will match an empty string.

If the preprocessor used does not leave any signature when a file is included then an appropriate comment should be inserted which can be identified by **Autobuild**. For example, a Pascal source file that is to be included in another source file by the **Prep** (see [Preprocess Tool](#)) tool might start with the line:

```
{ prep include file: @FILE }
```

and the include file string would be:

```
{ prep include file:
```

include_dir is optional, and may only appear if *include_file_string* does too: it should give the directory path(s) in which include files are sought. Multiple paths are separated by ; (semi-colon) or , (comma). If no *include_dir* is present and the **Include directories** option is not used the current directory is searched; the current directory may appear explicitly in the list of directories.

include_file_extension is optional, and may only appear if *include_dir* does too: it is used if the include commands do not include a suffix for the file included.

Target Translation Rules Section

This section should contain lines of the form:

```
TARGET : $(OBJECT)
        instructions for making TARGET from $(OBJECT)
```

(Note that `TARGET` is *not* enclosed in `$()`.)

A modified copy of these lines will be added to the buildfile immediately before the dependency lists each time a new target is added to the file via **Autobuild**. Each occurrence of `TARGET` is replaced by the name of the target, `OBJECT` by the name of the object files macro and `SOURCE` by the name of the source files macro — see the section on Updating buildfiles below. It is only used when creating a new buildfile or adding a new target to an existing buildfile.

Search Path for Files

Autobuild will search the directories specified with the Include directories option, and those specified in the include pattern specifications in the `build.tpl` file.

Autobuild will also use the `VPATH` macro (see [Search Path for Files](#)) when it is updating an existing buildfile (see [Updating Buildfiles](#)). It will search this list of directories in a similar way to **Build** for any source and header files. If any file is found along `VPATH` then it is entered into the buildfile without any matching leading path so that build will also look along `VPATH` to locate the file used.

The `IGNOREPATH` macro is similar but any files found along this path will be omitted from the buildfile. This allows system libraries etc to be omitted thereby keeping the buildfile down to a reasonable size.

Any macro references in the preprocessor commands (if used) will be expanded before the command is run, thereby allowing the command to use include directory lists and similar defined within the buildfile. Macros may be defined as a command option in the same way as **Build**.

The include directory list (taken from the `build.tpl` file and command line) may also include macros to be expanded before searching. Macro expansion will take place for each include found and therefore after the list of directories has been split up. The `$(@)` dynamic macro may be used to enable the included file to be found relative to the including one. For example, if all header files are found either in the same directory as the including source files, or in a subdirectory called `headers` then the include pattern line might look something like:

```
#include " $(@D);$(@D)/headers
```

Updating Buildfiles

If **Autobuild** is run in a directory already containing a buildfile then instead of creating a new buildfile **Autobuild** will update the old one, recalculating dependency lists. The old buildfile will be retained as *buildfile.old*. No alterations or additions should be made to the last section of the buildfile as these will be lost the next time **Autobuild** is used to update the file.

If a target is specified when updating an existing buildfile then:

- if this is a new target it will be added to the buildfile, along with its source files
- if the target already exists in this buildfile then its source (and object) file lists will be updated.

For speed, only dependencies for the specified target are calculated. If *no* target is specified, dependencies for *all* targets are recalculated. In all cases the `SOURCE` and `OBJECT` macros and the dependency lists for the entire file will be updated.

In order for **Autobuild** to maintain a buildfile it must be set out in a particular way. Whenever **Autobuild** creates a new buildfile it will ensure that it conforms to these rules, but existing buildfiles may have to be modified before **Autobuild** can be used to update them. **Autobuild** will only recognise (and act on) the following macro definitions, which must appear in the following order:

1. `TARGETS` contains a list of all the target names in the buildfile.
2. Macros of the form `TARGETNAME``SOURCE` that contain the source file lists for each target. These can be updated with **Autobuild** by specifying the corresponding target and the affected source files: if the **Delete source files** option is specified the files are removed from the macro; if the **Replace source files** option is specified the current files are replaced by the new ones; otherwise the new files are added to the current ones. Alternatively, the source filenames can be edited by hand and **Autobuild** rerun — this is probably the simplest way of renaming or removing a file in the list.
3. Macros of the form `TARGETNAME``OBJECT` that contain the object file lists for each target. Each of these will be updated to correspond to the source file macro whenever **Autobuild** is used.
4. `SOURCE` is used by **Autobuild** to record *all* the source files for all targets. The `SOURCE` macro definition must come after any of the above macro definitions and is essential for **Autobuild** to usefully maintain the buildfile.
5. The `OBJECT` macro definition must come immediately after the `SOURCE` macro and is maintained as a list of the object files corresponding to each of the files in `SOURCE`.

Note that these macro definitions may not refer to other macros.

Autobuild recognises one further macro. Normally object files are assumed to reside in the same directory as their corresponding source file. If the special macro `OBJISINCWD` is defined *before* any of the above macros all object files are assumed to reside in the current directory instead.

Equivalent Filenames

Normally **Autobuild** will deduce the name of an object file by looking for a `source_file_extension-object_file_extension` rule corresponding to the extension (suffix) of the source file and replacing the `source_file_extension` with the `object_file_extension` given in the rule. However there are instances when this may not be sufficient: for example, if the source file `afile.c` is compiled with one set of `CFLAGS` to produce `afile1.o` when compiling to produce `target1`, and another set of `CFLAGS` to produce `afile2.o` when building `target2`. In this case it is necessary to provide **Autobuild** with a list of equivalent object filename *roots* for each such source filename. This is done by including in the buildfile rules of the form:

```
.EQUIVS : source-filename object1-root object2-root ...
```

So for the above example this would be:

```
.EQUIVS : afile.c afile1 afile2
```

These lines must appear near the head of the buildfile (before any source or object file list macro definitions).

Whenever **Autobuild** needs to construct an object filename from a source filename any `.EQUIVS` lists are checked first. If a list matching the source filename is found one of the object filenames given in the list is chosen by the following rules and the appropriate object file extension added:

1. If an existing target is being updated and one of the equivalent object filenames is already given for that target **Autobuild** will leave the existing choice untouched.
2. If a matching object name was given on the command line (with the **Object filenames** option) then that name will be used.
3. If neither of the above applies the first name in the equivalence list is chosen.

This facility relies on finding `.EQUIVS` lists in the buildfile. If a new buildfile is being created and this facility is required the buildfile should be created normally (with **Autobuild**) and then edited to include the correct `.EQUIVS` and object filename lists.

Autobuild Examples

Assume buildfile does not already exist in the current directory. [Figure 8.3](#) shows the `build.tpl` file used.

Figure 8.3: Example build.tpl file

```
.c      .o cl -E %s > %s
# * "
---
#      How to build TARGET
TARGET : $(OBJECT)
        $(CC) -o $@ $(OBJECT)
        $(LDFLAGS)
---
# buildfile constructed from tem-
  plate
LDFLAGS =
---
all : $(TARGETS)
```

If `source1.c` contains the following lines:

```
#include "general.h"
#include "header1.h"
#include "source1.h"
```

and `source2.c` includes the lines:

```
#include "general.h"
#include "header2.h"
#include "source2.h"
```

then **Autobuild** invoked with a **Target** of `thisprog` and **Source** files of `source1.c` and `source2.c` will produce the buildfile shown in [Figure 8.4](#).

Figure 8.4: buildfile produced by Autobuild

```
# buildfile constructed from template

LDFLAGS =

### The following lines are automatically defined
### - be careful about altering them
TARGETS = thisprog
THISPROGSOURCE = source1.c source2.c
THISPROGOBJECT = source1.o source2.o
SOURCE = source1.c source2.c
OBJECT = source1.o source2.o
### End of automatic lines

all : $(TARGETS)

#      How to build thisprog
thisprog : $(THISPROGOBJECT)
          $(CC) -o $@ $(THISPROGOBJECT) $(LDFLAGS)

### DO NOT REMOVE THIS LINE
```

```

### Do NOT make any alterations after this line
### Any changes will be lost on the next use of autobuild
source1.o : source1.c ./general.h ./header1.h ./source1.h
source2.o : source2.c ./general.h ./header2.h ./source2.h
### END OF FILE

```

Any minor adjustments to this buildfile can now be made with a text editor and from now on a **Build** command will perform any necessary compilations and/ or linkings. Whenever changes have been made to the `#include` lines in any of the source files the dependency lists can be updated by invoking **Autobuild** with no **Target** or **Source** files.

If **Autobuild** is invoked with a **Target** of `thisprog` and **Source** of `source3.c` then an additional source dependency file for `thisprog` will be added and the dependencies recalculated

If, on the other hand, **Autobuild** is invoked with a **Target** of `newprog` and **Source** of `newsrcl.c newsrcl2.c newsrcl3.c` then this will add a second target (`newprog`) to the buildfile, along with its source files and dependencies.

For a different example, suppose a Pascal compiler is being used, where:

- source file suffix is `.pas`;
- object file suffix is `.bin`;
- there is no conditional compilation and/ or preprocessor;
- files with a `.inc` suffix may be included by `{$include file}`, where the include may be written in upper or lower case and the `.inc` suffix omitted;
- included files are sought first in the current directory, then in `c:\include`.

Furthermore, some source files are assembler files, where:

- source file suffix is `.asm`;
- object file suffix is `.bin`;
- there is no facility for file inclusion in the assembler.

For the above example, the first section of the build template file should read:

```

.pas .bin
{$include      .;c:\include .inc
{$INCLUDE     .;c:\include .inc

.asm      .bin
---
```

Some programming languages, e.g. Dataflex, compile directly from a single source file — which may have include files — directly to an end result without any intermediate object file. In this case:

- the end result is treated as an object file;
- there are no target files;
- references to targets should be ignored;
- the second section of the build template file should be empty and the **Target** option should not be used.

A minimal template file (assuming the rule for going from a `.frm` to a `.flx` file were specified in the `build.ini` file read by **Build**) could read:

```

.frm .flx
#include
---
---
---
all : $(OBJECT)
```

Autobuild with **Source** of `file1.frm file2.frm file3.frm` would then produce the necessary buildfile for constructing `file1.flx`, `file2.flx` and `file3.flx` from their corresponding sources (and any include files).

Autobuild Notes

The buildfile produced may be edited by the user to include extra facilities, remove redundant ones, and so on. Providing this editing is carried out carefully then **Autobuild** can still be used to maintain the dependency lists etc. from then on. Any editing must be restricted to the buildfile above the line:

```
### DO NOT REMOVE THIS LINE
```

which marks the start of the dependency lists. Lines should *not* be inserted between the macros defined and maintained by **Autobuild**.

Autobuild does not recognise **Build** directives, e.g. `!if`, `!include`.

Build Builtin Rules and Macros

Build has a number of *builtin* rules and macros appropriate to the host operating system which it reads in from a buildfile called `build.ini` when it starts up. It looks for this file first in the current directory, then in the project directory, then in the system directory. Builtin rules and macros are included in the output from the **Print macros and targets** option.

If all the information required to build a target from a dependent is already found to be in-built, e.g. how to build an executable program from a C source file, it is permissible to enter simply:

```
build target
```

having created an empty buildfile.

Build Environment

If the operating system supports environment variables these are read by **Build** and processed as though they were macro definitions. They are normally processed after the internal inference rules read from the `build.ini` file but before the buildfile. Hence macro assignments in the buildfile override those in the environment, unless the **Environment variables override** option is used. Furthermore, if the special macro `INIOVERRIDEENVS` exists it is taken to hold a (space-separated) list of the names of macros for which a definition (if any) in the `build.ini` should override a definition (if any) in the environment.

If the value of a macro with the same name as an environment variable is altered while reading the buildfile the environment variable is altered to this value too. This is especially useful under MS-DOS where the limited length of a command line can prevent options being passed to a sub-program other than through an environment variable. Say, for instance, the compiler looks for an environment variable called `INCLUDE` to pass a long list of semi-colon separated directories to search for included files. Suppose you already have a `INCLUDE` environment variable set, e.g. under MS-DOS:

```
SET INCLUDE=C:\INCLUDE
```

before calling **Build**, but wish to extend this list in a particular buildfile. You may put the following line into the buildfile (MS-DOS):

```
INCLUDE := $(INCLUDE);C:\A\LONG\PATH;D:\ANOTHER\PATH
```

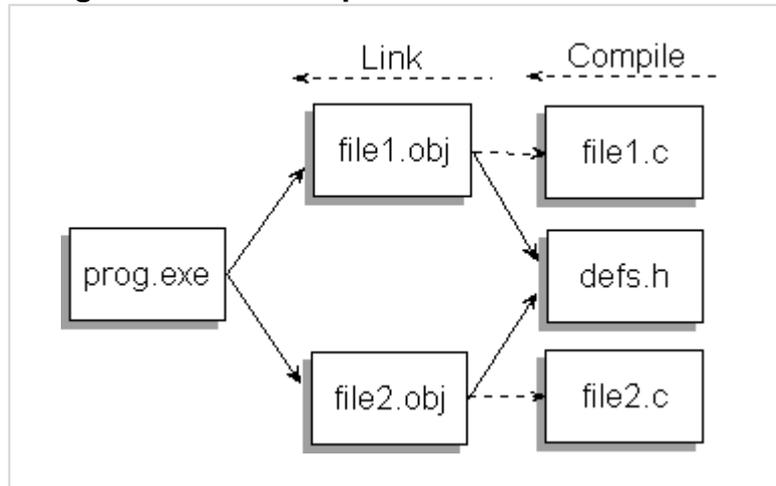
and **Build** will export the redefined environment variable to all commands it calls.

Build only usually exports those macro definitions derived from environment variables which already existed when **Build** was invoked (otherwise the environment might get very large); an environment variable may be defined to an empty value before invoking **Build** to force it to be exported. In addition, if the special macro `EXPORTENVS` exists it is taken to hold a (space-separated) list of the names of macros that should be exported to the environment regardless of whether or not they already existed in the environment when **Build** was invoked.

Example Buildfile

Figure 8.5 gives the dependencies and actions for constructing an executable program, `prog`, from two C source files, `file1.c` and `file2.c` which both include a header file, `defs.h`.

Figure 8.5: Build dependencies and actions #2



The buildfile given in Figure 8.6 could be used in this example. The rule to build an object file (suffix `.o`) from a C source file (`.c`) and various macros have been included for clarity, though in practice they may already be known to **Build** from the `build.ini` file.

Figure 8.6: Example buildfile #2

```

# This buildfile builds prog from file1.c and file2.c

CC = cc                # invoke C compiler/linker
CFLAGS = -O            # -O for optimisation
LDFLAGS =              # no linker flags
SOURCE = file1.c file2.c # C source files
OBJECT = $(SOURCE:.c=.o) # change .c to .o

.c.o:                  # rule to build .o from .c
    $(CC) $(CFLAGS) -c $<

prog : $(OBJECT)       # link OBJECT to give prog
    $(CC) $(LDFLAGS) -o $@ $(OBJECT)

file1.o : file1.c defs.h # file1.o's dependencies
file2.o : file2.c defs.h # file2.o's dependencies
  
```

Note that since the action part of the dependency entries for `file1.o` and `file2.o` is empty the action part of the `.c.o` rule will be used. Note too that it is the *object* file which depends on the header file and *not* the *source* file, i.e. if the header file is altered it is the *object* file which is out of date and needs to be rebuilt.

About The Build Tool

The underlying build tool is called `build.exe` and it may be used from a Command Prompt on its own if required, although this is not generally to be recommended.

This section provides some documentation specific to invoking the tool from the command line.

Exit Codes

Normally **Build** uses exit code 0 to indicate that there were no problems encountered while building the target(s) and exit code 2 if there were (either because of an actual error, or because an action issued to the operating system returned an error code). If the **Question mode** option is used it uses exit code 1 to indicate that one or more targets need rebuilding.

Command Line Options

The Build Tool accepts various command line options which modify its behaviour. Most of these correspond to options available from the **AllChange Build**, but there are a few additional options.

The options which are also available from **AllChange Build** are:

- Debug mode option — `-d`
- Filename of buildfile option — `-f filename`
- Ignore error codes from actions option — `-i`
- Continue after error code option — `-k`
- Macro assignments option — `-m macroassignment, macroassignment`
- No-execute mode option — `-n`
- Print macros and targets option — `-p`
- Unconditionally build — `-u`

Additional options include:

- Use Build Threads option — `-b`
This option directs the **Build Tool** to use BTs to determine whether objects are to be (re)built rather than time stamps only. When using the **AllChange Build**, this option is used automatically.
- Environment variables override option — `-e`
Normally, if a variable/ macro assignment (see [Build Macros](#)) appears in the buildfile that assignment overrides any value that the variable is currently set to in the environment — see sections [Macro Priorities](#) and [Build Environment](#).

Thus if there is currently an environment variable:

```
CFLAGS = -O
```

and there is an explicit line in the buildfile of the form:

```
CFLAGS = -p
```

the assignment in the buildfile will override the environment.

Using this option causes assignments from the environment to override those in the buildfile.

- Question mode option — `-q`
This option instructs **Build** to just return a zero or non-zero status according to whether the target is or is not up-to-date, rather than performing any actions. See the **Exit codes** section below.
This is used simply to determine whether or not a target is up to date.
- Do not use builtin rules option — `-r`
This option instructs **Build** not to read in the file of builtin rules and macros described in [Build Builtin Rules and Macros](#).
- Silent mode option — `-s`.
Normally, **Build** writes on the screen all actions as they are about to be issued to the operating system.
This option can be used if the user does not wish this output to be produced. The echoing of individual actions can be suppressed by preceding the actions with @ — see [Action Line Modifiers](#).
- Touch (update) target option — `-t`.
This tells **Build** to *update* the file modification time of all specified targets and their dependencies instead of issuing the associated actions.

This option might be used if, for example, a comment had been changed in a source file and it was not desired for the time-consuming process of a complete recompilation to be performed.

This option should be used with care: since it bypasses the actions which would normally be performed to bring the target up-to-date, it also risks infringing source and object integrity.

Notes

A common error in a buildfile is not to indent each action line in a dependency entry. This leads to unexpected behaviour and errors.

MS-DOS does not allow the = (equals) character to appear in the value of an environment variable or in an argument to a .BAT file, situations where it might be desirable for **Build**. Accordingly, under MS-DOS, ` (backtick) will be replaced by = wherever it appears in environment variables and command line macro assignments, e.g.

```
SET OPTFLAGS=/DEFINE FLAG`value
```

Under MS-DOS **Build** recognises that action lines starting with any internal MS-DOS command (`DEL`, `ECHO` etc.) or containing redirection symbols `<`, `>` or `|` must be run via `COMMAND/C`. It should be noted that `COMMAND` always returns success for actions performed this way.

About The Autobuild Tool

The underlying Autobuild tool is called `autobuild.exe` and it may be used from a MS-DOS command prompt on its own if required.

This section provides some documentation specific to invoking the tool from the command line.

Command Line Options

Most of these options are available from the **Autobuild** dialog, however, there are a few additional options also available.

The options which are also available from **AllChange Autobuild** are:

- Target filename — `-tfilename`
- Filename of buildfile — `-ffilename`
- Include directories — `-idirectories`
- Macro assignments — `-mmacroassignment,macroassignment`
- Do not recalculate dependencies — `-n`
- Replace source files — `-r`
- Delete source files — `-d`

Additional options include:

- Filename of build template file — `-bfilename`
This option is used to specify the filename of the build template file to be used by **Autobuild**. If this option is not specified the filename `build.tpl` will be assumed.
- Object filenames — `-jfilenames`
This option is used to specify which object filenames should be selected from any equivalence lists (see [Equivalent Filenames](#)) when determining the object files for this target. It may only be used if a target is specified.

Preprocess Tool

About Preprocess Tool

Prep is a (language-independent) macro preprocessor tool which is included with **AllChange**. It may be used on any kind of *text* file: program source, documentation, arbitrary configuration/ information files used by programs (e.g. the buildfile used by **Build**). It is very useful in producing different configurations for various target machines and/ or facilities offered.

Certain programming languages, such as C, may provide a macro preprocessor; others do not, or provide directives but not macros. Such preprocessors may be unsuitable for processing files other than program source code. **Prep** provides a single, consistent macro preprocessor across all programming languages, text files, etc.

Prep copies the specified *files* (standard input if none specified) to its standard output (the screen by default, but this can be directed to a file if the **Output filename** option is used), transforming the text under the control of user-specified *directives* and *macro substitutions*.

Directives are commands recognised and acted upon by **Prep**. The facilities they provide include conditional output (i.e. output or suppression of blocks of text depending on certain conditions), inclusion of other files into the output and definition of macros. *Macros* are names defined by the user to have arbitrary values; whenever a macro is encountered in the input its value is substituted into the output.

Both directives and macro substitutions are preceded by the *command character*, @ (at) by default, to so designate them. A single literal command character is produced by doubling the command character, i.e. @@. All directive and macro names are case-insensitive, so, for example, @include, @INCLUDE and @InClude are equivalent. Directives and macros are explained after the command line options.

Prep Command Line Options

Prep may be invoked with various command line options which affect how it processes files.

-c (Command character option)

This option may be used to change the command character to something other than the default @; it should be followed by the desired character. This is useful if the text contains many literal @ characters to avoid having to double them to @@.

-d (Define macro option)

This option allows macros/ variables to be defined from the command line. It should be followed by a (comma-separated list of) definition(s) using the = symbol between the macro name and its value — see [Prep Macros](#) and [Prep Examples](#).

-i (Include directories option)

This option should be followed by a list of directories. Any (relative) pathname appearing as the argument to a @include is sought first in the current directory, then in each directory in turn specified in this list.

-l (Line number mapping option)

As a result of directive lines encountered and multiple files processed the position of lines (in terms of their line numbers) in the output from **Prep** may well be quite different from their line numbers in the input. A program, such as a compiler, run on the output may report a line number in that output and it may be desired to locate the corresponding line and file in the input to **Prep**.

The **Line number mapping** option should be followed by a (list of) line number(s). **Prep** then pre-processes the input exactly as normal but, instead of outputting the resulting text, reports the line number and filename in the *input* corresponding to the specified line number(s) in the *output*.

-n (No macro substitutions option)

If this option is used no macro substitutions are performed, though any directives encountered are still obeyed. It might be used to speed execution or prevent any unintentional substitutions if it is known that the input will contain no macros.

-o (Output filename option)

Normally the output from **Prep** goes to the screen (standard output). It can be sent to a file using the **Output filename** option and specifying the desired file. This file must *not* be the same as any of the input files, otherwise it will be overwritten before it has been read.

Prep Directives

Prep Directives are commands recognised and acted upon by **Prep**. The command character preceding the directive *must* be in column 0 (unless noted otherwise); there may then be any number of spaces/ tabs (if desired for clarity) before the directive. Directive lines do not appear in the output. A list of the directives recognised by **Prep** is given in [Figure 8.7](#).

Figure 8.7: Directives recognised by Prep

@commandchar	@else	@error	@message
@comment	@elseif	@if	@substoff
@debug	@elseifdef	@ifdef	@subston
@define	@endif	@include	@undefine

@commandchar *char*

Changes the command character to *char* from now onwards, like the **Command character** option.

@comment *rest-of-line*

The rest of the line is treated as a comment and so ignored. Unlike the other directives, @comment does not have to start in column 0 — it may be placed anywhere on any line after normal text or another directive. However, if the @ does *not* lie in column 0 there may be no space separating it from the comment. Note also that if macro substitution has been disabled, @comment will not be found and removed from the output unless it starts in column 0.

@debug *rest-of-line*

Rest-of-line is output if and only if the macro name `debug` has previously been defined, either from the command line or by a @define directive; otherwise it is discarded. The first space after @debug is stripped; any further spaces are included in the output. This is exactly the same as:

```
@ifdef @debug
  rest-of-line
@endif
```

@define *macro-name=value*

Defines *macro-name* to have *value* from now on, like the **Define macro** option. The = and/ or *value* may be omitted. See [Prep Macros](#).

@else

Reverses the sense of the current @if or @ifdef.

@elseif *condition*

Equivalent to @else followed by @if *condition*, but does not require an extra @endif.

@elseifdef @*macro-name*

Equivalent to @else followed by @ifdef @*macro-name*, but does not require an extra @endif.

@endif

Ends a conditional section begun with an @if or @ifdef. Each @if or @ifdef must have a matching @endif, but note that elseif and elseifdef do not themselves have a matching @endif (though the @if or @ifdef inside which they appear does).

@error *error-message-text*

Issues an error message including the filename and line number where the @error appears, then terminates processing. The *error-message-text* is optional: if supplied, it is included in the message. This directive is intended for "Should not have reached here" or "This part not implemented" type situations.

@if *condition*

Starts a conditional output section till the matching @endif. If *condition* evaluates as *true* (see [Prep Expressions](#)) lines up to the matching @endif or one of the @else directives are processed as nor-

mal and output; if *condition* evaluates as *false* lines up to the matching `@endif` or one of the `@else` directives are ignored. `@ifs` may be nested within one another to a maximum depth of 20.

`@ifdef` *macro-name*

Equivalent to `@if @defined(@macro-name)` (see [Prep Expressions](#)).

`@include` *filename*

Temporarily switches input to come from *filename*, returning to the line after the `@include` at the end of *filename*. The contents of *filename* are processed in the normal fashion. `@included` files may `@include` further files; up to 5 files (including the top-level) may be active at any one time. *Filename* may be enclosed in double quotes if desired; if it begins with a `@` it is taken as a **Prep** macro which is expanded to give the filename.

`@message` *message-text*

Similar to `@error`, but processing continues as normal. This directive is intended for information messages like "This part not complete yet".

`@substoff`

Switches off macro substitutions from now onwards, like the **No macro substitutions** option.

`@subston`

Switches on macro substitutions from now onwards.

`@undefine` *macro-name*

Removes the definition of *macro-name*.

Prep Macros

About Prep Macros

Macros are defined with the **Define macro** option or the `@define` directive. A macro definition takes the form:

```
@define macro-name = value
```

Macro-name may be made up of letters, digits and underscores (but not starting with a digit); only the first 32 characters are significant. The `=` symbol and surrounding spaces are optional. *Value* is arbitrary text, with leading and trailing spaces stripped. If no *value* is supplied in a `@define` line it is set to the empty string, but it is set to 1 if in a **Define macro** command line option. A warning message is issued if *macro-name* is already defined, though the redefinition still goes ahead.

Whenever **Prep** subsequently finds `@macro-name`, optionally followed by another `@`, in the input it substitutes *value* into the output. (For convenience the `@s` may be included or omitted when defining *macro-name*.) The second `@` *may* always appear for clarity if desired; it *must* be present if the macro call is immediately followed by a letter, digit or underscore, another macro call or a literal `@` or `.` For example, in:

```
@define ME = I
@define NAME =jim
@define NAME_1 Fred
@define NAME_2 Smith
@define TIME 8:00
@ME met @NAME@my and @NAME_1@@NAME_2@@@ @TIME@ (pm)
```

there are no optional `@s`; it would produce:

```
I met jimmy and FredSmith@ 8:00 (pm).
```

The *value* part of a macro assignment may contain `@macro-names` to define a macro in terms of other macros, e.g.

```
@define NAME_1 Fred
@define NAME_2 Smith
@define WHOLE_NAME @NAME_1 @NAME_2
Whole name is @WHOLE_NAME
```

would produce

```
Whole name is Fred Smith
```

Prep maintains two internal macros, @FILE and @LINE, which expand to the current filename and line number wherever they are used. This permits the user to have lines of the form:

```
This was line @LINE in file @FILE of the input
```

in the output, which might perhaps be included in error messages in, say, program code.

A macro definition may be removed with @undefine.

An error is issued if an attempt is made to reference an undefined macro.

Prep Macro arguments

None of the macros examined above take any arguments; they resemble variables in programs in many ways. Macros may also take arguments and include these in their substitution, so that each particular expansion depends on the way the macro is called; they then resemble subroutine calls in programs.

Arguments are included in macro calls by following the @*macro-name* with a (, a comma separated list of arguments and a), then the optional terminating @. They are referred to in the *value* part of a definition by @1, @2, ..., @9, standing for arguments 1–9, and @0 refers to all the arguments (including commas). (There is no optional terminating @.) When the macro call is expanded occurrences of @*digit* in the *value* part of the definition are replaced by the appropriate argument. For example:

```
@define MACRO Arg 2: @2 arg 1: @1
Macro call: @MACRO(first arg, second arg)
```

would produce:

```
Macro call: Arg 2: second arg arg 1: first arg
```

Note the difference between @MACRO(1, 2)@ — a macro call with arguments 1 and 2 — and @MACRO@(1, 2) — a macro call (with no arguments) followed by (1, 2).

If a macro is called with more arguments than are referenced in its definition the extra arguments are ignored; if called with fewer arguments than in the definition the unsupplied arguments are taken to be empty.

Prep Expressions

Prep can evaluate expressions involving integers, strings and booleans. Expressions can contain macros, with or without arguments. The operators recognised are shown in [Figure 8.8](#) in descending precedence; those on the same line have the same precedence.

Figure 8.8: Operators recognised by Prep

Operators	Meaning
!	not
* /	multiply, divide
+ -	add, subtract
= # < >	equal, not equal, less, greater
&	and
	or

Precedence is similar to that in most programming languages; parentheses ((and)) can be used to alter the order of evaluation as expected, e.g. $4 - 2 * 2 = 0$ but $(4 - 2) * 2 = 4$, or for clarity. The operators = and # can also be used to test whether two strings are equal (case *sensitive*).

Expression evaluation occurs in two situations:

1. If the macro call `@eval(expression)` (optionally followed by a `@`) is encountered in the input it is replaced in the output by the result of evaluating *expression*, e.g.

```
2 times 2 = 2 * 2
2 times 2 = @eval(2 * 2)
```

would produce:

```
2 times 2 = 2 * 2
2 times 2 = 4
```

`@eval` may appear in macro definitions, e.g.

```
@define HEIGHT 10
@define WIDTH 20
@define AREA @eval(@HEIGHT * @WIDTH)
Area = @AREA
```

would produce:

```
Area = 200
```

An advanced usage of `@eval` is to precede it with an extra `@`: in this case the evaluation is not performed at the time of the `@define` but when the macro is called instead. This is very useful when macro arguments are to be used, e.g.

```
@define AREA @@eval(@1 * @2)
Area 1 = @AREA(5, 10)
Area 2 = @AREA(20, 3)
```

would produce:

```
Area 1 = 50
Area 2 = 60
```

2. The *condition* which follows the `@if` directives is automatically evaluated: it is *false* if it is zero, *true* otherwise.

The macro call `@defined(@macro-name)` (optionally followed by a `@`) is allowed inside `@ifs` and tests whether *macro-name* has previously been defined:

```
@if @defined(@MACRO)
@ifdef @MACRO
```

are equivalent, but `@defined` also allows for:

```
@if @defined(@MACRO_1) | @defined(@MACRO_2)
```

A reference in a condition to an undefined macro is treated as *false*, allowing tests of the form:

```
@if @MACRO_1 | @MACRO_2
```

for convenience, regardless of whether or not the macros have been defined.

Prep Sample input file

[Figure 8.9](#) gives a sample **Prep** input file, attempting to illustrate many features.

Figure 8.9: Sample Prep input file

```

@if @OS = UNIX
@ define HEADER (input, output) @comment Standard
    @comment FREE_MEM is imaginary calculation
@ define FREE_MEM @eval((256 * 4 + 1) * 8)
@elseif @OS = MSDOS
@ define HEADER @comment No header
@ define FREE_MEM 1024 * MemFree @comment Function call
@else
@ error Unimplemented Operating System
@endif
@define increment @1 := @1 + 1 @comment Increment arg
@define VERSION 6
@include copyright
PROGRAM sample_prep@HEADER;
VAR x : integer;
BEGIN
    writeln('Pascal program running under @OS');
@if @DEMO | @FUNNY_VERSION
    writeln('Demonstration version only');
@ if @VERSION > 3 & @VERSION < 9
    writeln('Version @VERSION@(Old)');
@ endif
@endif
@substoff @comment Want to write loads of '@'s
    writeln('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@');
@subst on @comment Switch substitutions back on
    x := 0; @increment(x);
    writeln('Memory is ', @FREE_MEM);
END.

```

[Figure 8.10](#) then shows the output that would be produced by executing the command:

```
prep -DOS=UNIX,DEMO file.pas
```

assuming the file were called `file.pas` and the file `copyright` contained the copyright comment.

Figure 8.10: Sample Prep output

```

{ Copyright Intasoft Limited }
PROGRAM sample_prep(input, output);
VAR x : integer;
BEGIN
    writeln('Pascal program running under UNIX');
    writeln('Demonstration version only');
    writeln('Version 6(Old)');
    writeln('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@');
    x := 0; x := x + 1
    writeln('Memory is ', 10250000);
END.

```

Prep Examples

```
prep -dvar1=string,var2,var3=6 file
```

Preprocesses `file`, writing its output to the screen, with variables `var1` defined to `string`, `var2` defined to `1` (the default when no value is specified from the command line) and `var3` defined to `6`. This could equivalently be invoked as:

```
prep -dvar1=string -dvar2 -dvar3=6 file
```

if preferred.

```
prep -c% -l15,64 file1 file2
```

changes the command character to `%` and reports the filename (`file1`, `file2` or any `%included file`) and line number which produce lines 15 and 64 in the output. No other output is produced.

ACCEL

ABOUT ACCEL

At this point in the manual we digress to take a brief look at ACCEL since this will be used in forthcoming sections.

AllChange includes a *Command Evaluation Language*, called ACCEL, which is used for writing conditions, actions and reports. It is used at many different points in **AllChange**, giving powerful database querying facilities and enormous flexibility in the set up of the system. Among others, it is involved in specifying conditions in browsers and for certain commands, writing report format files, checking that a given **AllChange** command may be issued, performing actions required by a command and controlling aspects of the ACEFront-end.

Most — or perhaps all — of the time it is anticipated that only the **AllChange** Administrator will need any knowledge of ACCEL. Accordingly — since the description is lengthy — ACCEL is documented in the **AllChange** Administrator Manual rather than this manual. Where ACCEL conditions may be entered inside ACE the ACCEL Condition Editor is available to aid in the task.

In summary, ACCEL is used at the following times:

- when composing report format files
- when specifying a condition to certain commands (e.g. **report**, **find**, **baseline**), as described under those commands
- when using the **eval** command
- when setting up conditions and actions for executing commands and progressing through life-cycles, as described in the **AllChange** Administrator Manual
- when writing user-defined functions
- when specifying conditions on what items to show in ACE windows
- when tailoring the ACE Front-end to site-specific requirements by "greying out" or setting the behaviour of items under certain circumstances

ACCEL provides:

- access to all fields of all databases
- hooks to **AllChange** functions and commands
- evaluable conditions
- variables
- user defined functions
- operating system calls, file handling, communication with other programs
- functions to interact with the user from ACE

Although it may be quite possible to use the **AllChange** system with no understanding of ACCEL, there are times when some understanding will be beneficial or required. The ACCEL condition editor dialogs allow ACCEL conditions to be specified without requiring a great deal of knowledge about ACCEL.

ACCEL Condition Editor

The ACCEL condition editor is a dialog inside ACE which allows simple ACCEL conditions to be constructed by selecting the required components from lists. The condition editor may be used in browsers where the condition restricts what items are shown or in command dialogs where the command has a condition as an option.

The last ten conditions used from any window or dialog will be stored and displayed in a combo box next to the condition editor button. This means that frequently used conditions may be selected from the list rather than re-specifying them in the condition editor.

An ACCEL condition is made up of a sequence of *simple conditions* joined by 'and' or 'or'. A *simple condition* is of the form:

field operator value

The *field* is a field of the database, the *operator* is a relational operator used to compare the contents of the selected field against the *value* specified. These simple conditions may be specified in the first four lines of the condition editor.

Another type of *simple condition* is a function call which is of the form:

function-name(parameters)

The *function-name* is the name of the function to call, and the *parameters* required will vary from function to function. The condition editor allows up to three parameters to be specified. On selecting a function, the required parameter names will be shown above each entry. Function calls are available on the last two lines of the condition editor.

Field	Operator	Value	and/or
User	Includes	(Current user)	
<input type="checkbox"/> not			
<input type="checkbox"/> not			

OK Cancel Clear Help

A condition is built up by selecting these components from lists; where appropriate, the lists offered are tailored to the context, e.g. when the condition editor is invoked from the Parts Browser only fields in the parts database are offered for the *field*.

When a *field* or *function-name* is selected then — where possible — appropriate values are entered into the value lists, e.g. if a user field is selected then the valid *values* would be the registered **AllChange** users.

Fields are specified according to their *common* name as defined by the **AllChange** administrator.

The operators may be site specific, but those initially supplied are:

Equal to

performs a case-insensitive comparison for equality

Not equal to

performs a case-insensitive comparison for inequality

Like

performs a (case-insensitive) pattern match (the *value* should contain * to denote zero or more of any characters).

Not like

performs a not a pattern match

Contains

performs a search for the *value* in the *field* which is a list.

Contains string

performs a (case-insensitive) search for the *value* in the *field*

Contains words

performs a (case-insensitive) search for each of the words specified in the *value*. All the words must be found for the condition to succeed.

Does Not Contain

performs a search for the *value* not being in the *list*.

Does not contain string

performs a search for the *value* not being in the *field*.

Matches

performs a pattern match for *value* in a *part field* (the *value* should contain * to denote zero or more of any characters).

One of

performs a search for the *field* in the *value* which is a list.

Not one of

performs a search for the *field* not being in the *list*.

Includes

looks for the specified user as included in (a member of) the **field**, where the field may be a user name or a group. i.e. if the field has a group value it will look to see if the specified user is a member of the group. If the field has a user value it will look for equality of the specified user with the value.

Is true

tests whether the field has the value *true*

Is false

tests whether the field has the value *false*

Before

tests whether the value of a date field is *before* the date specified

After

tests whether the value of a date field is *after* the date specified

On

tests whether the value of a date field is the date specified as the value

On or after

tests whether the value of a date field is on or after the date specified as the value

On or before

tests whether the value of a date field is on or before the date specified as the value

Note that the operators are not actual ACCEL operators, but map onto various ACCEL operators and functions. The actual ACCEL code resulting from the condition selected in the condition editor will be shown in the window that the condition editor was invoked from on selecting **OK**.

The *value* may be selected from the list shown in the combo box where a single value is required. Where a list of values are required by the selected operator, then the *value* combo box will change into an edit control and a button (. . .) which allows several values to be selected from a list.

Notes:

- If the field is a field which holds a user name (such as the Assignee, User, Reviewer):
 1. the Value list will show the users Full Name the value in the index will be the users logon id and so the sort order will be according to this and *not* the Full Name.

2. if the Value is specified as a wild card (e.g. f*) then this will be a pattern match on the index value (i.e. the users logon id) and not the Full Name
- If the field is an arbitrary field and is only defined for specific classes then the condition generated will test for the field being defined before testing the value. This is to ensure only valid fields are tested.

Creating and Managing Change Requests

About Creating and Managing Change Requests

The change management facilities provided by **AllChange** are fully integrated with the configuration management facilities.

Change management centres around *change requests* (CRs) which are used to log upgrade requests and bug reports; they form an important part of the change control process. They may be used to enforce change control procedures and to automate procedures using life-cycles.

Note that the term CR may have been mapped to site specific nomenclature (e.g. ACMD or RFC) in which case all references to CR will show using the mapped nomenclature instead.

CRs in **AllChange** include facilities for:

- Any CR numbering scheme
- Different classes of CR
- Hierarchical CRs
- Life-cycles according to class
- File attachments
- Relationship to parts and baselines
- Assignment of CRs to users/ groups
- Information awareness maintained by use of email
- Approvals via Voting facility
- Site definable fields
- Full search and query support
- Metrics generation and graphical reporting
- Audit trail
- Automatic mailing

CRs are identified to **AllChange** by their "number". The CR numbering scheme is site-specifiable, and may be made to vary according to the CR's class. A CR's "number" is its primary identification field; it will probably look like a number, or at least contain a number, though it does not have to. **AllChange** refers to this CR identifier as a *CR number* regardless of its actual format.

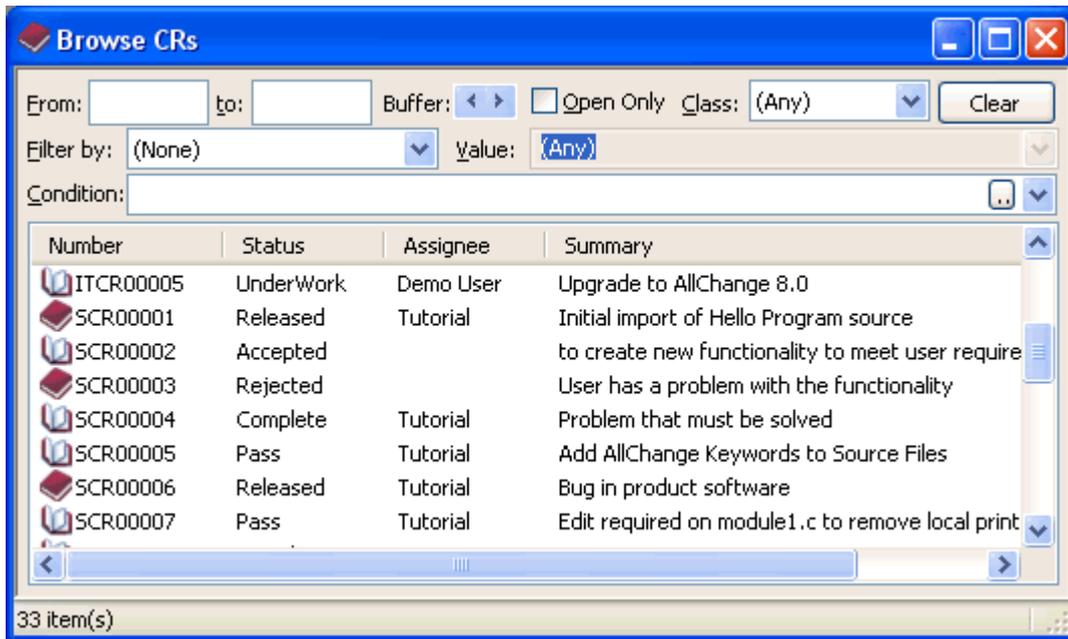
Different (site-defined) classes of CR may be used to classify different kinds of CRs such as upgrade requests, bug reports, product defects, problem reports, etc. Each class of CR may have a different life-cycle and different fields if so required. CRs may progress through their life-cycle's statuses as they are actioned. This mechanism allows approval procedures to be implemented.

CRs may be linked to the items that they affect and the items which implemented a solution for it. This allows for impact analysis and change control to be logged and enforced. It also allows for easy access to information as to, for example, *which version implemented a solution to this CR?*

The CRs are stored in the CR database which may be used for querying/ reporting purposes.

Browsing CRs

The CR browser (available from **CR | Browse**) shows an overview of the CRs which exist.



The precise information shown for each CR is user definable (see [Customising the User Interface](#)). The icon will show whether the CR is open or closed according to the life-cycle definition and the current status. In addition the icon will show a paperclip to indicate that the CR has an attachment if the [Show attachments flag in icons options](#) is selected.

Icon	Description
	CR is open
	CR is closed
	CR has an attachment

Since there may be a large number of CRs it could be very slow to read them all in order to fill the browser list. Therefore the list will only contain a buffer at a time; [Customising the User Interface](#) describes how the buffer size may be changed.

The **From** and **To** may be used to limit the CRs displayed within a range of CR numbers. Partial CR numbers may be expressed and a pattern match will be performed for CR numbers which match the string specified in its leftmost part (i.e. effectively any cr number matching *from** or *to**).

The **Buffer** arrows will read the next/ previous buffer's worth of CRs.

If **Open Only** is selected then only CRs who are in a status which is classified as open will be shown.

The **Class** drop-down list may be used to limit the items listed in the browser to those matching a specified class. Only classes allowed to be viewed by the current user will appear in the list. If only one class name appears in the list, then it will be auto-selected.

The **Filter By** may be used to restrict what is displayed according to the **Value**, see [Selecting CRs](#) Shown in the Browser.

The **Condition** may be used to further limit the CRs to be shown to any arbitrary criteria.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/ Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.

- The sort order by CR number may be listed in alphabetical or reverse alphabetical order. Right click on the title bar of the browser and select **Sort A-Z** or **Sort Z-A**
- The current buffer may be sorted by any column if the **Allow CR browser header sort** option is selected, see [Customising the User Interface](#).
- [Folding View](#) showing the relationship between CRs

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

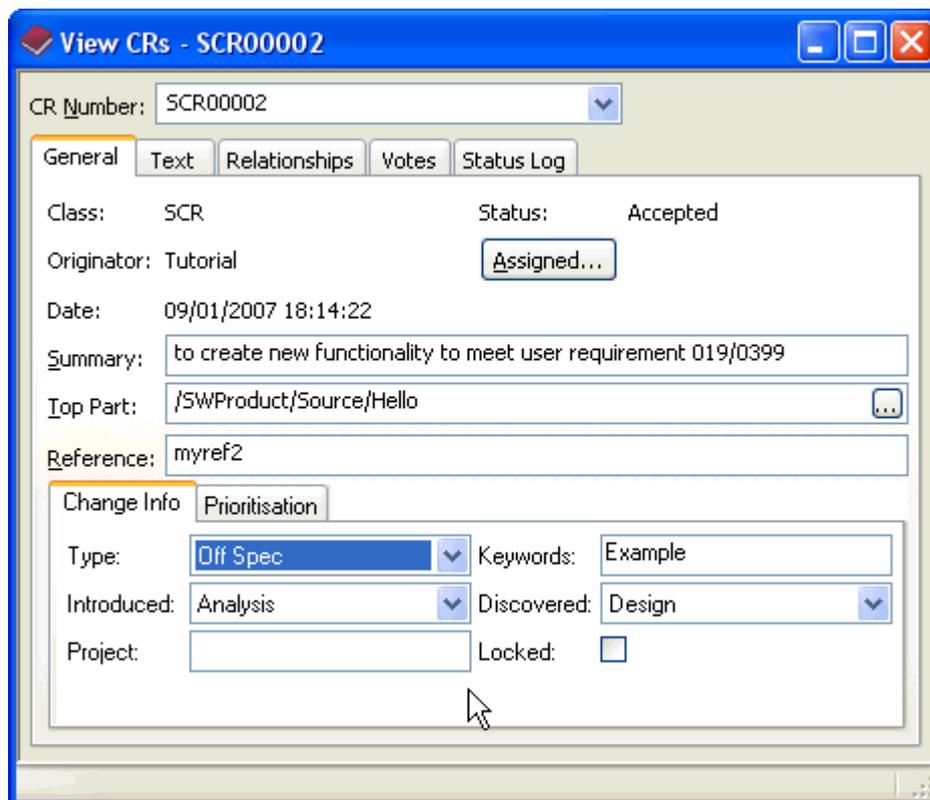
Double clicking on a CR in the browser will show the details of the selected CR in the CR viewer window.

View CR Information

View CR Information

CRs have a number of fields and relationships to other databases associated with them. The fields and relationships may be set by various commands or used for querying/reporting.

To view the details of a particular CR use the cr viewer; this may be accessed by **CR | View**, or double clicking on a CR in the CR browser.



The information about the selected CR is shown in a series of tabs. Select the tab required and the information for that tab will be shown. The tabs for the CR viewer are:

General

Shows general information about the CR

Text

Shows the descriptive text associated with the CR

Relationships

Shows information about relationships to other items such as Parts, Baselines, other CRs, and attachments .

Votes

Shows the votes/approvals associated with the CR.

Status Log

Shows the status and assignment audit trail for the CR

Each CR is identified with a CR number, the CR currently viewed is shown in the window title and in the **CR Number** edit control.

CR Number*About CR Number*

The CR number is determined at the time the CR is created and depends on the CR class. It may be formed partially or wholly from a string supplied by the user (the **Id**) and/ or a number generated by the system when it is first created. There are two features of CR numbering which may be controlled:

- CRs may be "numbered" in almost any site-definable fashion (including using a purely user-supplied string generated externally to **AllChange**);
- there are (10 independent) auto-incrementing counters for use in automatic generation of new CR numbers.

The default numbering scheme for all CRs which do not have an explicit numbering format associated with their class is a plain 5 digit number which starts at 00001 and is automatically incremented by the system whenever a new CR is created. All such CRs, regardless of class, share the same incrementing number.

CRs must *always* be referred to by their full number, e.g. 00017, never just 17. For simplicity most examples in this manual use the default numbering scheme; a site, however, might have a CR number like MyCR1234-BUG/1.

CR numbering formats are totally site-definable so the **AllChange** Administrator should inform users if a convention other than the default is adopted.

Obsolete CRs

If the CR is *obsolete* then this will be shown at the top right of the CR view window as **OBSOLETE**. This flag may be used if a CR is no longer to be considered but needs to be kept for historical purposes.

This obsolete flag may be changed using **CR | Alter | Make Obsolete**. The obsolete flag may be used to restrict the CR browser, reports etc. to include only those CR which are not obsolete. By default obsolete items are not shown in browsers, this may be changed using **View | Show Obsolete Items**.

The General Tab

The general tab shows the base fields for the CR record together with the site specific arbitrary fields.

The screenshot shows a window titled "View CRs - SCR00002". At the top, there is a dropdown menu for "CR Number:" set to "SCR00002". Below this are several tabs: "General", "Text", "Relationships", "Votes", and "Status Log". The "General" tab is active, showing fields for "Class:" (SCR), "Status:" (Accepted), "Originator:" (Tutorial), "Date:" (09/01/2007 18:14:22), "Summary:" (to create new functionality to meet user requirement 019/0399), "Top Part:" (/SWProduct/Source/Hello), and "Reference:" (myref2). There is an "Assigned..." button next to the Originator field. Below the "General" tab is a "Change Info" section with a "Prioritisation" sub-tab. This section contains fields for "Type:" (Off Spec), "Keywords:" (Example), "Introduced:" (Analysis), "Discovered:" (Design), "Project:" (empty), and "Locked:" (checkbox).

The fields shown are:

Class:

The class of a CR. This may be any of the defined classes as described in the **AllChange** Administrator Manual. The class of a CR defines the life-cycle of the CR, its numbering scheme, the site specific fields and the template for the text.

Note that the term class may have been mapped to site specific nomenclature in which case all references to class will show using the mapped nomenclature instead.

Status:

The current status of a CR in its life-cycle. Each status is classified as either *open* or *closed* in the life-cycle definition — this is shown in the CR browser. This field is set when the status of the CR is changed and is initialised to the initial status of the CR life-cycle.

Note that the term status may have been mapped to site specific nomenclature in which case all references to status will show using the mapped nomenclature instead.

Originator:

Identifies the originator of the CR. This field is automatically set to the user who created the CR.

Assignee:

The user/ group currently assigned to the CR. This field is set when the CR is assigned.

Summary:

May be used to summarise what the CR is about. May contain any text.

Toppart:

The full path to the top level part which a CR affects. The CR should not affect any part higher in the design tree than this part.

Reference:

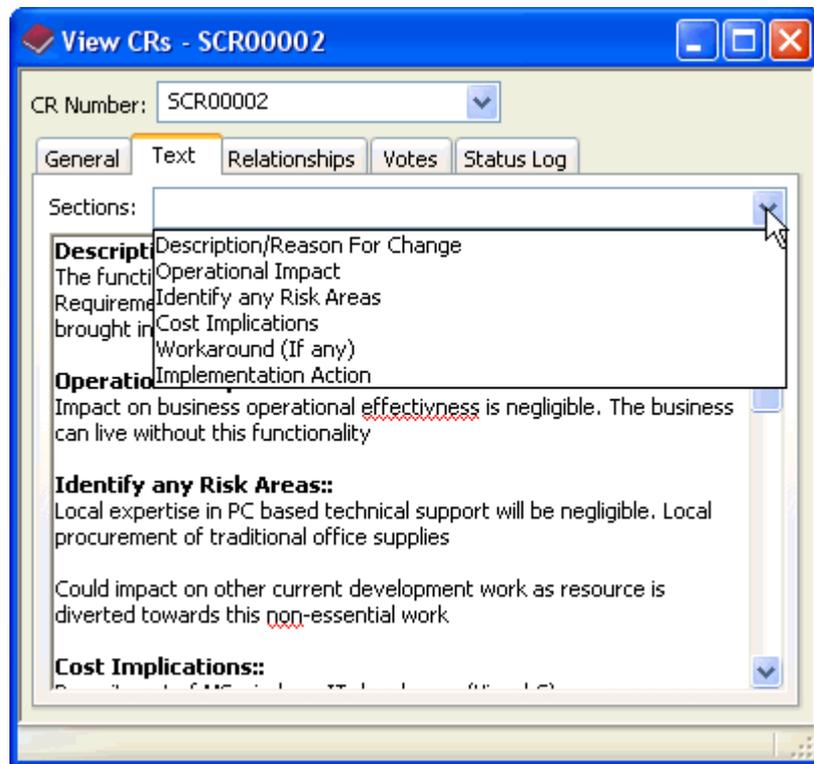
Any text which is desired to be used as a reference for a CR. This may be useful as an internal reference relating the CR to a paper document for example. This field is *not* used by the system to identify the CR.

Arb1... Arb100:

Each CR has forty arbitrary fields which may vary in contents for each class of CR. They are displayed on one or more tabs on the viewer as defined for your site. Each site should have a policy as to whether these fields are to be used at all and, if so, what information they should contain. If these fields are in use the **AllChange** Administrator should have assigned them a name indicative of their function and conveyed this information to users.

The Text Tab

The text tab shows the text associated with the CR, Part or Baseline.



No constraints are imposed on its format or contents by **AllChange**. However any line matching a site defined pattern as a section start will be listed in the **Sections**. See the **AllChange** Administrator Manual for details on defining the section pattern. The default pattern is lines ending in : : are treated as sections. Section headings will be displayed in bold. Selecting a section will scroll the text to the start of that section.

Hyperlinks in the text will be shown as clickable links. Hyperlinks are activated for common protocols such as "http", "ftp", "mailto", etc, and also for the "allchange" protocol, and any other registered **AllChange** protocols. If the text is read-only, then the link is followed by single-clicking the link. If editable, then the Ctrl key needs to be held down while clicking to follow the link. Right-clicking on links will show a menu where the link may be opened or copied to the clipboard.

When items are created they have an initial text "template": it is up to the individual site to supply suitable empty template forms for their items and to ensure that users complete these in an acceptable fashion. Different classes of item may have different initial text templates.

Classes of items may be defined as having *protected text*. This means that the text for the item may be entered as normal at creation time, but thereafter may only be appended to within each section together with an automatic stamp of the date/time of the addition and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the CR, Part or Baseline menu. If the text is *protected* the background colour of the text tab will be by default *greyed out*, so show that text may not be

[Customising the User Interface](#) for details.

Text spelling is checked in the text tab. Any misspelled words are underlined with a red wavy line. See [Modifying Text](#) for more information on spelling options etc.

Searching in Text

Text may be searched for a specific word or string. Also text may be replaced using a Find/Replace facility.

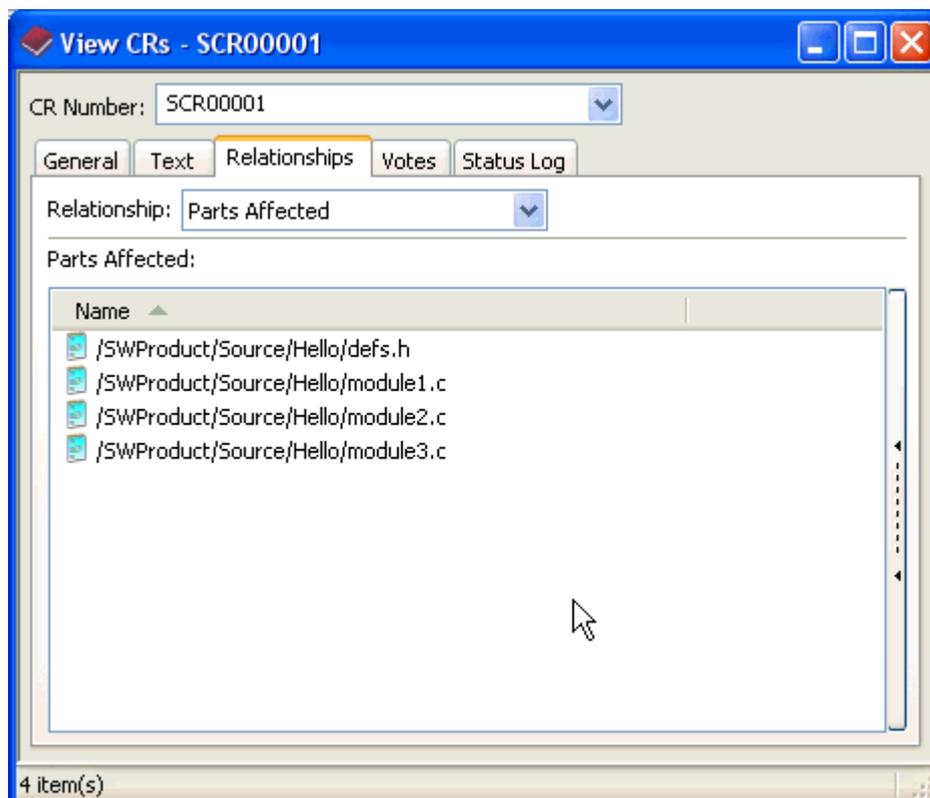
To invoke Find, right-click the text field and select Find, or use the shortcut keys Ctrl+F or Ctrl+D. This will open a dialog box where the search string can be entered. Options may also be specified here to match the case, or match a whole word. Once a word has been searched for, pressing F3 will repeat the search starting from the current cursor position. Pressing F3 without having specified a search string will open the dialog as shown on pressing Ctrl+F/Ctrl+D.

To use Find/Replace, press Ctrl+H, or right-click the text and select Replace. A dialog will open allowing the search string to be specified, along with the replacement string. Options may also be specified here, as per Find. If the text is protected against editing, then the Replace functionality is disabled.

The Relationships Tab

The relationships tab shows the relationships of the selected CR to other items.

Note that the terms CR, part and baseline may have been mapped to site specific nomenclature (e.g. RFC, CI, Release) in which case all references to that term will show using the mapped nomenclature instead.



The **Relationship** to be viewed can be selected from the drop down list as follows:

Parts Affected

Shows a list of the parts which the CR *affects*. These are the parts which are designated as affected by the CR and will be (or have been) changed as a result of the CR.

Parts would normally be added to this list automatically as a result of a checkout for edit against the CR.

Versions Solved

Shows a list of the versions which *solved* the CR. These are the specific versions of the **Parts Affected** which implemented the solution to the CR. The versions solved may include versions which are *obsolete* (or whose component is *obsolete*) if, for example, the change required to implement this CR is to remove that part.

If the **Show Latest Solved Only** checkbox selected it will display only the latest version solved for a part, where there is more than one. The latest version would be the latest version created, by date and time, which appears as a version solved on the CR. **Show Latest Solved Only** will be disabled if the Only 1 Versions Solved configuration option has been selected as there will only be one version for any given part.

Baselines may be updated/populated based on the versions solved by selected CRs allowing releases to be made based on the changes which are to be included in the release. This allows versions created as a result of a CR to be added to a baseline and versions/components made obsolete by a CR to be removed from a baseline, see [Updating a Baseline from CRs](#) for details.

Versions would normally be added to this list automatically as a result of a check in of the version which was previously checked out for edit against the CR.

CRs Affected

Shows the CRs that the CR selected affects — i.e. the CRs that the CR viewed points to. This may be used, for example, to create a hierarchy of CRs — *Incidents* could be related to *Problems* raised as a result of the *Incident* which in turn could cause a number of *RFCs* to be raised as used in the ITIL out-of-the-box project configuration.

These relationships may be automatically created for example by a life-cycle or automatic cross referencing.

CRs Referring

Shows the CRs which *refer* to the CR selected — i.e. the CRs which point to the CR selected.

CRs Affected and Referring

Shows both the CRs Affected by and the CRs Referring to the selected CR.

Baselines Affected

Shows the baselines that are designated as affected by the CR. This can be used to reflect releases in which the problem occurred.

Baselines Solved

Shows the baselines that *solve* the CR. This may be used to show the baselines in which the changes resulting from the CR are implemented.

If using automatic baseline population from CRs via the baseline life cycle or using the [CRs For Baseline](#) functionality then this list is created automatically.

Attachments

Shows a list of the files which are *attached* to the CR. This may be used, for example, to attach screen dumps or test results to a CR.

Double clicking on a file name in the list will allow you to view the contents of the file using the application associated with that file type in your operating system.

Attachments may be held as *copies* or as *links*. If a copy attachment is used then the attached file will be *copied* to a secure area on attaching it. If a link attachment is used then the path to the file will

simply be stored on performing the attachment. Linked attachments will change as the original (and only) file attached changes. A copy attachment will remain as it was at the time of attachment until it is re-attached. On the other hand there is no protection against a file which is attached as a link, from being removed.

Whether copy or link attachments are enabled is site specific and will have been defined by the **All-Change** administrator. Attachments may be defined as always links, always copies or *ask*. If *ask* is used (the default), then on attaching a file you will be prompted as to whether you wish the file(s) to be attached as links.

Any attachments shown with a full path are link attachments, otherwise they are copies.

The **Attach** menu option or button will present you with a file browser allowing you to select the file(s) that you wish to attach to the CR. For copy attachments the selected file(s) will be copied to a secure area under **AllChange**'s control. Each file attached to a CR must have a unique filename (case insensitive); however, different CRs may have quite different attachments which happen to bear the same filename.

If, at a later date, you wish to replace an attached file with a newer/different version/instance of the file then:

- for attachments which are copies:
 1. use the **Extract To** option on the right click menu to create a (writable) copy of the attached file in another directory
 2. modify the extracted file
 3. attach the file again and it will be re-attached as the new version of the file.
- for attachments which are links:
 1. use the **View** option on the right click menu (or double click)
 2. edit the file and save

Use the **Detach** button to remove any file attachments that are no longer required. Note that this will remove the copy file created when the attachment was created for a copy attachment and will have no effect on the file for a link attachment.

CRs which have attachments will be shown in browsers with a paperclip on the icon if the [Show attachments flag on icons](#) option is selected.

As appropriate for the relationship shown there may be actions performed to add or remove items from the list. These actions will be available from right click context menu, or from buttons available when the *show/hide action buttons* button to the side of the list is selected.

Double clicking on an item in a list will open the viewer for the item selected.

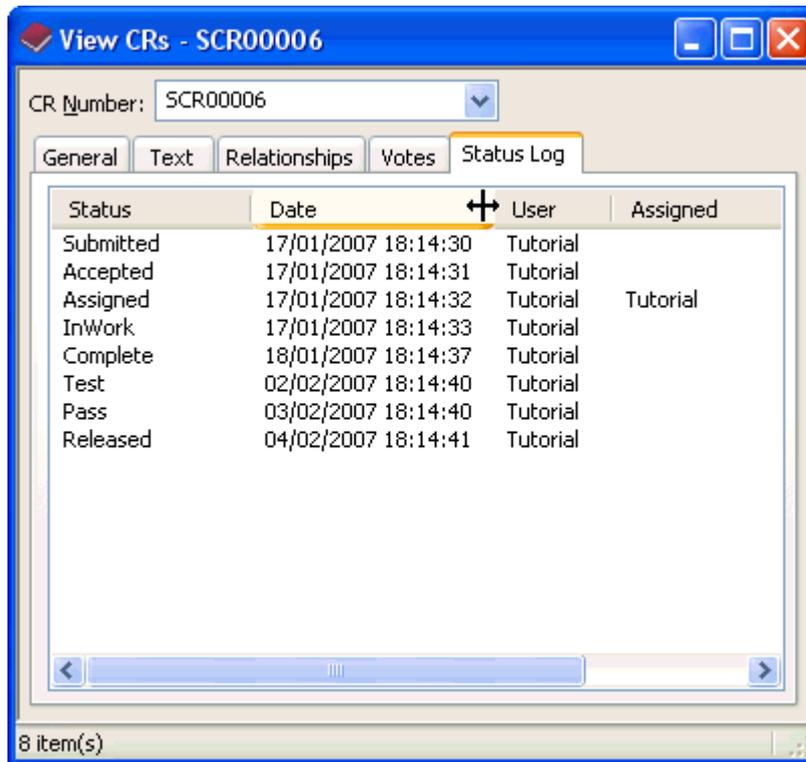
Additional details about the relationship may be view using [View CR Relationship](#) menu item on the right click context menu. This is also available from the **CR | View CR Relationship**

All relationships for CRs may also be browsed using **CR | [Browse CR Relationships](#)**

The Status Log Tab

Details of all status changes and CR assignments are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on, or viewed in the **Status Log** tab. This provides an historical record of the progress undergone by a CR, thus providing an audit trail.

Note that the term status log may have been mapped to site specific nomenclature (e.g. audit trail) in which case all references to status log will show using the mapped nomenclature instead.



The status log will show the statuses that the CR has been through. By default it will show the status, the date and time the status was entered and the user who entered the status; however, this may be configured to site/ project/user requirements.

The status log has one field which may be used to hold arbitrary, site-specific information about the status log.

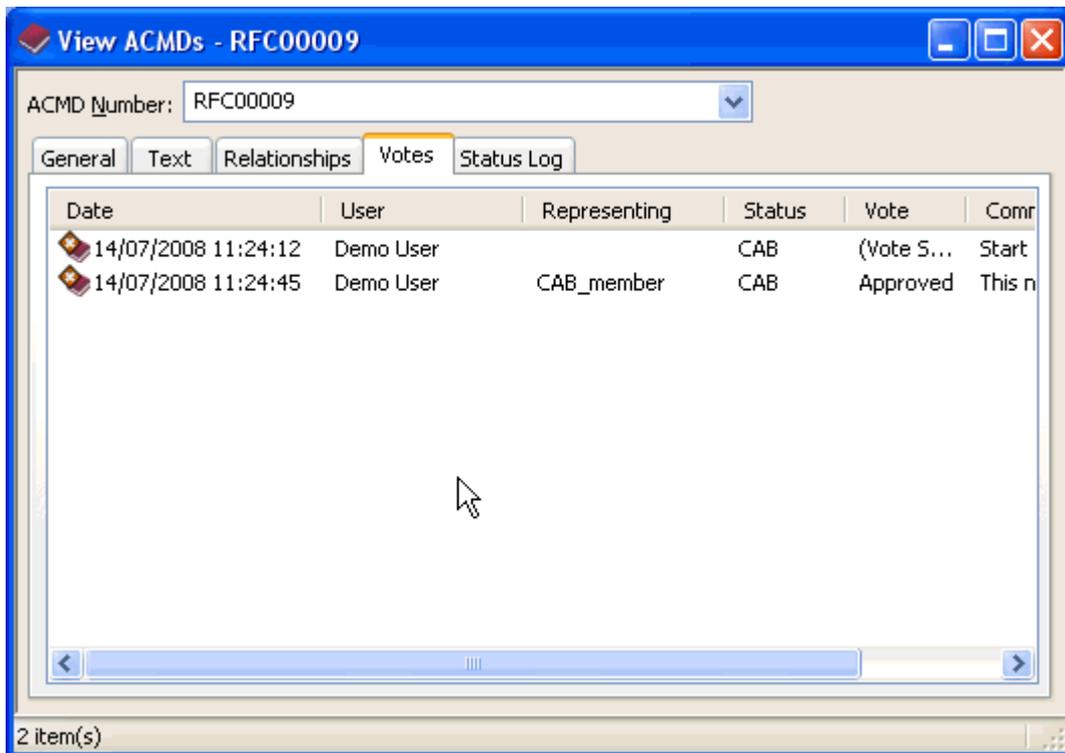
In addition to logging status changes, assignments of CRs is also logged in the status log. The status field for an assignment will be **Assigned**, the date and time of the assignment is the **Date/Time** field, the **User** field is the user doing to assignment and the user/group assigned to is logged in the arbitrary field.

The use of the arbitrary field to hold assignment information does not prevent it from being used for site specific information for status changes.

The Votes Tab

The Votes tab shows a list of all the votes which have been cast for the CR.

Note that the term vote may have been mapped to site specific nomenclature (e.g. approval) in which case all references to votes will show using the mapped nomenclature instead.



The details of a specific vote may be seen in the Vote Viewer by double clicking on the vote or from the right click context menu.

Votes may be cast when the part is in a vote status by use of the cycle viewer or **Vote | Cast Vote**

Browsing and Viewing CR Relationships

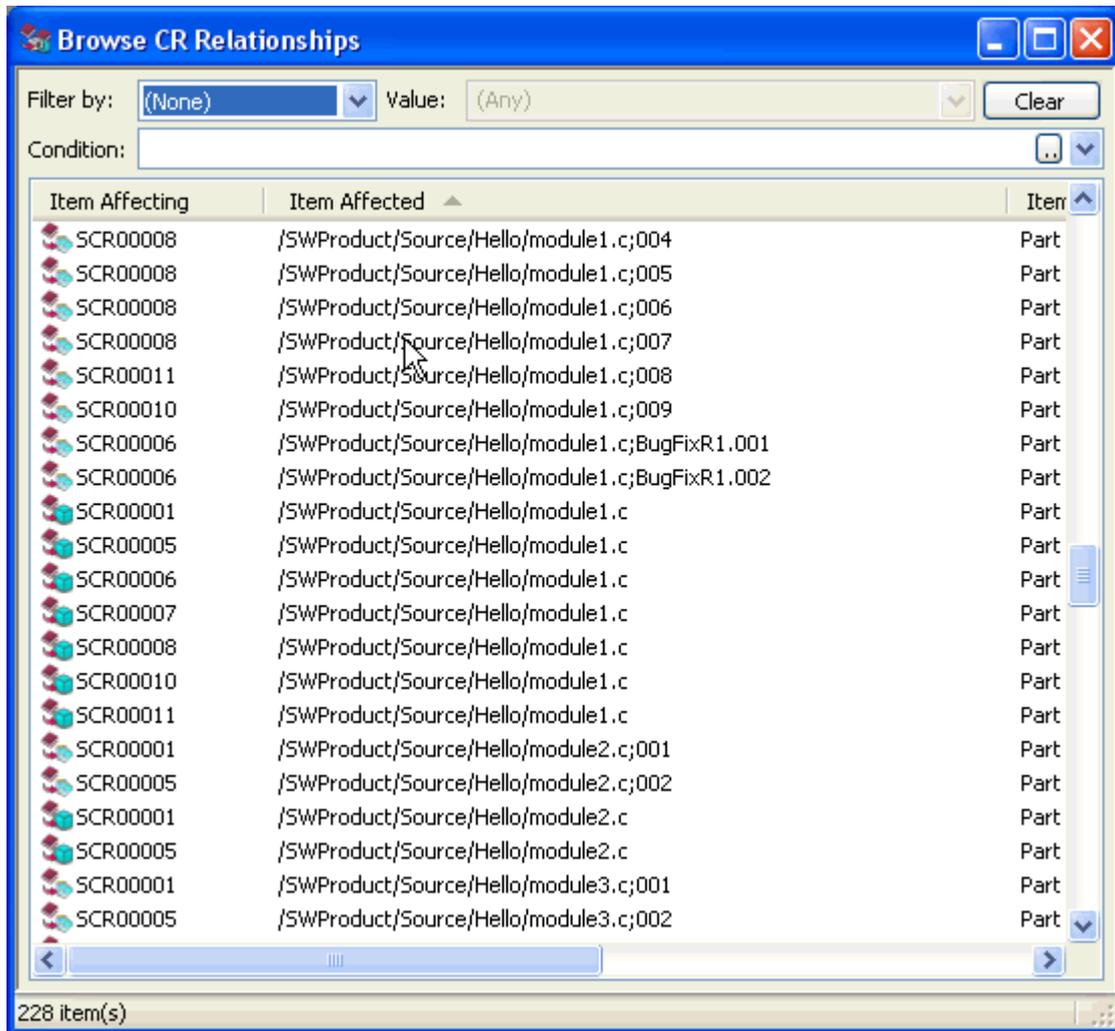
CRs can be related to parts, baselines, other CRs and files. In addition there are 2 types of relationships for parts and baseline, affected and solved. These relationships can be examined using:

- [CR Relationships Browser](#)
- [CR Relationship Viewer](#)
- [Relationships Tab](#) of the CR Viewer to view the CRs relationships to parts, baselines, other CRs and attachments
- [Relationships Tab](#) of the Part Viewer to view the CRs related to the part as a part affecting or a versions solved
- [Relationships Tab](#) of the Baseline Viewer to view the CRs related to the baseline as a baseline affecting or baselines solved

These relationships may be created and deleted from the **Relationships** tab on the CR Viewer and may also be added automatically as the result of certain operations. They may also be deleted from the CR Relationships Browser.

CR Relationships Browser

The CR Relationships Browser allows you to browse the relationships between CRs and other items (parts, baselines, files and other CRs), it is accessible from the CR Menu



The precise information shown for each item affected is user definable (see [Customising the User Interface](#)).

The **Filter By** may be used to restrict what is displayed according to the **Value**.

The **Condition** may be used to further limit the items affected shown to any arbitrary criteria.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.
- The information may be sorted by any column
- [Folding View](#) showing the relationship relationships for each CR

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

Double clicking on an item in the browser will show the details in the CR Item Affected viewer window.

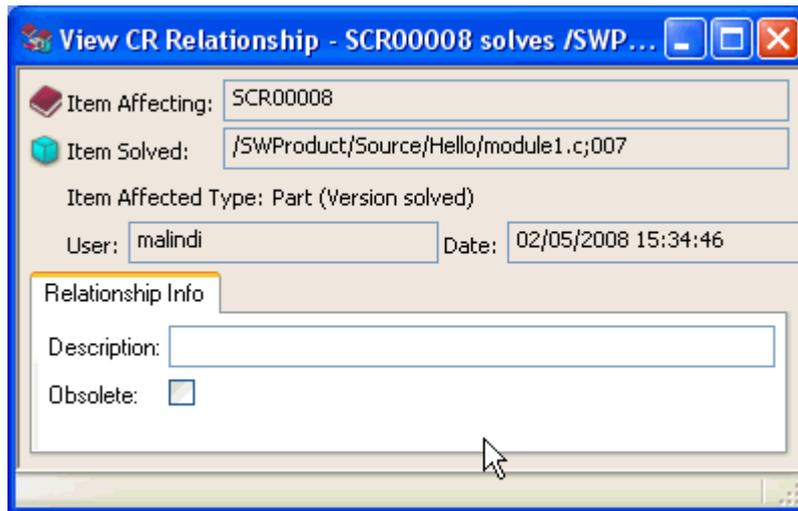
CR Relationship Viewer

Information about the CR relationships to other items in the system is stored in the cr relationships (also known as items affected) database. CR relationship entries have a number of fields associated with them.

To view the details of a particular cr relationship entry use the cr relationship viewer; this may be accessed by **CR | View CR Relationship**, or double clicking (or using the right click context menu) on a cr rel-

relationship entry in the cr relationship browser or right click on an item in the relationships tab in a CR, Part or Baseline viewer.

The CR Relationship Viewer shows the detailed information about a CR relationship to another item



The fields are:

Item Affecting:

The CR which holds the relationship.

Item Affected/Solved:

The item which is affected by the CR or the item which was used to *solve/implement* the CR which is the **Item Affecting**. This may be a part, version, baseline, cr or file and the icon next to it will denote which type of item it is.

Item Affected Type:

This will show information as to the type of the item related to the **Item Affecting** and, for parts and baselines, whether is it a part affected or version solved, or baseline affected or baseline solved

User:

The user who created this relationship. *Note this may be blank for data created prior to version 7.2 of AllChange*

Date:

The date when this relationship was created. *Note this may be blank for data created prior to version 7.2 of AllChange*

Arb1... Arb40:

Arbitrary, site-specific information about the item affected relationship. The field names may be configured as required for your site.

Creating New Change Requests

About Creating New Change Requests

There are a number of different means by which CRs may be submitted and stored in the **AllChange** database.

- [From within ACE .](#)
- [Via Email](#)
- [From the AllChange Web Browser Interface](#)
- [From a Word document](#)
- By Importing from a CSV File

Creating a CR in ACE

About Creating a CR in ACE

In order to create a new CR in ACE use the **New CR** dialog available from **CR | New** or from the toolbar when the CR browser or viewer is the current window.

Fields which are compulsory will be shown in red and a value must be supplied for these fields.

The screenshot shows the 'New CR' dialog box with the following fields and options:

- General Tab:**
 - Class: SCR (dropdown menu)
 - ID: (text input field)
 - Summary: (text input field)
 - Top Part: (text input field with a browse button '...')
 - Reference: (text input field)
- Change Info Section:**
 - Prioritisation sub-tab
 - Type: (dropdown menu)
 - Introduced: (dropdown menu)
 - Project: (text input field)
 - Keywords: (text input field)
 - Discovered: (dropdown menu)
 - Locked: (checkbox)

Buttons at the bottom: OK, Cancel, Help.

An **Id** should only be entered if the class of the CR to be created defines a CR numbering scheme which requires an **Id** to be input. The **Id** will be used to form part or all of the CR number for the new CR.

A **Class** should be selected. This will determine the new CR's:

- Number (i.e. what number, prefix, suffix or id to use to create the CR number)
- Life-cycle (this will determine the initial status of the CR)
- Arbitrary fields
- Text template

When a class is selected the arbitrary fields on the **General** tab and the **Text** tab will change as appropriate to that class of CR.

The **Summary** should be entered as a brief description of the CR.

The **Toppart** should be entered if required, and may be selected from a part browser dialog from the ... button.

The **Reference** and **Arbitrary** fields should be entered as required for the site.

The **Text** should be entered as required according to the template.

Any file attachments required may be specified on the **Items Affected** tab, together with any related parts, baselines and CRs. Select the **Create as link** option if the attachment is to be created as a link rather than a copy, see [Attachments](#).

Once a new CR is created, if the CR viewer is open, then it will be shown in the viewer. The new CR number will be reported in the output window.

The new CR will automatically have the **Originator** logged as the user who created the CR.

The CR status will be set to the initial status in the CR life-cycle and (providing CR status logging has not been disabled) the status log will be updated to show the initial status, date and user who entered the status.

Creating a New CR from an Existing CR

An alternative method for creating a CR is to create one which is a copy of an existing CR.

The **Copy CR** function is available from the **CR** menu and 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.

Creating a CR by Email

In order to create a CR by email your **AllChange** administrator must have configured the mail system and **AllChange** to accept CRs in this manner, see the **AllChange** Administrator Manual for details.

You will then need to send a mail message using your mail system to a designated user, e.g. **AllChange**. Any subject specified in the message will be used to mail you back some feedback after the CR has been created.

The mail message text will need to conform to the format shown below:

```
RemoteCommand=NewCR
Summary=<summary>
AssignTo=<assignee>
class=<class>
<arbfield name>=<arbitrary field contents>
CRText=<Cr text>
<more CR text>
```

The only compulsory entries are **RemoteCommand** and **class**, other than any compulsory rules implemented within **AllChange**. Any additional fields may be specified in the form of *field=value*.

When the CR is created in **AllChange** you may receive an acknowledgement that looks something like:

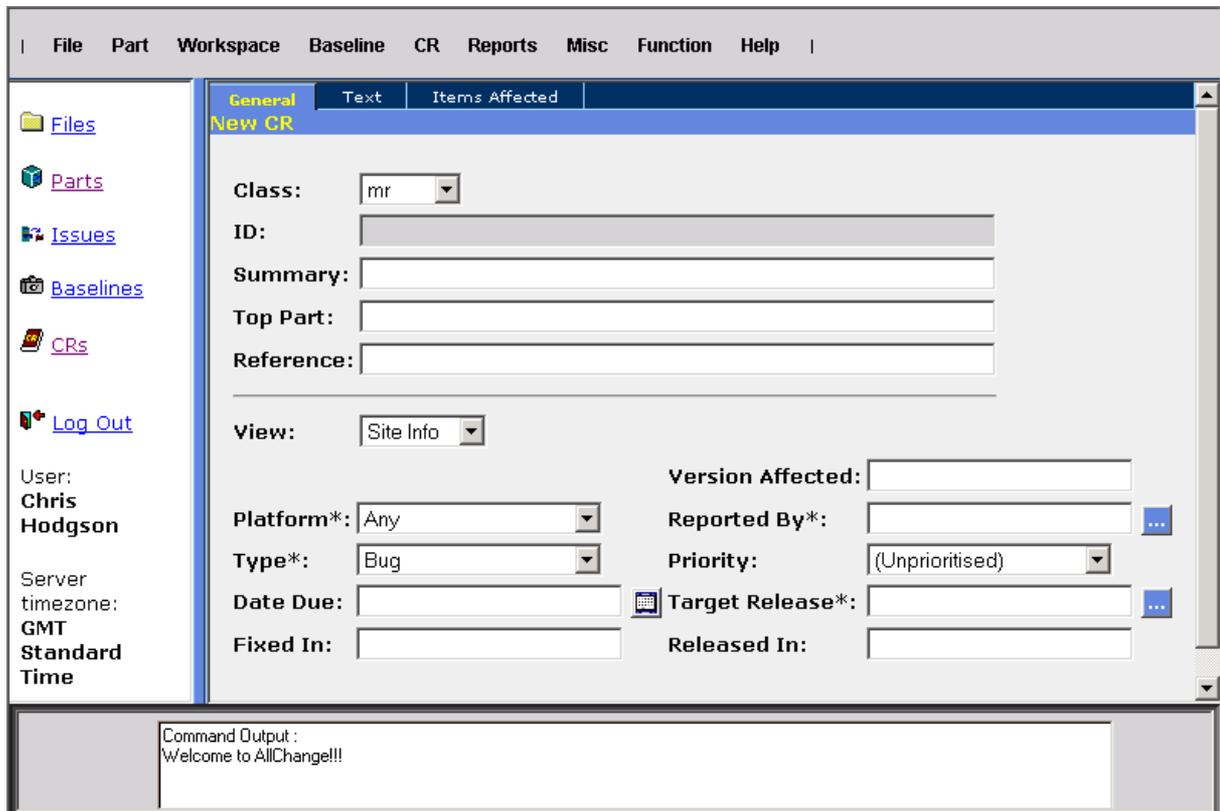
```
Subject: RE: <subject originally specified>
CR number <XXXX> has been created for you.
```

In order for the CR to be created in **AllChange** an **AllChange** function must be invoked — this may have been configured to be done automatically from time to time or it may be that a user must periodically invoke the **ImportFromEmail** function from the **CR** or possibly **Function** menu. This function will read any mail for the special **AllChange** user and process it to create the required CRs. See the **AllChange** Administrator Manual for details on how to set this up.

Creating a CR using the Web Browser

The Web browser interface to **AllChange** may be used to create new CRs over the inter/intranet.

Select the **New** from the **CR Menu**. Enter details as needed as for [creating a CR in ACE](#) and select **OK**



Full details of the **AllChange** Web Browser interface may be found in [AllChange](#) Web Browser Interface.

Creating CRs from a Word Document

The Word interface to **AllChange** allow CRs to be created from the content of a Word document.

This facility may be used to import, for example, requirements into **AllChange** as CRs from a requirements document.

For this facility to be available the Word Interface must have been selected during a Workstation install.

Details are given below which are correct at the time of printing, however, please see the `read.me` file for information on any changes which may have occurred,

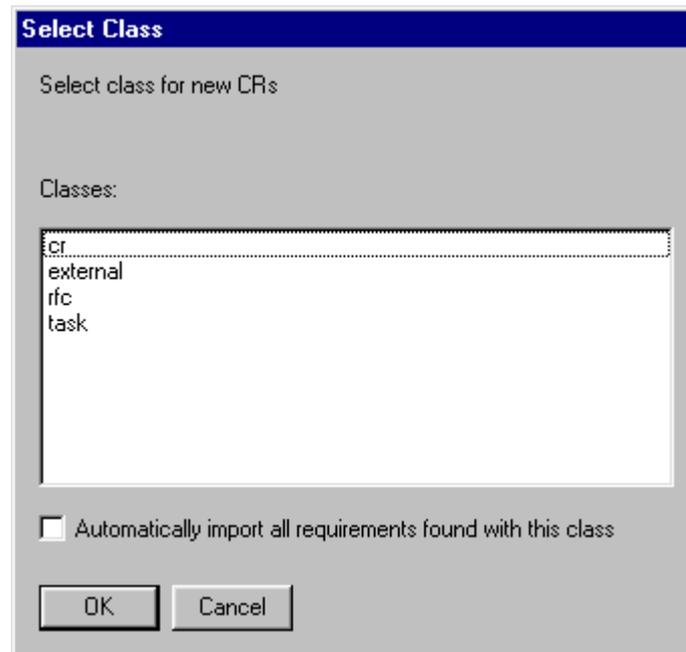
If enabled there will be an **AllChange** menu on the Word menu bar, the **Add Requirements** option of this menu will allow CRs to be created in **AllChange** from information found in the current Word document.

This will search for potential CRs to be imported in either the selected paragraphs if there are any, or, if no selection, then from the paragraph containing the current insertion point (cursor position) to the end of the document.

A potential CR is identified by either of the following:

- Numbered sections, each numbered section is a potential CR
- Titles specified in the **MSWordReqTitles** configuration option, see the *AllChange Administrator Manual*. The title must be some text in bold and in its own paragraph.

When a potential CR has been identified, you will be prompted for whether to add it as a CR, if you select **Yes** the you will be prompted for what class to use for the new CR.



Select **Automatically import all requirements with this class** if you do not wish to be prompted for the class of each CR to be created.

It will then search for any *fields* that it can identify to populate the new CR with.

A *field* is recognised as text in bold, followed by a space/ tab or non bold text.

The *value* for the field is taken to be any non bold text up to end of line or the next recognised field.

For each field identified a search is made to see if this matches any of the fields defined for the class of CR to be created, if found then this is the **AllChange** field that the value will be associated with in the **All-Change** CR. If the field does not match any of those in **AllChange** then you will be prompted to identify which **AllChange** field is to be used (if any).

If is desired to use a *field* of the word document as the identifier of the CR (the cr number) then the class used for the CR should have a user enterable ID. Choose **(ID)** as the field to map to when prompted and this will be used as the CR number.

If the CR number (or requirement ID) is specified in the document (as described above) **AllChange** will attempt to identify any relationships between CRs/ requirements imported and reflect these relationships in the CRs Affected list on the CR.

Only parent/ child relationships are recognised and the *child* will point to its *parent* (i.e. the CRs Affecting the child will be the parent).

Parent/ child relationships are recognised where the CR number is terminated by a sequence of:

`<number>.<number> . . .`

A parent is identified if its CR number matches the child's number excluding the last `.<number>`. For example **RQ_1.1** will be identified as the parent of **RQ_1.1.3**.

The entire text of the requirement/ CR, including both mapped and unmapped fields, will be placed in the text of the CR.

Updating a Change Request

About Updating a Change Request

Many of the fields of a CR may be updated from the CR viewer. Simply select the new value(s) required for any editable fields in the viewer and then select **Misc | Update**. Alternatively the **Update** button on the toolbar may be used.

The screenshot shows a window titled "View CRs - SCR00002". At the top, there is a dropdown for "CR Number" set to "SCR00002". Below this are several tabs: "General", "Text", "Relationships", "Votes", and "Status Log". The "General" tab is active, showing fields for "Class" (SCR), "Status" (Accepted), "Originator" (Tutorial), "Date" (09/01/2007 18:14:22), "Summary" (to create new functionality to meet user requirement 019/0399), "Top Part" (/SWProduct/Source/Hello), and "Reference" (myref2). There is an "Assigned..." button next to the Originator field. Below the "General" tab are two sub-tabs: "Change Info" and "Prioritisation". Under "Change Info", there are dropdowns for "Type" (Off Spec), "Introduced" (Analysis), and "Discovered" (Design). There is also a "Keywords" field with "Example" and a "Locked" checkbox.

Fields that are not alterable from the CR viewer (**Status, Assignee**) may all be altered using special dialogs.

The **Status** may be changed using the cycle viewer, see [CR Life-cycles](#).

The **Assignee** may be changed using the **Assign CR** command dialog, see [Assigning a Change Request](#), which is available directly from the CR viewer using the **Assigned** button.

An alternative method of updating many CRs at the same time is to select the required CRs in the CR Browser and then select the required action from the **CR** menu (i.e. **Alter, Assign** or **Status**), or by using the toolbar.

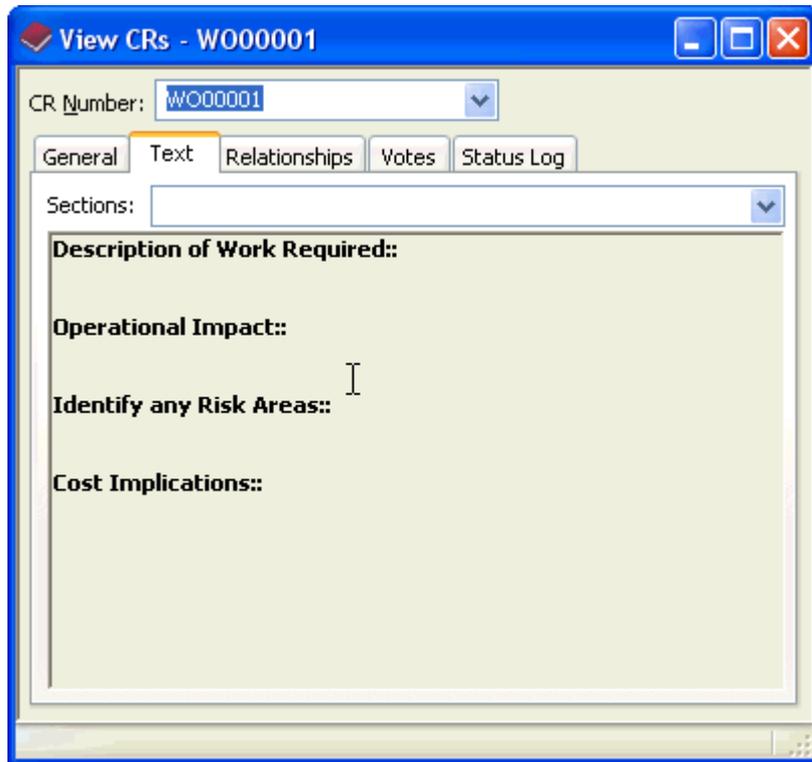
If the CR is of a class which is defined as having *protected text* then the text on the Text tab may not be arbitrarily altered but may only be appended to within each section together with an automatic stamp of the date/time and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the CR menu.

The [Insert into Text](#) facility may also be used with CRs which do not have protected text in order to obtain a date/time/author stamp to any modifications which may be made.

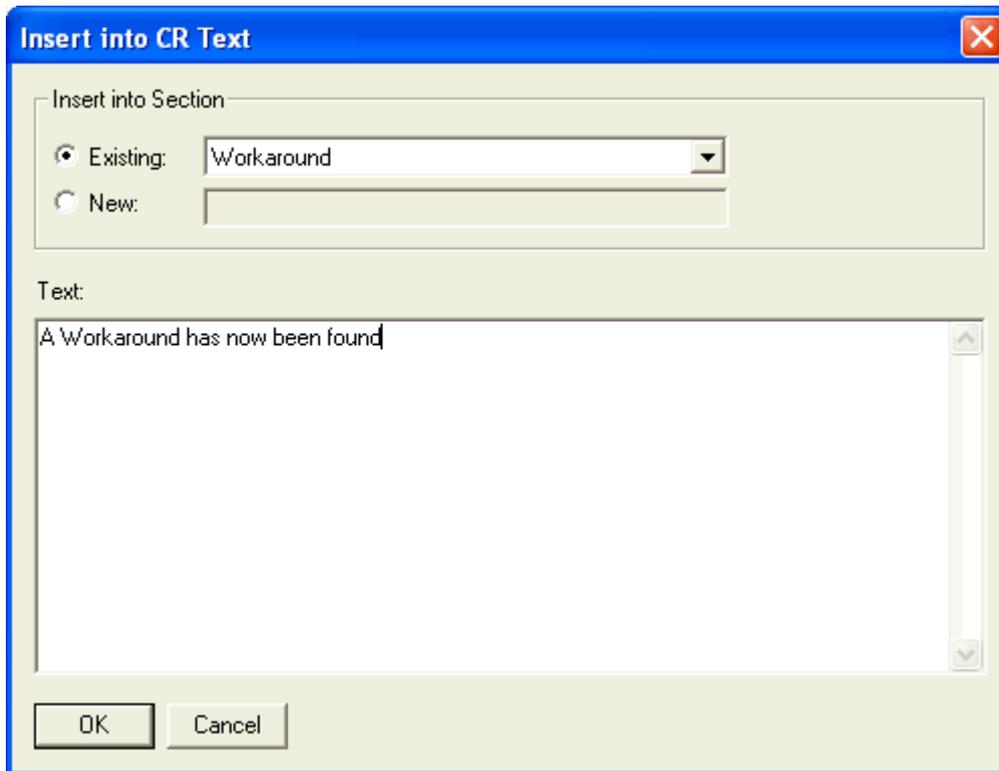
Modifying Text

If the item is of a class which is defined as having *protected text* then the text on the Text tab may not be arbitrarily altered (the edit control is shown read-only) but may only be appended to within each section together with an automatic stamp of the date/time and the user that made the addition. This ensures that text entered may not be modified and a full audit trail is automatically maintained.

[Misc | Options](#) allows the background colour of read-only text to be changed as required.



In order to append to a section, use the **Insert into Text** facility from the CR, Part or Baseline menu, having first selected an item in either a CR/Part/Baseline browser or viewer.



First you must select the section that you wish to insert into. An **Existing** Text Section may be selected from the drop down list or a **New** section may be created by selecting **New** and entering the title for the section in the edit control.

The text that is to be appended to the selected section may then be entered in the **Text** edit control.

The text entered will be appended to the section preceded by a stamp of the date and time and the user inserting the text.

Under certain circumstances it may be desirable to enable certain users to make modifications to protected text, for example to remove an undesirable entry. The users/roles permitted to do this may be defined as a part of the access control definition by the **AllChange** administrator, a user who has this permission will see the Text as editable as with classes which do not have the protect text attribute..

The **Insert into Text** facility may also be used with items which do not have protected text in order to obtain a date/time/author stamp to any modifications which may be made. Alternatively edits may be made directly in the Text tab edit control.

Spell-checking is available in both the Text tab, and the Insert into Text dialog. Note though, that if the text is protected, spell-checking is disabled in the viewer tab. See the [Spell Checking](#) topic for more details.

CR Life-cycles

Like parts, CRs may have life-cycles associated with them. A life-cycle is defined as a series of statuses through which a CR passes and may be used to control access to items, implement approval procedures, log progress, provide an audit trail etc. The cycle-viewer shows a graphical representation of the currently selected life-cycle.

Note that the term status may have been mapped to site specific nomenclature in which case all references to status will show using the mapped nomenclature instead.

The class of a CR is used to determine its associated life-cycle — see [View CR Information](#).

Each status also has associated conditions and actions. A status' conditions must be satisfied before a status change may take place, at which point the status actions will be invoked.

A status may also have vote associated with it which should be completed before the status is progressed, see [Voting](#).

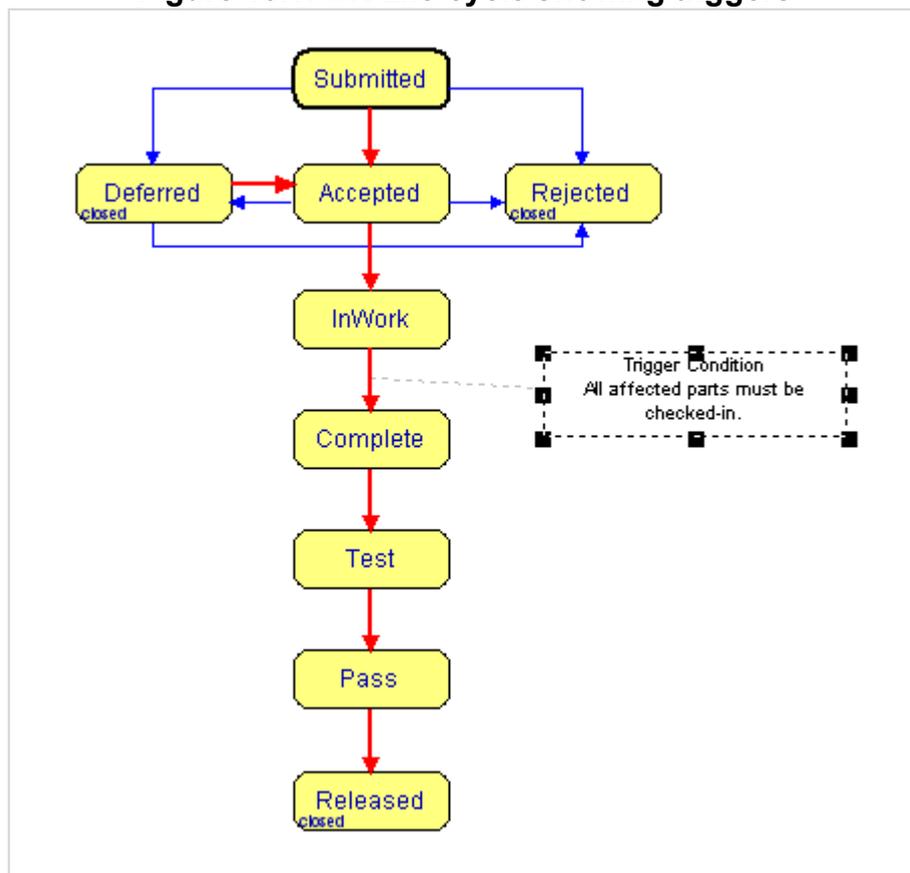
The conditions can be used to implement approval procedures and change control. For example, your site may have been configured to enforce that a part may not be changed unless there is a CR against it and that CR has been approved for action and it is assigned to you.

The actions can be used to trigger events: for example, the life-cycle may trigger the *issuing* of the part-affected by the CR.

In [Figure 10.1](#) a possible CR life-cycle is shown.

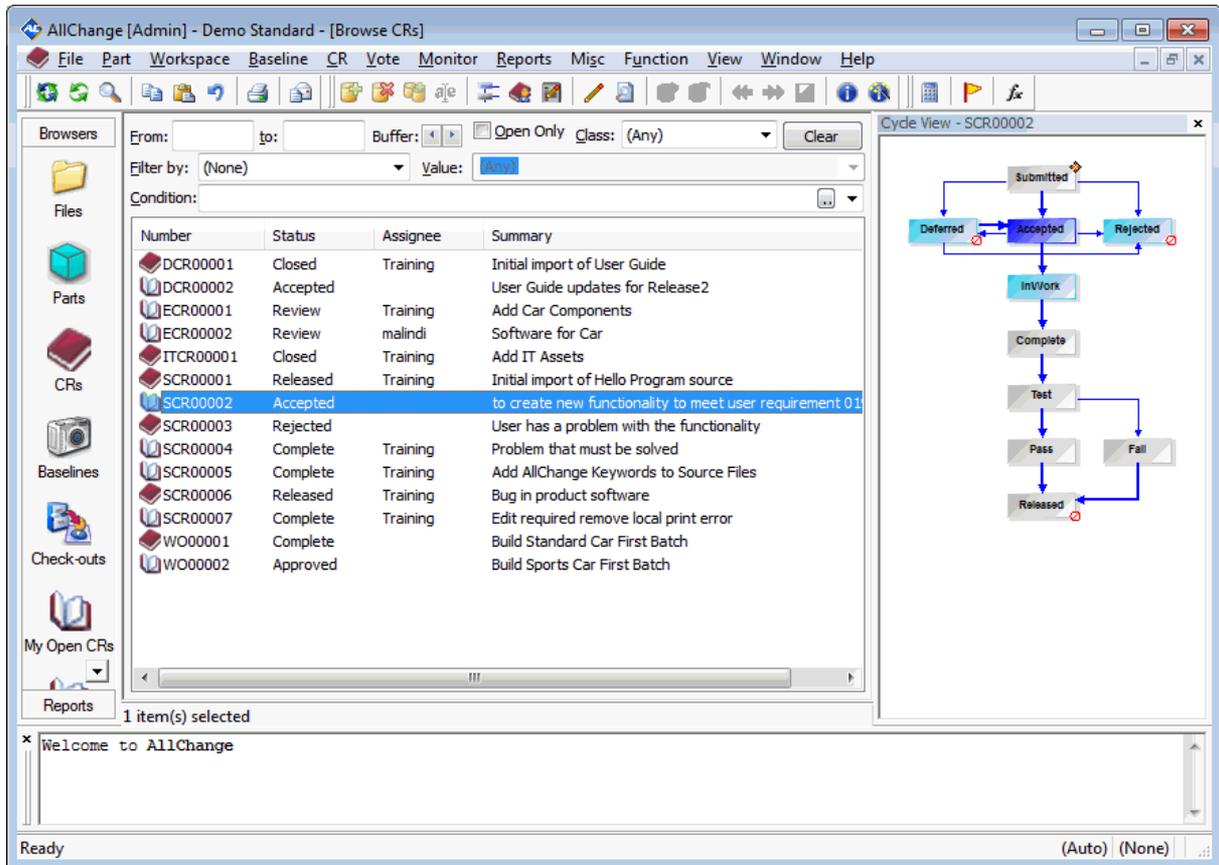
In order to enter the status of **Complete** a condition must be satisfied that ensures that there are no parts checked out for edit against the CR and that for each *partaffected* by the CR there is a *versionsolved* to ensure that all the required changes have been made.

Figure 10.1: CR Life-cycle showing triggers



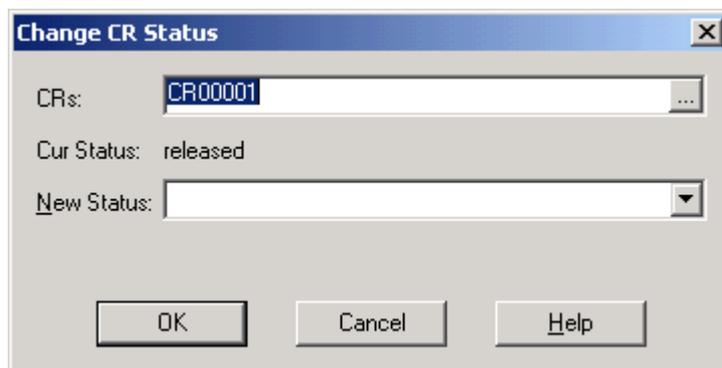
To change the status of a CR use the [cycle viewer](#). With a cycle displayed in the viewer double click on a status which is a valid progression and the status will be changed.

Note that if there is an open vote on the status, then double clicking will open the **Cast Vote** dialog instead of changing the status. If the vote permits status progression (e.g. an advisory vote) then shift double click will allow the status to be changed and the vote will be closed). This is indicated on the tool tip on the status.



Details of all status changes are logged in the status logs database and may be viewed in the **Status Log** tab of the CR viewer. This provides an historical record of the progress undergone by a CR, thus providing an audit trail.

The **Change CR Status** dialog available from **CR | Status** can also be used to change the status of a CR.



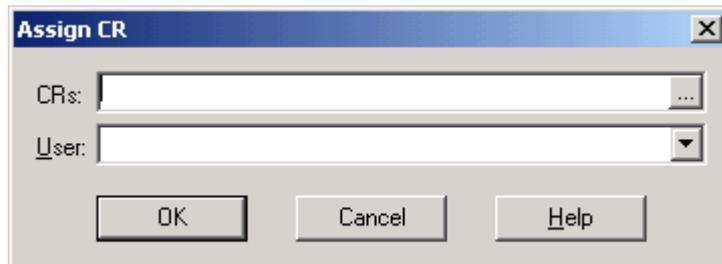
Any CRs selected will be copied as the **CR Num**s. The **New Status** combo box will show a list of the valid progressions from the current state. One of these should be selected.

The **Other** option may be used as required for site specific approval procedures. If it is used the label should have been configured by your administrator to indicate the required input.

Assigning a Change Request

CRs may be *assigned* to an individual *user* or a defined *group* of users, the current assignee will be shown in the CR viewer.

In order to change the current assignee of CRs use the **Assign CR** command dialog available from the CR viewer directly via the **Assigned** button, or from **CR | Assign** or toolbar or context menu.



Any CRs selected in the browser or the viewer will be copied as the **CR Nums** which are to be assigned.

The **User** to be assigned should be selected from the list of **AllChange** registered users and groups.

For each CR assigned the user will be informed of this using the host mailing system, providing this has been configured by the **AllChange** Administrator. If the CR is assigned to a group then each member of the group will be mailed either individually or making use of group mail facilities if this has been specified in the group definition, see the **AllChange Administrator Manual** for details.

This should be used both for assigning a CR for the first time and for re-assigning a CR.

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, the user doing to assigning and the user/ group assigned to are all logged.

Locking CRs

AllChange supports the concept of locking a CR against further change. This may be used, for example, to ensure that no further parts are modified against a CR, that the CR text is not changed any further etc.

Locking of CRs could, for example, be automated as a part of the CR life-cycle.

CR locking is enabled for a class of CR if the CR has an arbitrary field defined which is called **Locked** and has the values **Yes** or **No**. If no such arbitrary field exists, then CR locking is disabled for that class of CR.

Note that if a CR is locked, it may not be used to authorise a check out for edit as this would mean modifying the parts associated with the CR when the part is checked in.

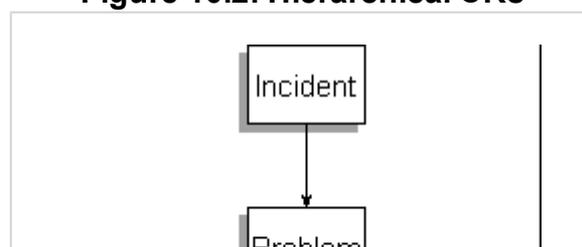
Hierarchical CRs

CRs may be linked to one or more other CRs that they "affect". It is up to the site to define precisely what this constitutes.

This facility may be used to define a hierarchy of CRs so that, for example, *Incidents* could be related to *Problems* raised as a result of the *Incident* which in turn could cause a number of *RFCs* to be raised as used in the ITIL out-of-the-box project configuration.

[Figure 10.2](#) illustrates hierarchical CRs.

Figure 10.2: Hierarchical CRs



Conditions/actions may then be performed on the parent task and/ or the child RFCs in whatever way the site desires: for example, a *Problem* CR cannot be closed until all its constituent RFCs have themselves been closed.

The child RFCs are known as the *CRs affecting (or referring to) the Problem CR*. The *Problem CR* is known as having *CRs affected* of the RFCs.

The CRs *affected by* and the CRs which *refer to* a CR may be viewed and modified in the Relationships tab of the CR viewer. The relationships may also be created automatically, as in the out-of-the-box ITIL configuration.

Selecting CRs

In order to select CRs to perform an operation on either the browser or viewer may be used

Reading a CR into the Viewer

If the CR you wish to view is one of the last 10 that you viewed then you could simply select it from the history list for the **CR Num** in the viewer.

For other CRs the simplest method of reading a CR into the CR viewer is to double click on the required CR in the CR browser.

The **Misc | Next** and **Previous** menu items (or the **Next**, **Previous** toolbar buttons may be used in the CR viewer to read in the next or previous CR to that currently displayed.

You can also type in the full CR number of the required CR as the **CR Num** followed by `Return`, or **Misc | Read** or the toolbar button for **ReRead**. This will cause the specified CR to be read into the viewer.

If a partial CR number is entered followed by a re-read, then the nearest matching CR will be read (e.g. 001 will cause the first CR starting 001 to be read — 00100). If no partial match can be found (e.g. there are no CRs starting 001) an error is issued and the Viewer contents are blanked.

Selecting CRs Shown in the Browser

The CRs shown in the CR browser may be restricted by various methods.

If the **StartCR** is empty then the CRs will be shown from the start of the database. If **StartCR** is a complete CR or partial CR number then the nearest match will be found as the starting point for reading the database.

The **Filter By** may be used to restrict what is shown according to the filter selected and the **Value** specified. If a value is selected then only those CRs which match the value selected for the field filtered by will be shown. The value may be:

- **(Any)** if all items are to be shown.
- **(Current User)** if the filter is a field which holds a user name such as **Assignee**, which will show CRs assigned to the *current user*.
- **(Empty)** will show CRs for which the selected Filter field has an empty value (e.g. CRs which are not assigned to anyone)
- may contain `*` to denote *any other characters* e.g. `starts-with*`, `*ends-with`, `*contains1*contains2*`. If no `*` is used then an exact match is required.
- If the **Filter By** is Text, the value is always treated as though it had `*` at start and end (regardless of whether you type them), and so is always a *contains* match.
- If the **Filter By** is a field which holds a user name (such as the Assignee), the value list will show the users Full Name to select from (if one has been specified by the AllChange administrator) but the data field actually holds the users log on id. This means that if the value is specified as a wild card (e.g. `f*`) then this will be a pattern match on the real data field value (i.e. the users logon id) and not the Full Name

For example, selecting **Assignee** as the **Filter By** and then entering a value of `fred*` will show all CRs assigned to anyone whose log on id begins with `fred` (e.g. `fred` and `freddy`).

The **Condition** may contain any valid ACCEL expression and all CRs displayed in the list will match that condition. The condition may be constructed using the ACCEL condition editor and can be used to restrict the CRs shown to, say, those which have a particular status, or word in the summary.

Selecting CRs Assigned to Me

The CR browser may be used to show only those CRs which are assigned to you.

Simply select the **Filter By** as **Assignee** and the **Value** as your user name.

The browser will be refreshed to contain only those CRs which are assigned to you as the named assignee. Note that this will not include CRs assigned to a group of which you are a member.

To show only those CRs which are assigned to you to include CRs assigned to a group of which you are a member a condition may be specified instead of selecting the Assignee filter.

To specify a condition use the condition editor by selecting the ... button for the **Condition** edit control.

In the condition editor select **Assignee** from the list of available fields and select **Includes** as the operator and select your user name as the value.

On confirming the dialog the condition for the browser will read:

```
is_group_user(cr_assignee, demo)
```

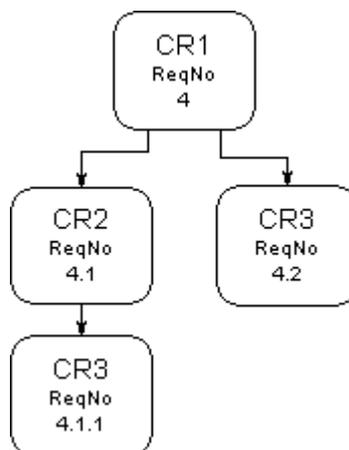
Filter selection may be combined with a **Condition** and whether you want to see **Open CRs Only** so for example if a filter of `Status` is selected with a value of `(Any)` and Open CRs Only is selected in addition to the condition above then only open CRs which are assigned to you will be displayed sorted by status.

Creating and Managing CRs from Doors

It is possible to create CRs directly from DOORS requirements within the DOORS interface.

To create a CR from within the DOORS interface click on a requirement object in a DOORS module and select the '**Synchronise Object with CRs**' option from the **AllChange** menu. If the CR corresponding to the requirement object does not yet exist it will then create a new CR with the contents of that requirement, while also creating new CRs for any further requirement objects underneath the selected object.

AllChange will set up a corresponding hierarchy between the CRs to mirror the levels that the DOORS requirements are in. For example:



To synchronise all of the requirement objects in the DOORS module with **AllChange** select '**Synchronise Module with CRs**' from the **AllChange** menu.

CRs created from Doors will have the class specified in the **DOORSCRCClass** configuration option. This must have the following arbitrary fields defined:

- Project:** The DOORS project containing this CR
- Module:** The DOORS module containing this CR

ReqNumber: Requirement Number used by the CR when using DOORS

(The Standard Configuration has a class of CR, `DoorsRequirement`, defined for this purpose.)

The following built in fields are also used as follows:

Summary: Set to the title of the Doors requirement

Reference: Set to the Doors Object Identifier for the requirement

Integration with Microsoft Project

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 MS Project integration is provided in the form of a function **Export to MS Project** available from the **CR** menu.

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 (look at the **CRs** tab on the **View CR** window), and so on recursively if these CRs affect any further CRs. 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. 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

```

Then select the top-level CR(s) (TSK-00001 in the above example) and invoke the **Export to MS Project** function.

Of course if you do not use the CRs affected facility in **AllChange** you can still view information in MS Project; there simply will not be any hierarchy.

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

This is the date the CR entered the status with the **MS Project Start** attribute e.g for CRs of class SCR this is 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

This is the date the CR entered the status with the **MS Project Finish** attribute e.g for CRs of class SCR this is the *Complete* status. A task that has not yet reached its finish status is set to finish *today*.

Task Anticipated Start Date

This is the date the CR entered the status with the **MS Project Bline Start** attribute e.g for CRs of class SCR this is the *InWork* status. A task with no anticipated start date has this set to the actual start date.

Task Anticipated Finish Date

This is the value of the field specified in the **MSProjectDateBaselineFinishFields** configuration option. 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 life-cycles and arbitrary field definitions, but may be changed to reflect a site's tailoring of these, see the **AllChange** Administrator Manual.

If you create a few CRs to test MS Project integration, fill in the appropriate fields and move it through the appropriate statuses in its life-cycle; do not be surprised if the bars look short/invisible if all the status changes take place on the same day!

Customising the Arbitrary Fields

There are 100 arbitrary fields associated with CRs. The label for these and the valid values may be configured as to your site requirements. This should be performed by the **AllChange** Administrator.

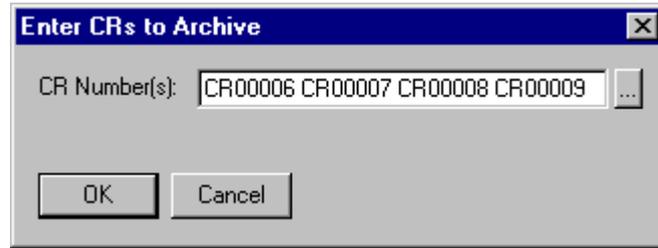
The customised labels will appear in the viewer and any relevant dialogs. The new field name may also be used in ACCEL conditions and will appear in the field list in the ACCEL condition editor.

Archiving of CRs

The facilities to archive (i.e. remove from the current database and re-import at a later date) CRs is intended for removing closed CRs from the database for efficiency.

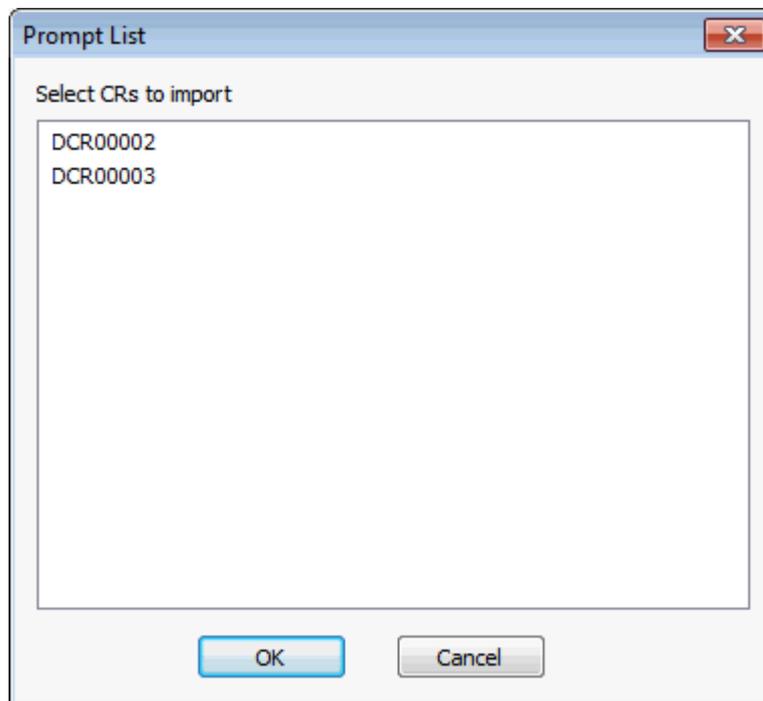
These facilities are available from the **CR | Archive CRs** menu, but are only available to **AllChange** administrators. The Archiving facilities will also only be available if the Archiving feature has been enabled by the **AllChange** administrator, see the **AllChange Administrator Manual**.

To archive CRs use **CR | Archive CRs | Archive**, this will export all information about the selected CRs to an external file in the project directory.



These CRs may then be deleted using **CR | Delete CR**.

If at a later date you wish to re-import archived CRs (e.g. to query them) use **CR | Archive CRs | Import**.



CRs are recreated with their original number and all their original information. 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.

Creating and Managing Baselines

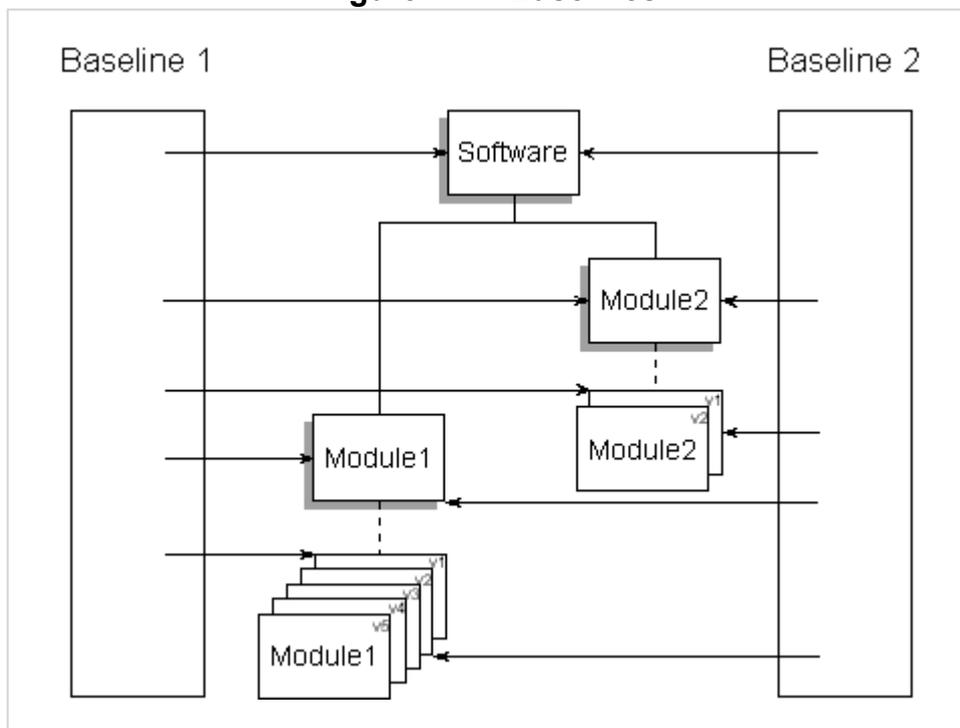
About Creating and Managing Baselines

A baseline is a snapshot of a system or subsystem at a point in time. **AllChange** stores information about what baselines have been taken and what is contained in each baseline.

Note that the term baseline may have been mapped to site specific nomenclature (e.g. release) in which case all references to CR will show using the mapped nomenclature instead.

The purpose of a baseline is to record the state of a section of the parts database at a point in time as illustrated in [Figure 11.1](#).

Figure 11.1: Baselines



There are two kinds of baseline:

Part Baseline: this is a baseline containing parts

Meta Baseline: this is a baseline containing baselines

Information is stored about the type of baseline taken and other general information about the baseline: this is known as the *baseline header*. Information is also stored about all the parts/ baselines that were included in that baseline: this is known as the *baseline detail*.

Baselines may be taken and used to create a snapshot of a system at that point in time which may be immediately locked and used to release the system.

Alternatively, an *incremental* baseline may be used where a baseline is created and then continually modified as changes are made. When a release is to be made the baseline is *frozen* against further change and a new baseline created for the next release.

Each baseline is identified by a name which is comprised of:

< *basename* > [; < *version* >]

The *basename* and the *version* may be any alpha numeric string, plus the following _ - + : , . characters — note that spaces are *not* allowed.

The *basename* must start with a letter, digit or _.

The *version* is optional and may be used to identify, for example, different releases (*versions*) of a product. It may also be used to limit the number of versions of baselines viewed, reported on etc.

Each baseline must be uniquely identified by the combination of its basename and version.

Part Baselines

Part baselines are baselines containing parts as the baseline details. They may have *only* parts as their detail, not baselines.

The parts included in a part baseline may be subject to certain criteria thus allowing a baseline to be very broad or highly specialised in its criteria and scope.

There are three types of part baseline.

A *design* baseline captures the structure of the parts database: it records what subsystems and components exist in the area of the parts database baselined. A design baseline may not include versions of the components baselined.

A *release* baseline, does not capture the structure of the parts database, but includes only information on individual version(s) of component parts. As well as being useful for informational purposes, release baselines may be used to define what versions of what items combine to form a consistent set, secure these versions for potential future reproduction and prepare for actual internal or external release.

An instance baseline captures instances of versions of component parts. This may be useful to capture what instances combine to create a specific product or delivery. Instance baselines may only contain instances of versions of component parts.

The method used to define the contents of your baselines will be closely linked to the release strategy you wish to adopt.

The following scenarios are all supported by the **AllChange** baselining mechanism:

- A *snapshot* for a release baseline may be taken at a particular point in time of a part of the tree using various *selection criteria* to define which specific parts are to be included and which versions of those parts are to be included. This may then be *locked* immediately and used to create a release.
- A *design* baseline may be created specifying which components a baseline is comprised of. When a release is to be made this *design* baseline may be used to create a *release* baseline which specifies which component versions which are included in the release. This release baseline may then be *locked* and used to create a release.
- Releases may be created by specifying the CR's which are to be released and the information associated with each CR is used to create a baseline which incorporates the component versions necessary to implement the CRs.
- An initial baseline may be created and then *incrementally* modified whenever a new component version is *checked in*.

All parts included in a release baseline may be "locked" against future alteration so that, in principle, the items included in such a baseline can be reproduced at a later date.

Meta Baselines

Meta baselines have (the names of) other baselines as their detail; they are effectively baselines of baselines.

Note that the term meta baseline may have been mapped to site specific nomenclature in which case all references to meta baseline will show using the mapped nomenclature instead.

Meta-baselines may contain other meta-baselines in their detail. This makes it possible to maintain small baselines over areas of a system and use meta-baselines to combine these into larger baselines reflecting various product configurations.

Meta-baselines may have *only* baselines as their detail, not parts; When a baseline is created the user must decide whether it is to be a meta-baseline or a normal ("parts-")baseline; this cannot be changed later.

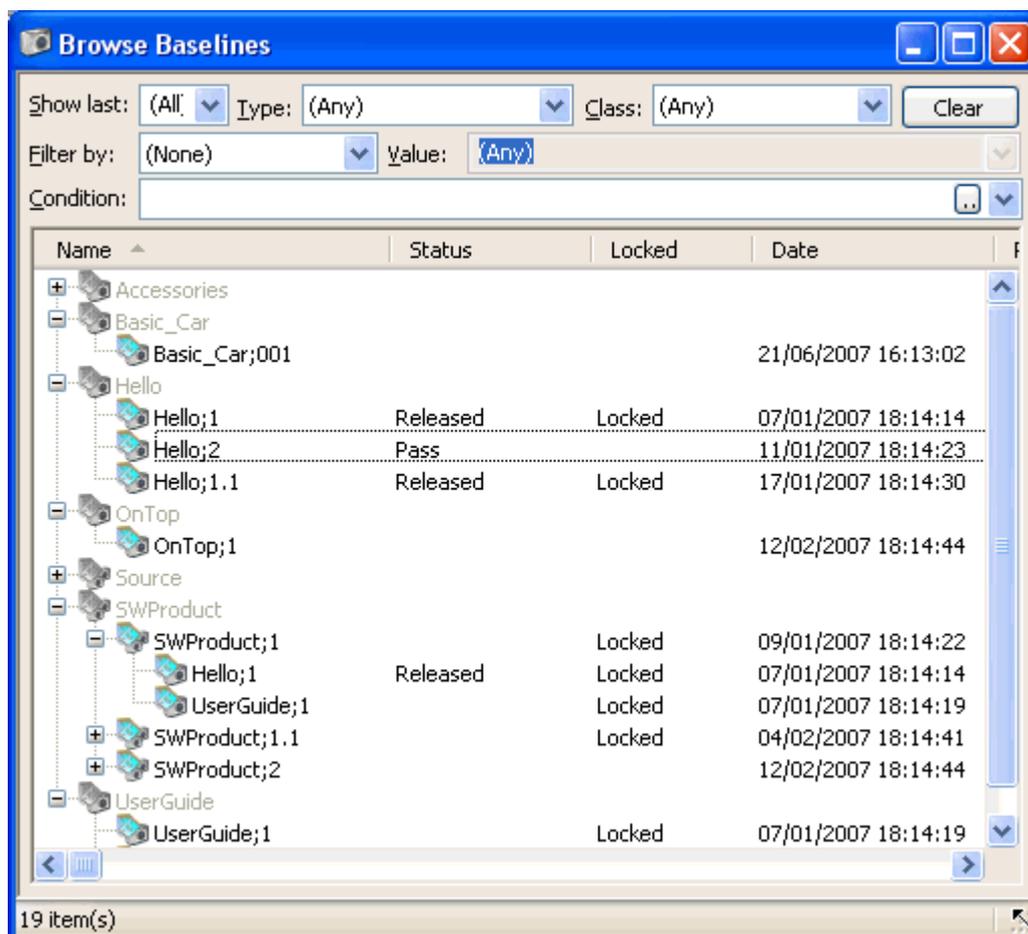
While the details contained in meta-baselines are always baselines, **AllChange** recursively follows these down to determine the actual parts/versions/instances referenced by a meta-baseline when the context requires this. (If multiple baselines referenced by a meta-baseline contain different versions of the same part the result is undefined, i.e. the parts contained in baselines grouped together in a meta-baseline should be discrete.)

It is important to understand that meta-baselines only contain *references* to other baselines, *not* their contents (details). If the details contained in sub-baselines change, so effectively will the parts/ versions ultimately referenced by meta-baselines.

Meta-baselines have several advantages compared to plain parts baselines: they break the process of baselining down into small chunks, they make the constitution of a large product baseline easier to comprehend, they make it easy to maintain variants of products, they allow changing the constituents of one baseline to be automatically propagated to a larger baseline. But they do have disadvantages: precisely because changing the constituents of a sub-baseline affects any meta-baselines in which it appears they must be used with care; and they are slower when searching for an ultimately referenced part since all sub-baselines must be searched recursively. These factors should be taken into consideration when deciding which kind of baseline to create.

Baseline Browser

The baseline browser (available from **Baseline | Browse**) shows a list of the baselines that exist.



The Baseline browser shows the name, the status, comment and other information about the baselines that exist. The icon will show whether the baseline is a part baseline (release, design or instance), a meta-baseline. If **Show attachments flag in icons** option is selected then the icon will in addition have a paperclip to indicate the presence of an attachment.

Icon	Description
	Release baseline
	Design baseline
	Instance baseline
	Part Meta-baseline
	Instance Meta-baseline
	Attachment to Baseline

The list may be filtered to restrict what is shown using either **Show Last** or **Filter By**.

Show Last may be used to limit the number of versions of the baselines that are to be shown to the last *how-many*. This may be **(All)** in which case all versions will be shown. The ordering of baseline versions is determined by the **Release Date** field. The most recent is the one with the most recent date etc. *The version number/ identifier has no significance to the order of baseline versions.*

Show Last may *only* be used in combination with a **Filter by Name**, if any other filter is selected then **Show Last** will revert to showing all versions.

The **Show Last** selection does not affect baselines with no version.

Baselines may be limited to those of a specified type by use of the **Type** control. This allows any combination of Design/Release and Meta/Non-meta to be specified.

The **Class** drop-down list may be used to limit the items listed in the browser to those matching a specified class. Only classes allowed to be viewed by the current user will appear in the list. If only one class name appears in the list, then it will be auto-selected.

The **Filter By** may be used to restrict what is shown according to the filter selected and the **Value** specified. If a value is selected then only those baselines which match the value selected for the field filtered by will be shown. The value may be:

- **(Any)** if all items are to be shown.
- **(Current User)** if the filter is a field which holds a user name such as **User**, which will show Baselines created by the *current user*.
- **(Empty)** will show baselines for which the selected Filter field has an empty value (e.g. baselines which do not have a class)
- may contain * to denote *any other characters* e.g. starts-with*, *ends-with, *contains1*contains2*. If no * is used then an exact match is required.
- If the **Filter By** is Text, the value is always treated as though it had * at start and end (regardless of whether you type them), and so is always a *contains* match.
- If the **Filter By** is a field which holds a user name (such as the **User**), the value list will show the users Full Name to select from (if one has been specified by the AllChange administrator) but the data field actually holds the users log on id. This means that if the value is specified as a wild card (e.g. f*) then this will be a pattern match on the real data field value (i.e. the users logon id) and not the Full Name

For example, selecting **Name** as the **Index** and then entering a value of `Product1*` will show all baselines whose name starts with `Product1`.

The entries shown, as well as matching the specified **Show Last** and/ or **Filter** and **Value** must also match any condition specified in the **Cond** edit control. This may be any valid ACCELexpression.

To find out what baselines are currently active (for example), select the **Condition** editor button and select the function `status_is_open`, on confirming the condition editor dialog, only those baselines whose status is currently an *open* status will be shown.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title.
- The information may be sorted by any column by selecting the column title.
- The columns may be reordered by dragging them and dropping them in the required position.
- [Folding View](#) to show baseline versions and baseline relationships

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

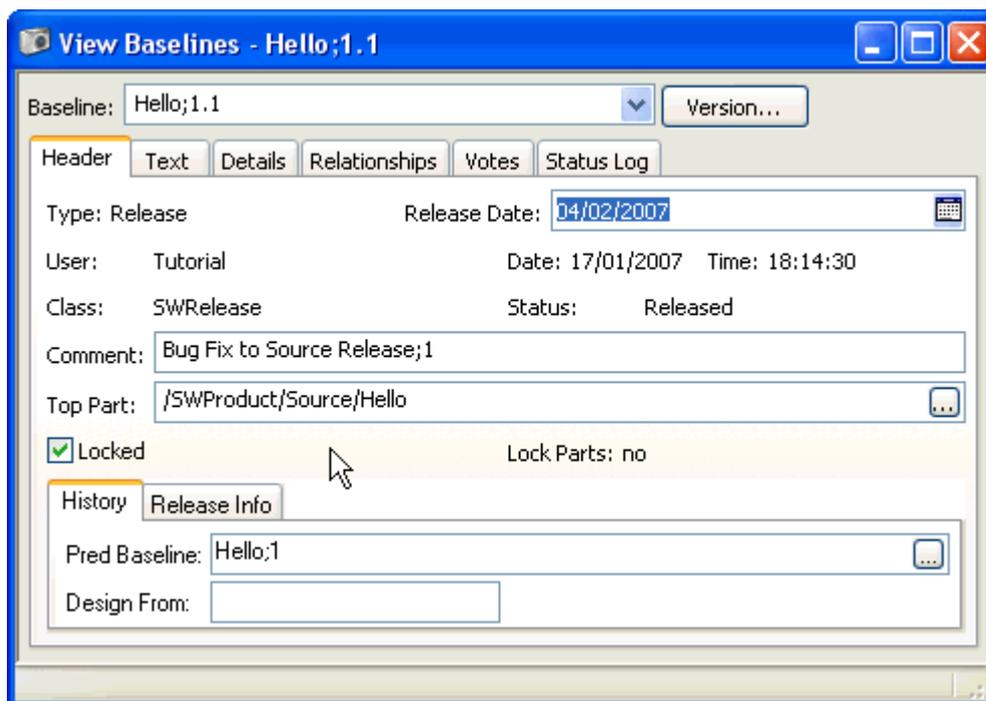
Double clicking on an item in the baseline browser will show the details of the baseline in the baseline viewer.

View Baseline Information

About View Baseline Information

Information about the baselines in the system is stored in the baseline database.

To view the details of a particular baseline and its relationships to other databases associated with it use the baseline viewer; this may be accessed by **Baseline | View**, or by double clicking on a baseline in the baseline browser.



The information about the baseline is shown in a series of tabs. Select the tab required and the information for that tab will be shown.

The tabs for the baseline viewer are:

Header

Shows general information about the baseline which is stored in a baseline header record.

Text

Shows the descriptive text associated with the baseline.

Details

Shows the content (detail) of the baseline.

Relationships

Shows information about relationships to other items such as CRs, other Baselines and attachments.

Votes

Shows the votes/approvals associated with the baseline.

Status Log

Shows the status audit trail for the baseline.

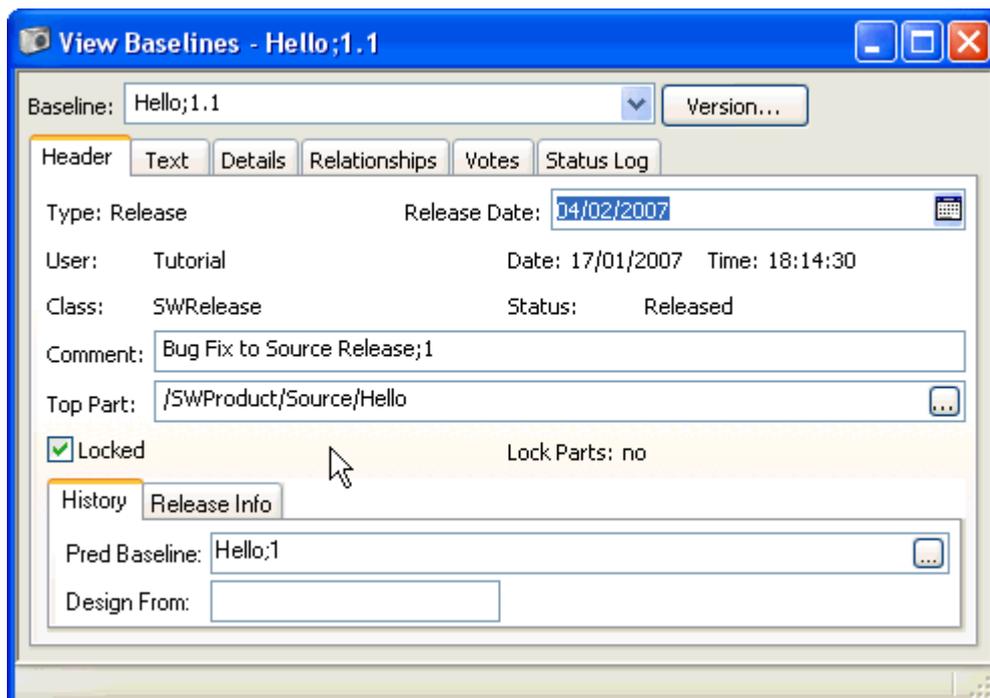
A baseline is identified by its basename and version, the baseline currently viewed is shown in the window title and in the **Baseline** edit control. A baseline name may contain (only) letters, digits and `_ - + : , .` and must start with a letter. The baseline name is made up of:

`< basename> [; < version>]`

The **Version** button allows different versions of the current baseline to be selected and viewed.

Header*The Header Tab*

The header tab shows the fields of the baseline header record.



The fields shown are:

Type:

This indicates the type of the baseline. It may be one of:

- Release: a release baseline containing versions of components
- Design: a design baseline containing components
- Instance: an instance baseline containing instances of versions of parts
- Release Meta-baseline: a meta baseline containing other release baselines
- Design Meta-baseline: a meta baseline containing other design baselines
- Instance Meta-baseline: an instance meta baseline containing other instance or release baselines

Release Date:

The date that the baseline was/will be released.

User:

The user who created the baseline.

Date:

The date on which the baseline was created.

Time:

The time at which the baseline was created.

Locked:

True/ false indication of whether the baseline is locked against further change. A locked baseline may still have its status changed allowing it to progress through its life-cycle .

Lockparts:

True/ false indication of whether the parts in a part baseline were locked. This flag is also used to determine whether to lock any parts added to the baseline.

Class :

The class of the baseline. This may be any of the defined baseline classes as described in the **All-Change Administrator Manual**. The class of a baseline defines its life-cycle if any.

Status:

The current status of a baseline in its life-cycle. This field is set when the status of the baseline is changed and is initialised to the initial status of the life-cycle if there is one, otherwise it is empty.

Top Part:

The full path to the top level part with which the baseline is associated. No parts which are outside of this part of the design tree may be added to the baseline.

Comment:

The comment associated with the baseline.

Arb1... Arb40:

Arbitrary, site-specific information about the baseline.

The **Release Date** is used to determine the order of baseline versions (if any) and is used by, for example, **Show Last** in the baseline browser.

Baselines may be of different (site-defined) classes: this may be used to classify different kinds of baseline such as baselines taken for test or baselines taken for a release. Baseline classes may have an associated life-cycle and baselines may progress through their life-cycle statuses as appropriate.

The lockparts field gives an indication of whether parts baselined were locked. If true then for each part/ version baselined a count in the locked field of the part/ version will be incremented. If the baseline is later deleted then that count will be decremented. (Should the count reach 255 it will neither be incremented nor decremented again.)

The toppart field allows the baseline to be associated with one particular area of the design tree, for example a particular product or the documentation as opposed to the code associated with a product. This assists with the baselining (and releasing) of discrete parts of a system independently of each other. Parts which lie outside the area defined by the toppart may not be added to the baseline.

Locking a baseline ensures that a configuration logged in the baseline is *frozen* and no further changes may take place — an incremental baseline may, for example, be locked when it is to be released.

Baseline Arbitrary Information

Each baseline has forty fields which may be used to hold arbitrary, site-specific information about the baseline. They are displayed on one or more tabs on the viewer as defined for your site. Each site should have a policy as to whether these fields are to be used at all and, if so, what information they should contain. If

these fields are in use the **AllChange** Administrator should have assigned them a name indicative of their function and conveyed this information to users.

Obsolete Baselines

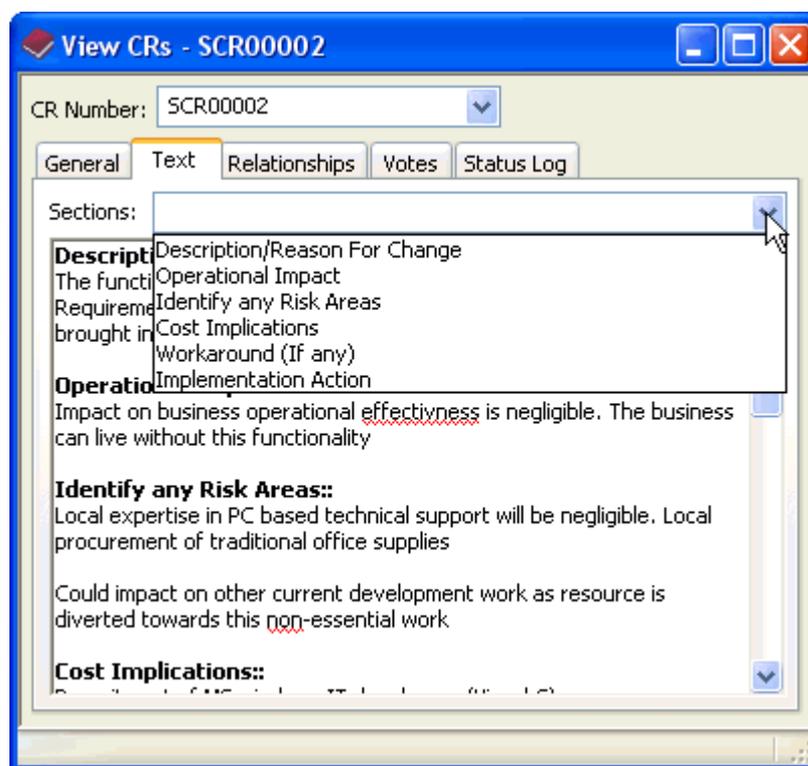
If the baseline is *obsolete* then this will be shown at the top right of the baseline view window as **OBSOLETE**. This flag may be used if a baseline is no longer to be considered but needs to be kept for historical purposes.

This obsolete flag may be changed using **Baseline | Alter | Make Obsolete**. The obsolete flag may be used to restrict the baseline browser, reports etc. to include only those baselines which are not obsolete.

By default obsolete items are not shown in browsers, this may be modified using **View | Show Obsolete Items**.

The Text Tab

The text tab shows the text associated with the CR, Part or Baseline.



No constraints are imposed on its format or contents by **AllChange**. However any line matching a site defined pattern as a section start will be listed in the **Sections**. See the **AllChange** Administrator Manual for details on defining the section pattern. The default pattern is lines ending in : : are treated as sections. Section headings will be displayed in bold. Selecting a section will scroll the text to the start of that section.

Hyperlinks in the text will be shown as clickable links. Hyperlinks are activated for common protocols such as "http", "ftp", "mailto", etc, and also for the "allchange" protocol, and any other registered **AllChange** protocols. If the text is read-only, then the link is followed by single-clicking the link. If editable, then the Ctrl key needs to be held down while clicking to follow the link. Right-clicking on links will show a menu where the link may be opened or copied to the clipboard.

When items are created they have an initial text "template": it is up to the individual site to supply suitable empty template forms for their items and to ensure that users complete these in an acceptable fashion. Different classes of item may have different initial text templates.

Classes of items may be defined as having *protected text*. This means that the text for the item may be entered as normal at creation time, but thereafter may only be appended to within each section together with an automatic stamp of the date/time of the addition and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the CR, Part or Baseline menu. If the text is *protected* the background colour of the text tab will be by default *greyed out*, so show that text may not be typed in directly. The background colour used for this may be changed according to user preferences, see [Customising the User Interface](#) for details.

Text spelling is checked in the text tab. Any misspelled words are underlined with a red wavy line. See [Modifying Text](#) for more information on spelling options etc.

Searching in Text

Text may be searched for a specific word or string. Also text may be replaced using a Find/Replace facility.

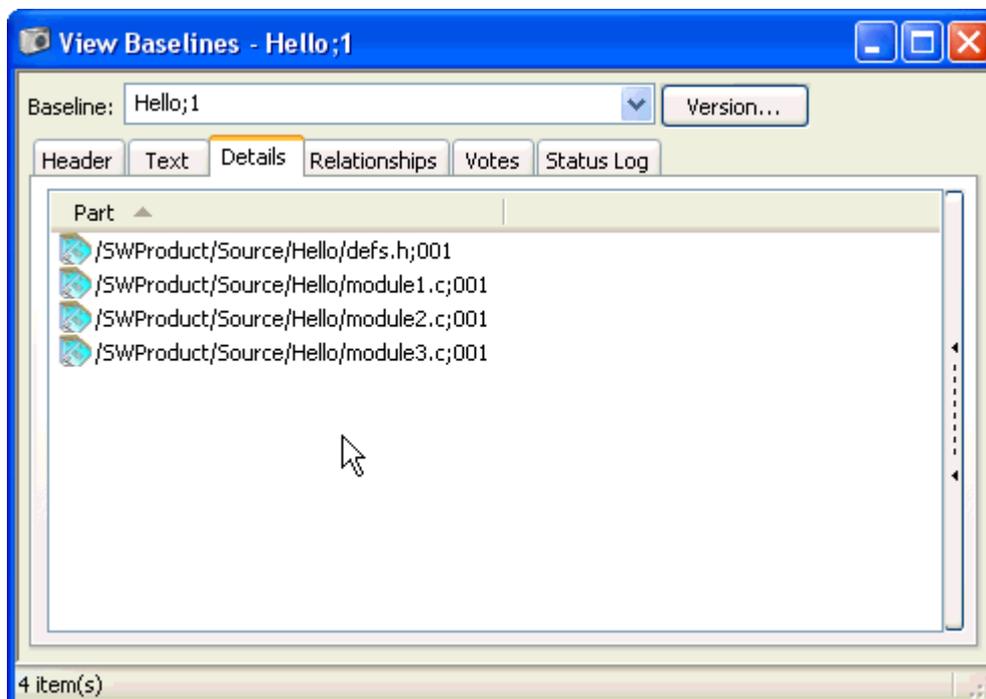
To invoke Find, right-click the text field and select Find, or use the shortcut keys Ctrl+F or Ctrl+D. This will open a dialog box where the search string can be entered. Options may also be specified here to match the case, or match a whole word. Once a word has been searched for, pressing F3 will repeat the search starting from the current cursor position. Pressing F3 without having specified a search string will open the dialog as shown on pressing Ctrl+F/Ctrl+D.

To use Find/Replace, press Ctrl+H, or right-click the text and select Replace. A dialog will open allowing the search string to be specified, along with the replacement string. Options may also be specified here, as per Find. If the text is protected against editing, then the Replace functionality is disabled.

The Details Tab

The Details tab shows the content (detail) of the baseline. The type of item listed will depend on the type of baseline.

Note that the terms baseline and part may have been mapped to site specific nomenclature (e.g. Release, CI) in which case all references to that term will show using the mapped nomenclature instead.



Parts

Shows the parts which a design or release baseline contains. Icons showing the issue state will be

displayed if both the **Show state of baseline details** option and the **Show Issue icon in part lists** options are enabled from the **Browsers** tab of [Misc | Options](#). Note though that enabling these options can cause slowness when running over a slow network or WAN.

Instances

Shows the instances which an instance baseline contains.

Baselines

Shows the baselines which a meta-baseline contains.

The following information is stored about each item in the baseline details database:

name:

The full name of the item contained in the baseline. For a release or design baseline this will be the full path to the part or version. For an instance baseline this will be the full path for the instance. For a meta baseline this will be the full baseline name including the version.

arb1-arb40:

Arbitrary, site-specific information about the baseline detail. This might be used, for example, in a Bill of Materials (BOM) to record the quantity required.

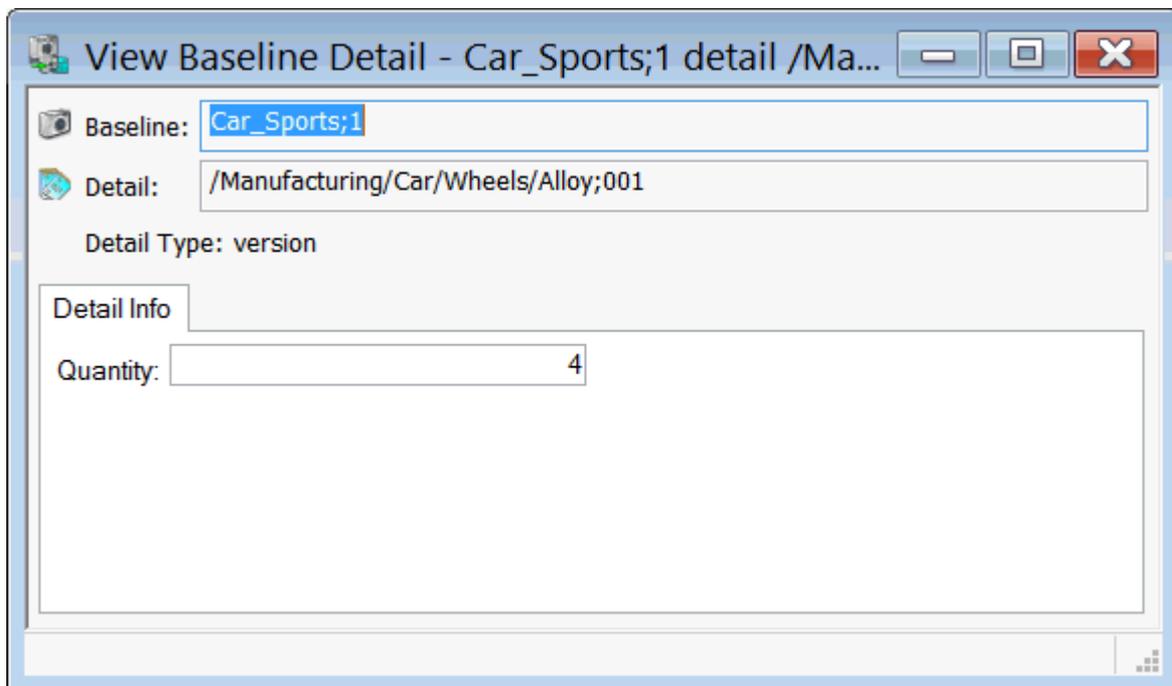
This information may be added using the **Add/Remove Column** dialog in the normal way.

To view the full information for each baseline detail use the Baseline Detail Viewer (available from Right Click menu or the Baseline menu).

Items may be added to and removed from the baseline details using right click context menus or the **Show/Hide Action buttons** button to show appropriate buttons, see [Updating the Details](#).

Baseline Detail Viewer

The Baseline Detail Viewer shows the fields of a specific baseline detail record.



The fields shown are:

Baseline:

The name of the baseline this detail belongs to.

Detail:

The full name of the detail to which the baseline refers. This may be a part or a baseline depending on whether the baseline is a meta or part baseline

Detail Type:

This indicates the type of the detail. It may be one of:

- Baseline
- Instance
- Version
- Part

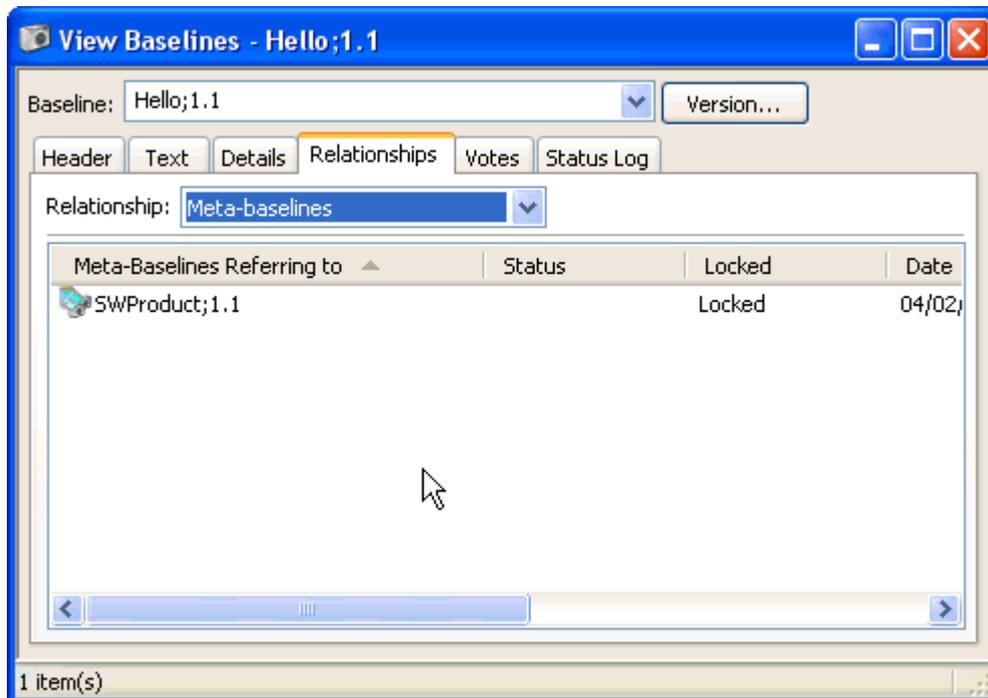
dtarb1-dtarb40:

Arbitrary, site-specific information about the baseline detail. This might be used, for example, in a Bill of Materials (BOM) to record the quantity required. They are displayed on one or more tabs on the viewer as defined for your site. Each site should have a policy as to whether these fields are to be used at all and, if so, what information they should contain. If these fields are in use the **AllChange** Administrator should have assigned them a name indicative of their function and conveyed this information to users.

The Relationships Tab

The relationships tab shows the relationships of the selected baseline to other items.

Note that the terms CR, part and baseline may have been mapped to site specific nomenclature (e.g. RFC, CI, Release) in which case all references to that term will show using the mapped nomenclature instead.



The **Relationship** to be viewed can be selected from the drop down list as follows:

Meta-baselines

Shows a list of the meta-baselines which refer to the baseline viewed; i.e. the meta-baselines which contain this baseline

CRs Affecting

Shows a list of the CRs which are *affected* by the baseline. It shows the CRs which have the baseline selected as a *baseline affected*.

CRs Solved

Shows a list of the CRs which are *solved* by the baseline. It shows the CRs that have baseline selected as a *baseline solved*.

Baselines Affected

Shows the baselines that the baseline selected affects, i.e. point to the selected baseline. Note this is distinct from meta-baselines and is used to allow the linking of baselines for informational purposes only.

Baselines Referring

Shows the baselines that affect the selected baseline, i.e. refer to the selected baseline.

Baselines Affected and Referring

Shows both the Baselines Affected by and the Baselines Referring to the selected Baselines.

Attachments

Shows a list of the files which are *attached* to the baseline. This may be used, for example, to attach release notes or test results to a baseline

Double clicking on a file name in the list will allow you to view the contents of the file using the application associated with that file type in your operating system.

Attachments may be held as *copies* or as *links*. If a copy attachment is used then the attached file will be *copied* to a secure area on attaching it. If a link attachment is used then the path to the file will simply be stored on performing the attachment. Linked attachments will change as the original (and only) file attached changes. A copy attachment will remain as it was at the time of attachment until it is re-attached. On the other hand there is no protection against a file which is attached as a link, from being removed.

Whether copy or link attachments are enabled is site specific and will have been defined by the **All-Change** administrator. Attachments may be defined as always links, always copies or *ask*. If *ask* is used (the default), then on attaching a file you will be prompted as to whether you wish the file(s) to be attached as links.

Any attachments shown with a full path are link attachments, otherwise they are copies.

The **Attach** menu option or button will present you with a file browser allowing you to select the file(s) that you wish to attach to the baseline. For copy attachments the selected file(s) will be copied to a secure area under **AllChange's** control. Each file attached to a baseline must have a unique filename (case insensitive); however, different baselines may have quite different attachments which happen to bear the same filename.

If, at a later date, you wish to replace an attached file with a newer/different version/instance of the file then:

- for attachments which are copies:
 1. use the **Extract To** option on the right click menu to create a (writable) copy of the attached file in another directory
 2. modify the extracted file
 3. attach the file again and it will be re-attached as the new version of the file.
- for attachments which are links:
 1. use the **View** option on the right click menu (or double click)
 2. edit the file and save

Use the **Detach** button to remove any file attachments that are no longer required. Note that

this will remove the copy file created when the attachment was created for a copy attachment and will have no effect on the file for a link attachment .

Baselines which have attachments will be shown in browsers with a paperclip on the icon if the [Show attachments flag on icons](#) option is selected.

As appropriate for the relationship shown there may be actions performed to add or remove items from the list. These actions will be available from right click context menu, or from buttons available when the *show/hide action buttons* button to the side of the list is selected. Double clicking on an item in a list will open the viewer for the item selected.

Additional details about the relationship may be view using **View *database* Relationship** menu item on the right click context menu, where *database* is CR or baseline as appropriate. This is also available from the appropriate main menu, **CR or Baseline | View *database* Relationship**

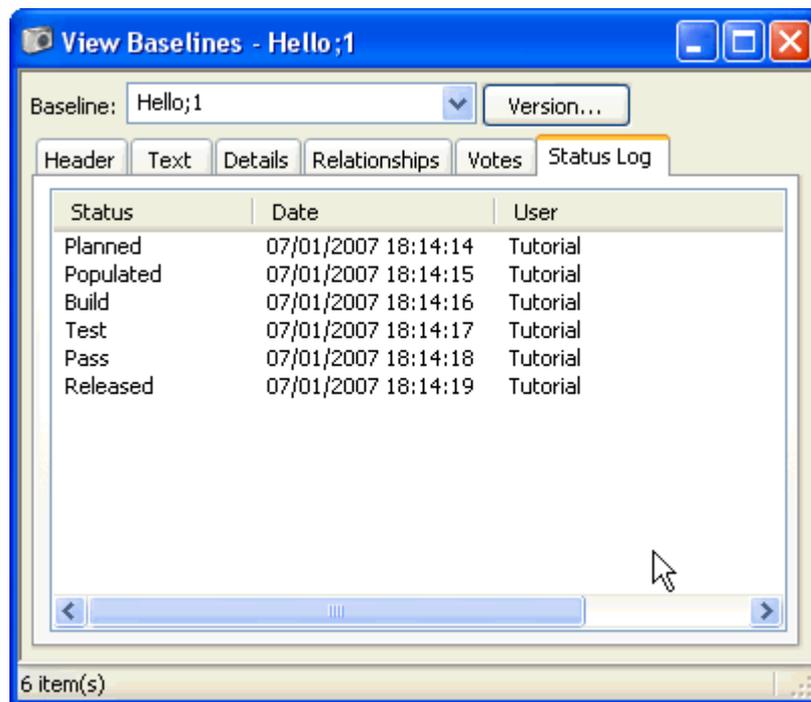
All relationships for each of CRs, Parts and Baselines may also be browsed using the appropriate relationship browser available from **CR, Part or Baseline | Browse *database* Relationships**

The Status Log Tab

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled) and may be reported on, or viewed in the **Status Log** tab.

The status log therefore provides a historical record/ audit trail of the progress undergone by an item.

Note that the term status log may have been mapped to site specific nomenclature (e.g. audit trail) in which case all references to status log will show using the mapped nomenclature instead.



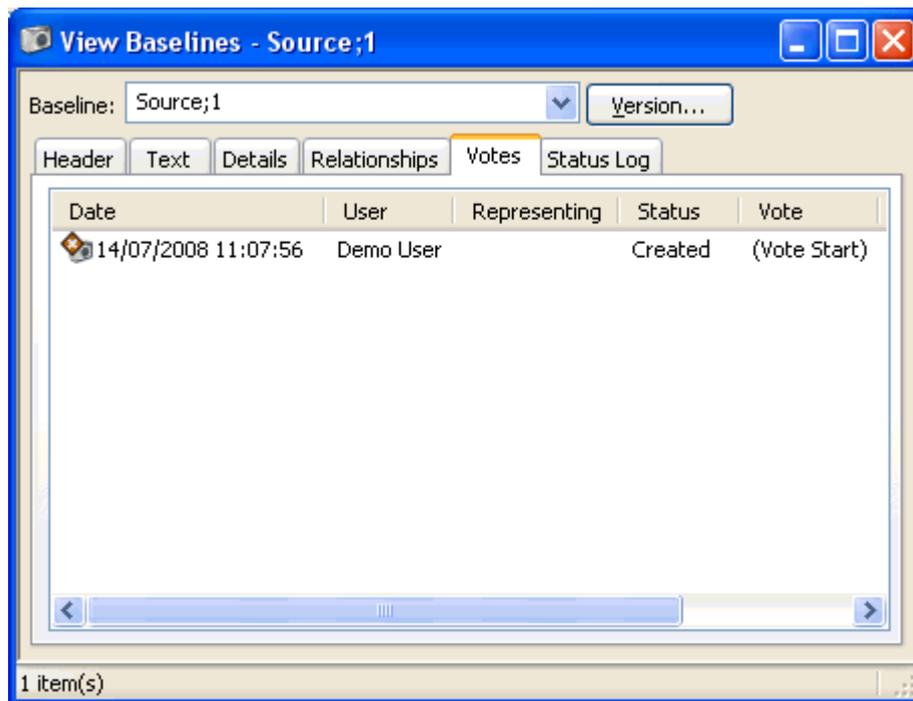
The status log will show the statuses that the baseline has been through, showing the status, the date and time the status was entered and the user who entered the status.

The status log has one field which may be used to hold arbitrary, site-specific information about the status log. This may be shown if it is used.

The Votes Tab

The Votes tab shows a list of all the votes which have been cast for the baseline.

Note that the term vote may have been mapped to site specific nomenclature (e.g. approval) in which case all references to votes will show using the mapped nomenclature instead.



The details of a specific vote may be seen in the Vote Viewer by double clicking on the vote or from the right click context menu.

Votes may be cast when the part is in a vote status by use of the cycle viewer or **Vote | Cast Vote**

Browsing and Viewing Baseline Relationships

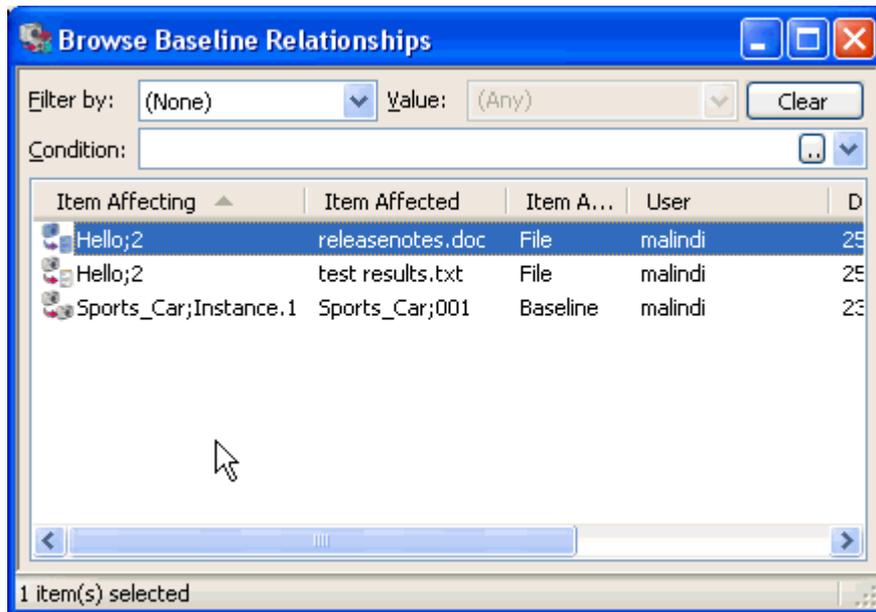
Baselines can be related to files as attachments and to other baselines. These relationships can be examined using:

- [Baseline Relationships Browser](#)
- [Baseline Relationship Viewer](#)
- [Relationships Tab](#) of the Baseline Viewer to view the files related/attached to the baseline and baselines linked to the baseline

These relationships may be created and deleted from the **Relationships** tab on the Baseline Viewer. They may also be deleted from the Baseline Relationships Browser.

Baseline Relationships Browser

The Baseline Relationships Browser allows you to browse the relationships between Baselines and other baselines and also the files which are attached to them, it is accessible from the Baseline Menu



The precise information shown for each item affected is user definable (see [Customising the User Interface](#)).

The **Filter By** may be used to restrict what is displayed according to the **Value**.

The **Condition** may be used to further limit the items affected shown to any arbitrary criteria.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.
- The information may be sorted by any column
- [Folding View](#) showing the relationship relationships for each CR

These features are common to all browser windows, see [Browsers](#) for a more detailed description.

Double clicking on an item in the browser will show the details in the Baseline Relationship viewer window.

Baseline Relationship Viewer

Information about the Baseline relationships to files attached to them and links to other baselines is stored in the baseline relationships (also known as items affected) database. Baseline relationships have a number of fields associated with them.

To view the details of a particular baseline relationship use the baseline relationship viewer; this may be accessed by **Baseline | View Baseline Relationship**, or double clicking (or using the right click context menu) on a baselinerelationship in the baseline relationships browser or right click on an item on the Relationships tab in a Baseline viewer.

The Baseline Relationship Viewer shows the detailed information about a baseline relationship to another item



The fields are:

Item Affecting:

The Baseline which holds the relationship.

Item Affected:

The item which is related to the **Item Affecting**. This will be another baseline or a file and the icon next to it will denote this.

User:

The user who created this relationship. *Note this may be blank for data created prior to version 7.2 of AllChange*

Date:

The date when this relationship was created. *Note this may be blank for data created prior to version 7.2 of AllChange*

Arb1... Arb40:

Arbitrary, site-specific information about the item affected relationship. The field names may be configured as required for your site.

Creating a New Baseline

About Creating a New Baseline

There are two methods of creating a baseline:

1. Taking a *snapshot* of a system using [Add Baseline](#)
2. Creating a new baseline from an existing one using [Copy Baseline](#)

The method which best supports your release strategy should be used.

Creating a Meta Baseline

Meta-Baseline should be selected in the general tab of the **Add Baseline** dialog.

The baselines to be included in the meta-baseline should be specified on the details tab. These may be meta-baselines themselves but should ultimately resolve to a discreet set of parts

A **Design** meta-baseline comprises only design meta or part baselines, whilst a **Release** meta baseline comprises only release meta or part baselines.

An **Instance** meta-baseline may comprise instance baselines and release baselines

Add Baseline

Add Baseline

In order to take a baseline use the **Add Baseline** dialog available from **Baseline | Add** or from the toolbar when the baseline browser or viewer is the current window.

Fields which are compulsory will be shown in red and a value must be supplied.

The screenshot shows the 'Add/NewVersion Baseline' dialog box. It has a blue title bar with a close button. The dialog is divided into several sections. At the top, there are three tabs: 'Header', 'Text', and 'Parts', with 'Header' selected. Below the tabs are several input fields and checkboxes. The 'Basename' field is a text box with a browse button. The 'Version' field is a text box with a browse button. The 'Type' field is a dropdown menu set to 'Release', with a 'Meta-baseline' checkbox. The 'Class' field is a dropdown menu with a 'Header Only' checkbox. The 'Release Date' field is a text box with a calendar icon. The 'Top Part' field is a text box with a browse button. The 'Comment' field is a text box. Below these are two checkboxes: 'Lock Baseline' and 'Lock Parts'. At the bottom, there is a 'History' section with a 'Pred Baseline' field. At the very bottom are three buttons: 'OK', 'Cancel', and 'Help'.

Each baseline is given a name which is comprised of the **Basename** and the **Version**; the **Version** is optional.

The baseline name used to identify the baseline will be:

`< basename> [; < version >]`

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.

If a new version of an existing basename is to be created then the **Basename** may be selected and a new **Version** should be specified.

If an entirely new baseline is to be created the required **Basename** and **Version** should be specified.

Baselines may be of **Type Design, Release** or **Instance**. A Design baseline contains information as to the content of the baseline but not specific details such as which versions of components. A Release baseline is specific to a particular release or snapshot and identifies precisely the content of the baseline. An Instance baseline contains information as to which specific instances of versions are in the baseline.

The **Class** should be selected as required for the site: there may be, for example, different classes of baseline for test and release purposes. The class will determine the life-cycle for the baseline if there is one.

If **Header Only** is selected, then an empty baseline will be created containing no items.

The **Comment** may be used to specify any arbitrary comment about the baseline (e.g. the reason for the baseline)

If **Lock Baseline** is selected then the baseline will be locked against further change once it has been created.

In addition if used, arbitrary field contents may be specified.

The above information may be specified regardless of the type of baseline to be created.

Creating a Part Baseline

Release, Design or **Instance** should be selected in the general tab of the **Add Baseline** dialog.

If **Design** is selected then no versions of components will be included in the baseline.

If **Release** is selected then only versions of components will be included.

If **Instance** is selected then instances of versions of components will be included.

The parts to be included in the baseline should be specified on the **Parts** tab.

One or more starting **Parts** should be specified as the root(s) of the subtree(s) to be baselined. The baseline proceeds by considering all descendants of these subtrees (only) as candidates for inclusion in the baseline, i.e. only that area of the parts database rooted at the starting node(s) is baselined.

The **Versions** of components to be included as candidates for the baseline must also be specified if creating a release or instance baseline: these may be **All, Default, Registered** or **Top**.

If creating an instance baseline then **All** or **Top** must be selected for the instances to be included for the selected versions.

A **Condition** may be supplied by the user to further limit which items are baselined; all candidates for inclusion in the baseline are submitted to the condition and only added to the baseline if the condition succeeded. Thus a baseline may be very broad or highly specialised in its criteria and scope.

The condition may be used, for example, to limit the items in the baseline to only those which have a specified class and status .

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

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. Lock Parts is not available for instance baselines.

If **Follow Uses** is selected, then any parts of type **uses** encountered as candidates for inclusion in the baseline will be included and the part *used* will be considered as a candidate together with all its descendants.

Design Baselines

A design baseline is a baseline which includes information about the product structure (possibly) as well as information about specific components. This means that the existence of subsystems and components making up the parts tree is stored in the baseline.

A design baseline which specifies which components make up a system may be used to:

- check out the default version of all the components in a design baseline
- create a *release* baseline which contains specific versions of all the components in a design baseline

In order to use either of the above it is necessary that the baseline does not contain any subsystems. You should therefore specify a condition of `pa_type != subsystem`, this may be selected in the condition editor.

Release Baselines

A release baseline is a baseline which does *not* include any specific information about the parts structure. It may contain only component versions.

Instance Baselines

An instance baseline is a baseline which includes specific instances of versions of parts and may be used to reflect a specific product comprised of specific instances.

Creating an Empty Baseline

It is often useful to create an empty baseline and then add items to it as they are developed.

In order to create an empty baseline, simply select **HeaderOnly**.

Copying a Baseline

It may be useful to create a new baseline based on an existing one. Some examples are given below as to when this may be useful:

- In order to create a baseline for the next release/ version of a product based on the current release which has just been frozen. This will then be modified for the next release.
- In order to create a release baseline for the next release from a corresponding design baseline
- To create a new empty baseline with the same header information as a previous baseline, which is to be populated later.

This may be achieved by copying a baseline to create a new baseline which is based on the original. This functionality is supplied as a site modifiable function available from **Baseline | Copy Baseline**.

The screenshot shows a 'Copy Baseline' dialog box with the following fields and options:

- Old Baseline:** Hello;1
- New Baseline:**
 - New Version
 - New Baseline
- Basename:** Hello
- Version:** (empty)
- Release Date:** 16/07/2008
- Type:** Release (dropdown), Meta-baseline
- Update Details To:**
 - None (leave empty)
 - [No Change]
 - Default Version
 - Registered Version
 - Top Version
- Buttons:** OK, Cancel, Help

The baseline to be copied should have been selected and the name for the new baseline may be specified as the new **Basename** and **Version**.

You should select whether you wish to create a **New Version** of the existing baseline or create an entirely **New Baseline**. A new version will have the same type as the existing baseline (i.e. design or release), a new baseline may be of a different type. This allows, for example, a release baseline to be created from a design baseline.

If a new version of the baseline being copied is to be created then only the new **Version** needs to be specified.

To create a new version of another baseline (e.g. creating a release baseline from a design baseline), select the appropriate **Basename** and enter the new **Version**.

To create an entirely new baseline enter the **Basename** and if required **Version**.

The **Update versions to:** options allow you to specify whether you wish any components added to the new baseline to have the specified version added as the version in the new baseline.

If **None** is selected then an empty baseline is created. This may be useful for creating a new baseline which is intended to be released at some point in the future, but for which the content is not yet known.

If **No Change** is selected then the new baseline will have exactly the same details as the baseline copied from provided the type is compatible. This should be selected if you wish to copy *from* a design baseline to another exact copy design baseline.

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. This may be used to create a release baseline from a design baseline; the new baseline will be created as a release baseline regardless of whether the original baseline is a design or release baseline.

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.

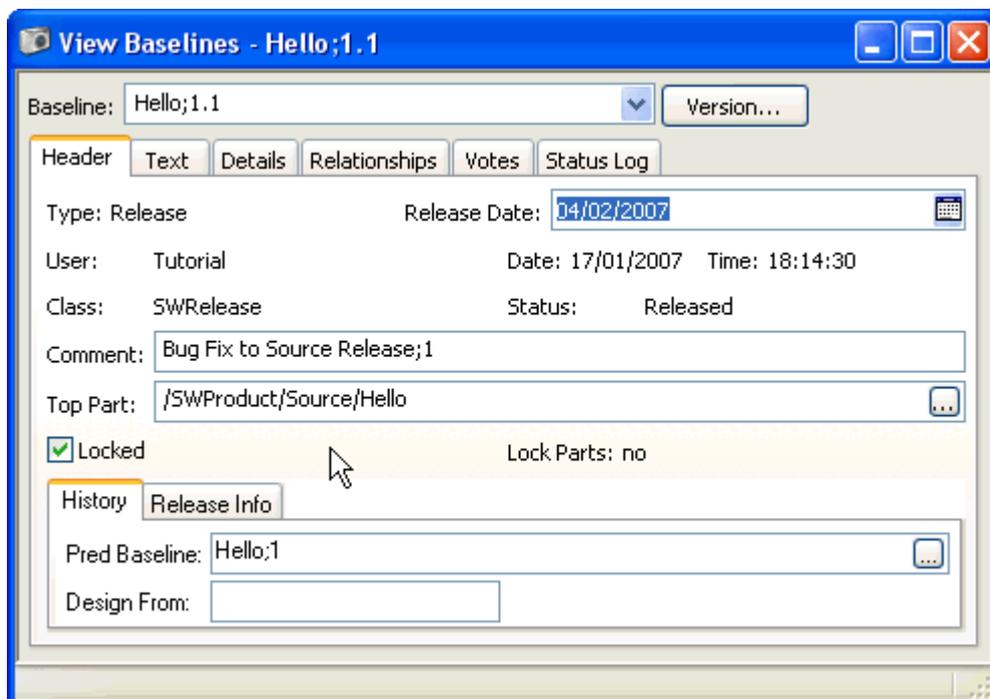
Another use would be to create new releases as the latest version of the items from the previous release.

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.

Updating a Baseline

Updating the Header

Many of the fields of the baseline header may be updated from the baseline viewer. Simply select the new value(s) required for any editable fields in the viewer and then select **Misc | Update**. Alternatively the **Update** button on the toolbar may be used.



The **Status** may be changed using the cycle viewer, see [Baseline Life-cycles](#).

Other fields which are not editable directly in the viewer may not be altered at all.

An alternative method of altering the fields of more than one baseline at a time is to select the required baselines in the browser and then select the required action from the **Baseline** menu, i.e. **Status** or **Alter**.

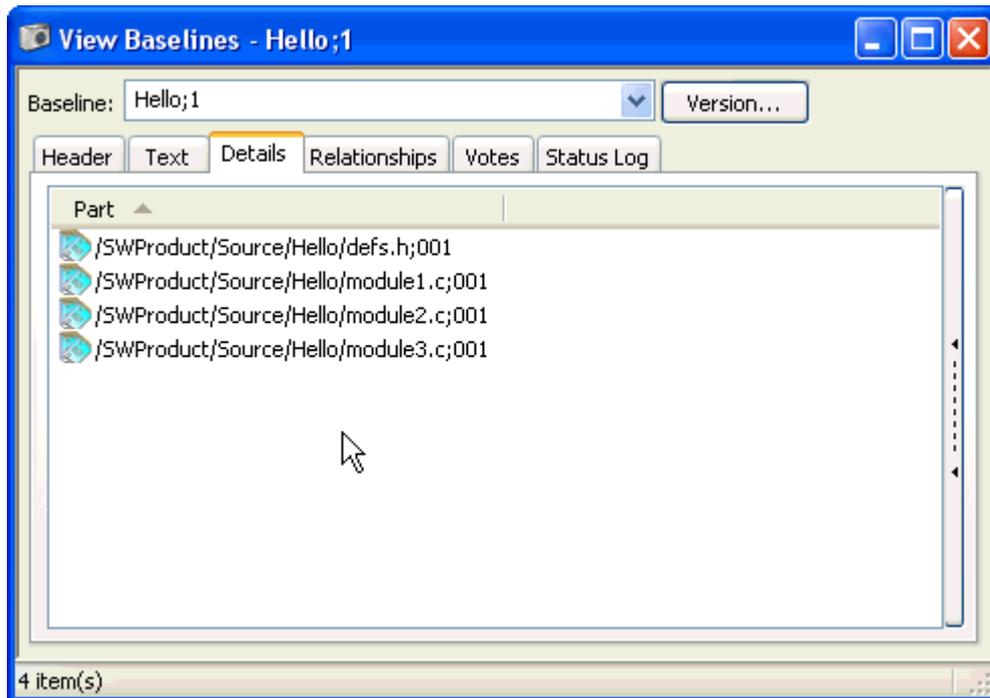
If the Baseline is of a class which is defined as having *protected text* then the text on the Text tab may not be arbitrarily altered but may only be appended to within each section together with an automatic stamp of the date/time and the user that made the addition. In order to append to a section use the [Insert into Text](#) facility from the Baseline menu.

The [Insert into Text](#) facility may also be used with Baselines which do not have protected text in order to obtain a date/time/author stamp to any modifications which may be made.

Text spelling is checked in the text tab. Any misspelled words are underlined with a red wavy line. See [Modifying Text](#) for more information on spelling options etc.

Updating the Details

The parts/baselines/instances contained in a baseline may be added to or deleted from a baseline using the details tab on the baseline viewer. Baselines may also be automatically updated via life-cycles or the [CRs For Baseline](#).



The right click context menu provides options to modify the baseline details or the Hide/Show actions button may be used to make action buttons available.

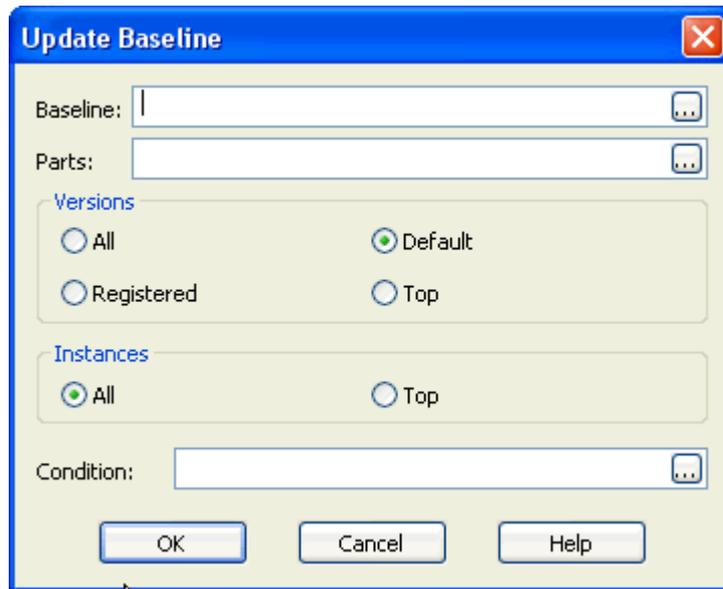
Add

Add allows new parts/baselines/instances to be added to the baseline. A part/baseline list dialog will be shown and any selected items will be added to the baseline on confirming the dialog.

If the item already exists in the baseline then it will be updated to the latest information on the item. If a new version from the one currently in a part baseline is specified, then it will replace the original version specified.

Update

The **Update** button allows the contents of a part/instance baseline to be updated.



Any **Parts** specified should be subsystems and will be recursively descended updating each part in the baseline as required.

Selecting **Versions** of one of **Default**, **Registered**, **All** or **Top** with a release baseline will add/replace the corresponding version of each component in the hierarchy. The **All** option for a release baseline should only be used in combination with a condition which will filter out all but 1 version.

For an instance baseline selecting **Instances** of **All** or **Top** will cause those instances of the selected **Versions** (**Default**, **Registered**, **All** or **Top**) to be added to the baseline.

If the baseline is a *design* baseline then it will be updated for any new components or subsystems which are not in the baseline.

With any of these options, each part added/updated must satisfy any *Condition* which is specified. A condition of `pa_type == component`, for example, may be used with design baselines to ensure that only components and not subsystems are added to the baseline.

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

Note that versions/parts are only added or updated in the baseline, never deleted (even if a part no longer exists).

Delete

Delete allows any selected items to be deleted from the baseline. For a part baseline if it was a locking baseline then any parts deleted from the baseline will have their lock count decremented.

Automatically Altering Part Baseline Details

Automatically Altering Part Baseline Details

The parts contained in a release baseline may be automatically updated by:

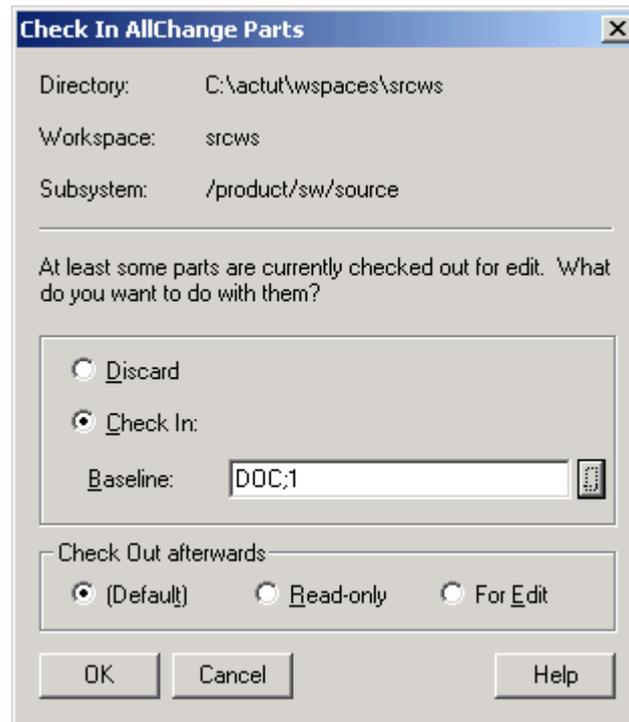
- Virtue of a new version being created on check in from edit
- Updating the baseline based on the CRs which are to be released.

Updating a Baseline on Check In from Edit

The details of a part baseline may also be automatically updated by specifying the name of the baseline to be updated as the **Bline** when the part is checked in from edit.

The baseline specified will be updated to the new version created when the part has been checked in from edit.

This may be achieved using the **Bline** option to the **Check In** dialog.

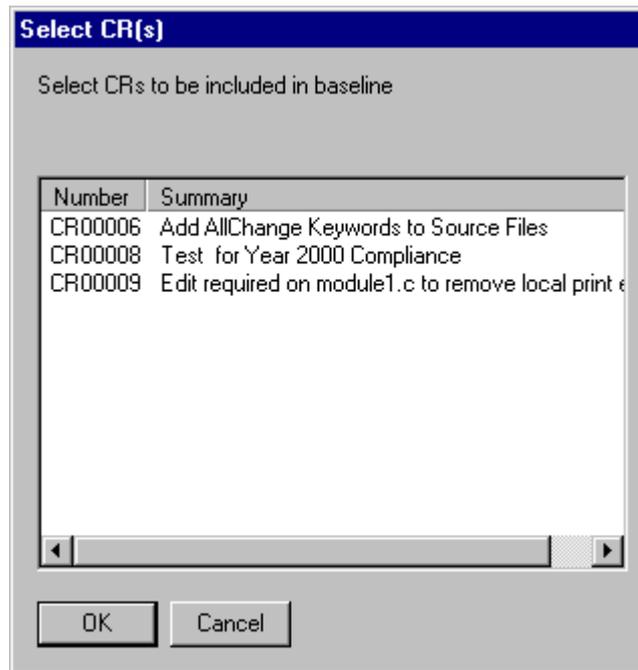


Updating a Baseline from CRs

A baseline may be updated according to the versions associated with certain change requests by use of **Baseline | CRs For Baseline** function. Both part and meta-baselines are supported.

This is a site modifiable function which — as supplied — adds to/ removes from a specified baseline the versions solved by specified CRs and associates the baseline with each CR as a **baseline solved** by that CR.

The CR's 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*.

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 the **TargetRelease** set to the baseline or the the baseline's **TargetRelease** if defined (if the **TargetRelease** field is in use for the CR or the baseline, this is defined by your **AllChange** Administrator)
- have at least one part associated with it which is within the baseline **TopPart**.

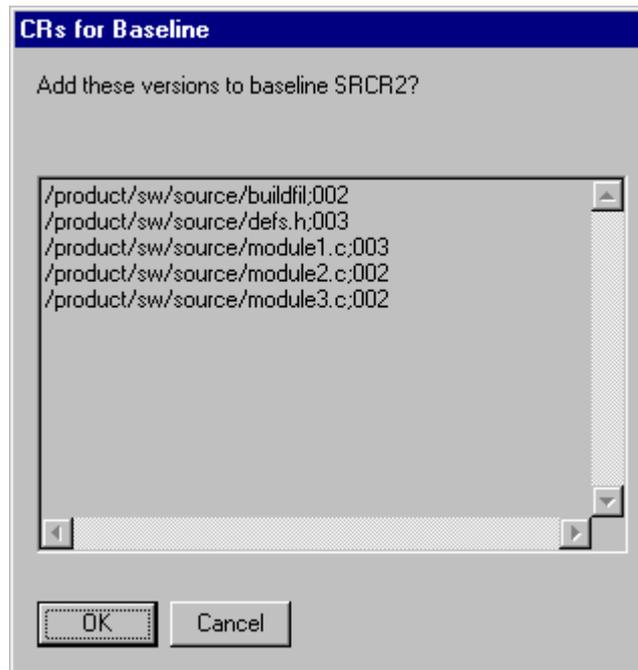
Those CRs which you wish to release should be selected from this list.

The function will then perform the following checks:

1. All CRs related to the CRs selected are also included
2. All CRs are included which are fixed as a result of a versionssolved of a selected CR and which have *not yet been released*

A CR is defined as having *been released* if it has a baseline associated with it as a baseline solved by the CR and it is not in a status which is a *release* status.

It will then calculate a list of the all the latest versionssolved associated with the selected CRs. You will then 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.



On confirmation to modify the baseline you will then be prompted with a list of all of those versions solved which were made obsolete by the CR (either the version or the component) to ask if you wish to remove these from the baseline. A part is regarded as being made obsolete by the CR if:

- The part/version is obsolete
- The CR Part Affected relationship has an Obsolete field defined and the value is Yes

On confirmation of this the baseline will be updated to the versions solved listed to be added and any that were obsolete will be removed from the baseline.

If the baseline is a meta-baseline then new versions of baselines in the predecessor baseline tree are prompted for as appropriate and unchanged baselines in the predecessor tree are placed in the new tree as appropriate, see CRs For Baseline for full details.

Each CR will also have the baseline associated as a *baseline solved*.

Selecting **Cancel** to either of the above prompts cancels the entire operation and no changes will be made to the baseline.

Baseline Life-cycles

Like parts, baselines may have life-cycles associated with them. A life-cycle is defined as a series of statuses through which a baseline passes and may be used to control release procedures etc. The [cycle-viewer](#) shows a graphical representation of the currently selected life-cycle.

The class of a baseline is used to determine its associated life-cycle — see [Header](#).

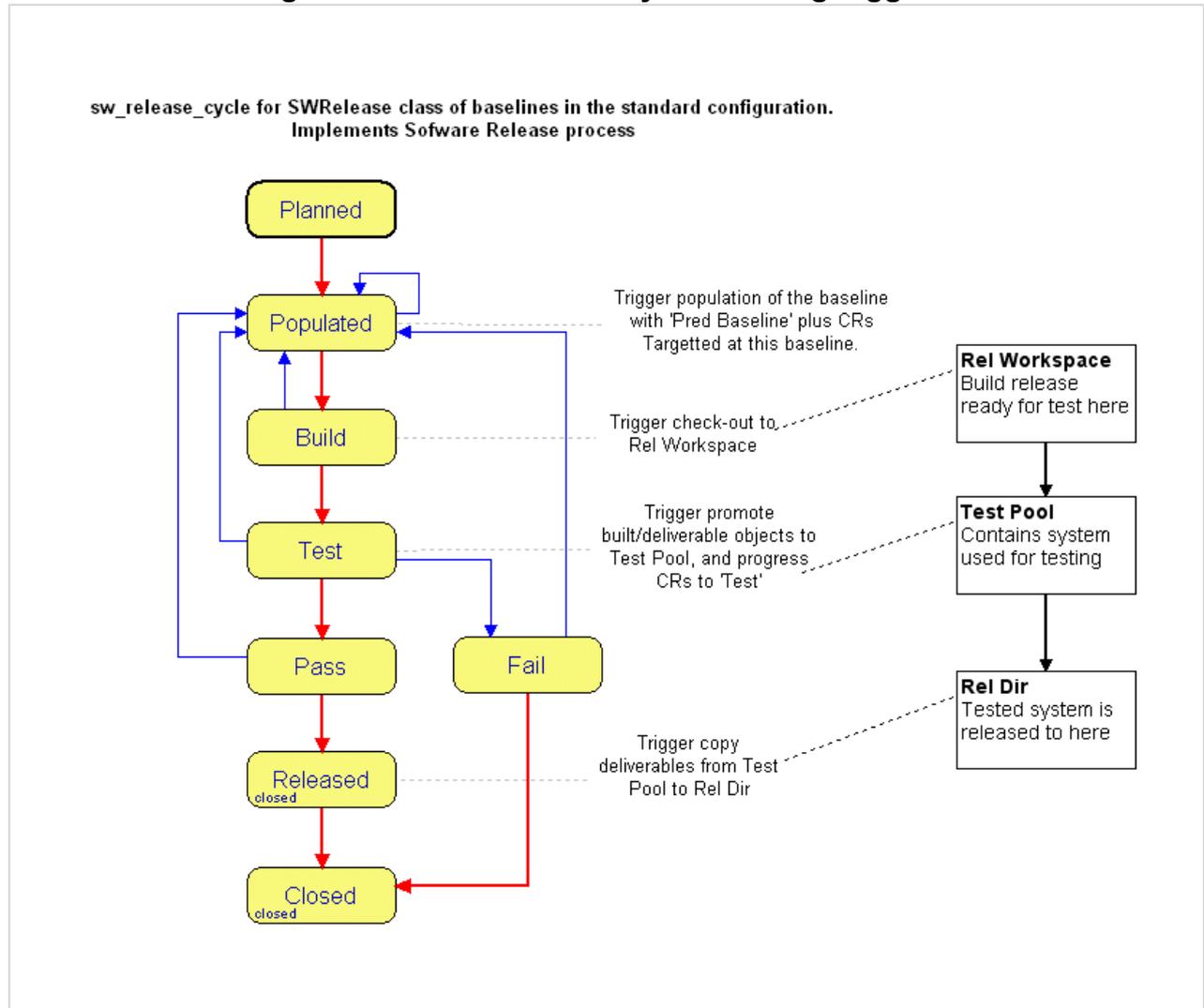
Each status also has associated conditions and actions. A status' conditions must be satisfied before a status change may take place, at which point the status actions will be invoked. A status may also have vote associated with it which should be completed before the status is progressed, see [Voting](#).

The conditions can be used to implement release procedures. For example, your site may have been configured to enforce that a baseline may not be progressed unless it is locked and all CRs which are related to it have been closed.

The actions can be used to trigger events: for example, the life-cycle may trigger the *issuing* of the parts in the baseline to a release area ready for building.

In [Figure 11.2](#) the Baseline life-cycle supplied with *AllChange* for the *SWRelease* class of baseline is shown. This is intended to be used for releasing software source.

Figure 11.2: Baseline Life-cycle showing triggers



Initially the baseline is in a status of **Planned** which indicates that code is being developed. When the code for the baseline has been developed the baseline may be **Populated**. If a predecessor baseline has been specified then the details are copied from the predecessor baseline. This means that the baseline, at this stage, comprises the same component versions as the predecessor.

The baseline is then further populated according specified CRs which are to be released — these will be prompted for.

The baseline details are modified according to the code which was changed to implement the selected CRs. Each CR is also then linked to the baseline. This means that at a glance you can see what CRs are contained within a release.

When ready to actually build the release, the baseline status is progressed to **Build**. This will check out all the parts in the baseline (read only) to the release workspace specified, ready to be built for testing.

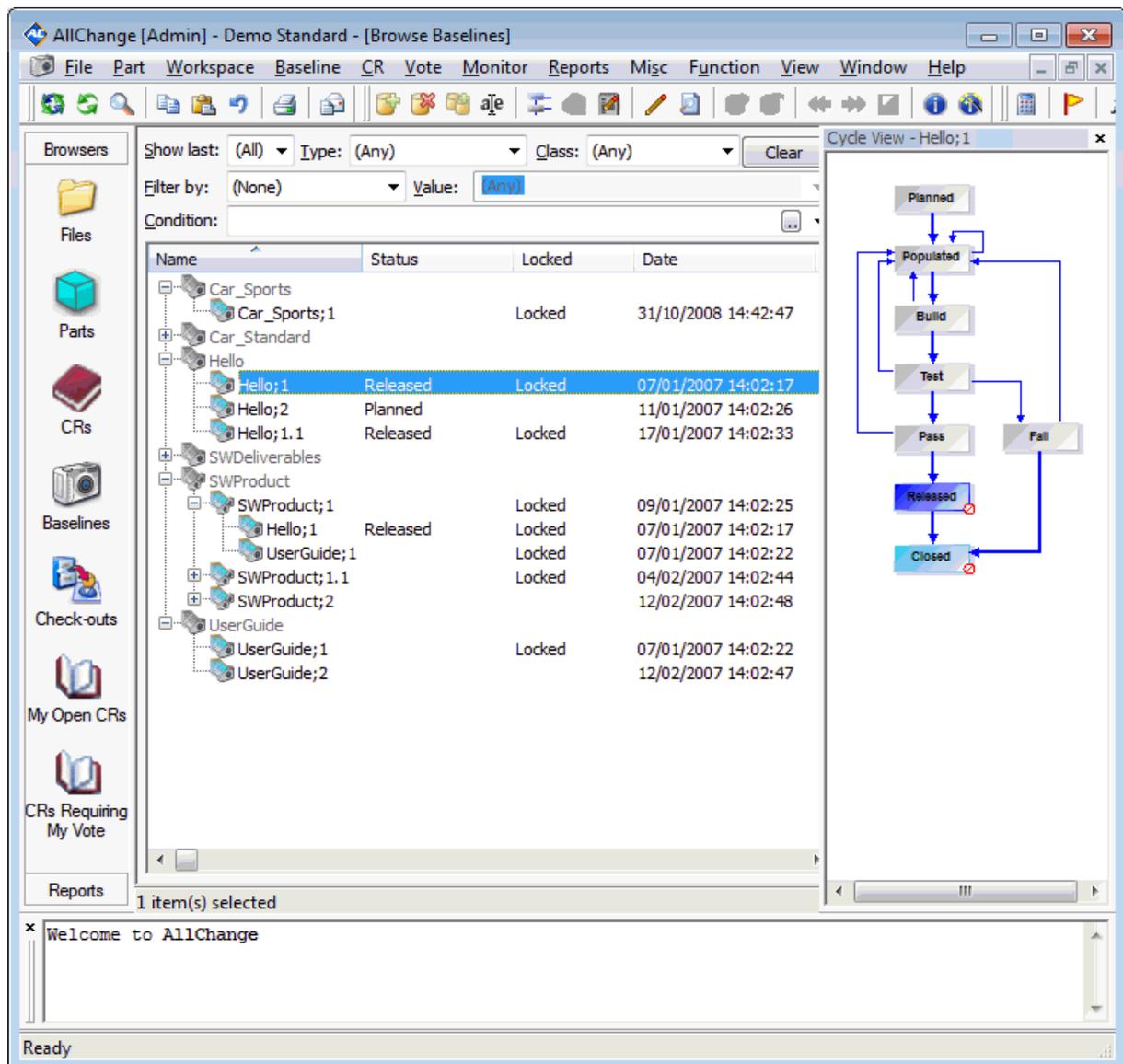
Once all build processes have been completed and the release is ready for test the status may be progressed to **Test**. The deliverables which have been built will then be put into the test pool specified. Also, all CRs included in baseline will also be progressed to **Test** to indicate to the testers that these are now ready for testing.

The **Test** status may not be exited unless all CRs associated with the baseline have been tested. When all the CRs have been tested then a decision should be made as to whether the baseline as a whole should pass or fail testing and the status should be set accordingly.

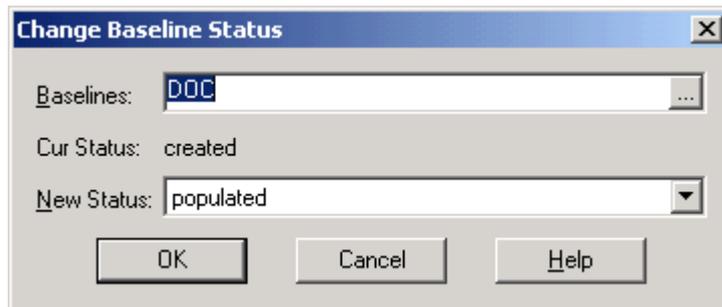
A baseline which **Passed** testing may be released. On release all deliverables will be copied from test to the specified **Release Directory**

To change the status of a Baseline use the [cycle-viewer](#). With a cycle displayed in the viewer double click on a status which is a valid progression and the status will be changed.

Note that if there is an open vote on the status, then double clicking will open the **Cast Vote** dialog instead of changing the status. If the vote permits status progression (e.g. an advisory vote) then shift double click will allow the status to be changed and the vote will be closed). This is indicated on the tool tip on the status.



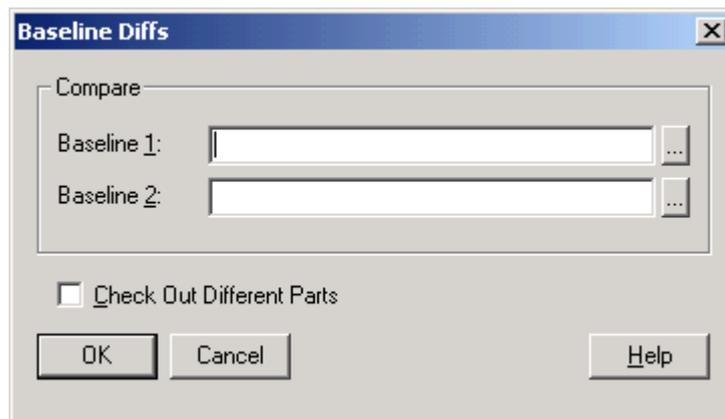
The **Change Baseline Status** dialog available from **Baseline | Status**, can also be used to change the status of a Baseline.



Any baselines selected will be copied as the **Bline**. The **New Status** combo box will show a list of the valid progressions from the current state: one of these should be selected.

Finding Out the Differences between Baselines

In order to find out what has changed in the baseline details between two baselines a **Baseline Diffs** function is supplied as a site modifiable function. The default configuration provides this function on the **Baseline** menu.



The default behaviour of this function is to take two selected baselines and report any parts added or removed and any version changes. The results of the comparison are displayed using the visual differencing tool. This will show a list of all the parts in both baselines highlighting those which are different between the two baselines.

The default differencing tool is **VisDiffs**. If Araxis Merge is specified as the preferred differencing application then interactive meta baseline comparison facilities are supported treating meta baselines in a similar fashion to folders allowing drill down into the constituent baselines. If using Araxis note that:

- Multiple occurrences of the same baseline in a meta-baseline are not supported
- Multiple occurrences of the same version of a part in the same baseline are not supported
- design baseline components, deleted components, etc. do not have a version time. A false time of 00:00:00 01/01/1990 is displayed
- As loading and comparison can take some time for large baselines we suggest that the recurs sub-folders is configured off in Merge if not necessary and re-test selected items as required
- File sizes are always shown as 0

If **Check Out Different Parts** is selected then any different parts will be checked out to the current workspace .

Checking Out a Baseline

About Checking Out a Baseline

A release or design baseline selected in a browser or viewer may be the subject of certain command dialogs which act on parts (e.g. **Check Out** and **Check In**): this will cause all parts in the baseline to be checked out/ checked in.

If a **design** baseline which contains only components is selected then the default version of each component will be acted upon (for **Check Out** and **Check In** commands).

If a meta-baseline is selected, then the baselines will be recursively descended and the resulting parts checked out.

Baseline names may be used in part paths to refer to whatever subsystems, components and/ or versions were recorded in a baseline so that a correct individual item or consistent group of items may be specified as an argument to a command in the future. [Specifying the Part in a Baseline](#) describes this facility in detail.

Sometimes it is useful to check whether the versions checked out to a workspace are the versions in a baseline, the [Check Baseline Check-outs](#) command may be used for this purpose, see Check Baseline Issues.

Incremental Releases

The parts which have changed between one baseline and another may be checked out to the current workspace using the **Baseline Diffs** function, see [Finding Out the Differences between Baselines](#).

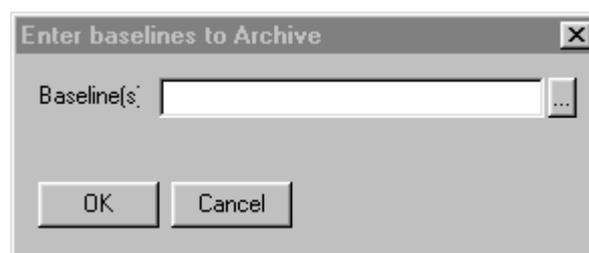
This may be useful when wishing to perform an *incremental* release rather than a full release from a baseline.

Archiving Baselines

The facilities to archive (i.e. remove from the current database and re-import at a later date) baselines is intended for removing old or obsolete baselines from the database for efficiency.

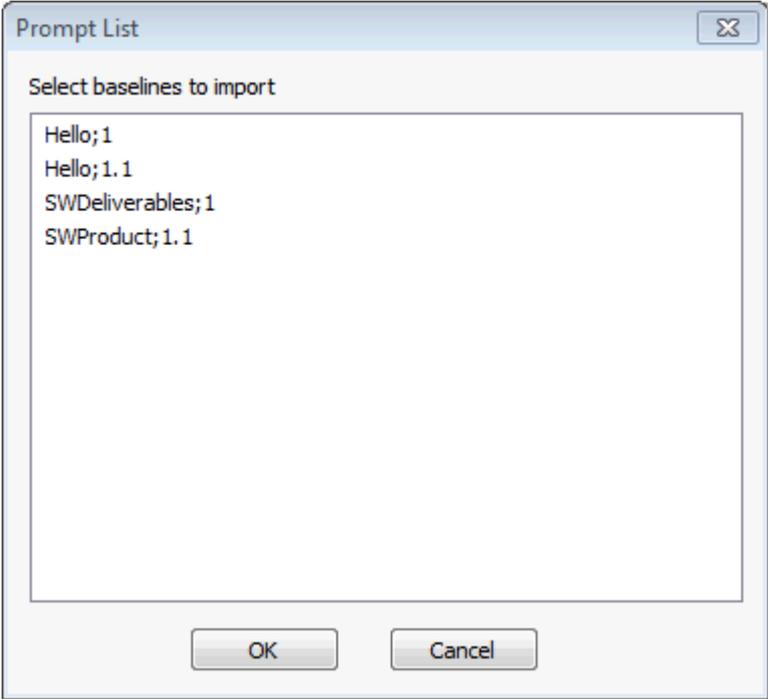
These facilities are available from the **Baseline | Archive Baselines** menu, but are only available to **All-Change** administrators. The Archiving facilities will also only be available if the Archiving feature has been enabled by the **AllChange** administrator, see the *AllChange Administrator Manual*.

To archive baselines use **Baseline | Archive Baselines | Archive**, this will export all information about the selected baselines to an external file in the project directory.



These baselines may then be deleted using **Baseline | Delete Baseline**.

If at a later date you wish to re-import archived baselines (e.g. to query them) use **Baseline | Archive Baselines | Import**.



Baselines are recreated with their original name and all their original information. 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.

Voting

About Voting

AllChange has a facility to allow decisions to be made based on a consensus opinion at defined points in a life-cycle, this is known as voting. Voting may thus be used to implement and record approvals. Votes may be defined for a particular status in the life-cycle of an item (the vote status), and in this status users may *cast* their vote for a particular progression in the life-cycle. The possible outcomes of a vote are the statuses to which the item can be progressed from the vote status.

Voting is supported for CRs, parts and baselines and each of these may be enabled/disabled by the AllChange administrator (see AllChange Administrator Manual). If voting has been disabled for all of these then the Vote menu will not be displayed.

Votes must be defined by the **AllChange** administrator (see the AllChange Administrator Manual).

For each vote defined the users who are permitted to vote are defined, the *voters*. Voters may be defined as *optional* in which case they may vote but are not required to do so for a decision to be reached. Note that in some cases optional voters may be required to vote in order to enable a decision to be reached.

There are various different types of vote (known as the **Decision Type**) which determines how a decision is reached.

The **Decision Types** are:

- **Advisory**: all progressions are valid regardless of votes cast. The vote remains open until the status is progressed
- **Specified number**: a specified number of votes for a progression are required for a decision to be deemed to have been reached. This number is defined by the **AllChange** administrator in the vote definition.
- **Largest number**: the progression with the largest number of votes is taken as the decision
- **More than half**: more than half of the votes cast must be for a single progression for a decision to be reached
- **Two-thirds**: at least two-third of the votes cast must be for a single progression for a decision to be reached
- **All but one**: all but one of the votes cast must be for a single progression for a decision to be reached
- **Unanimous**: all votes must be cast for a single progression for a decision to be reached

Votes may be parallel or serial:

Parallel:

All voters are invited to vote at the same time

Serial:

Voters are invited to vote in turn, with voters able to pass the vote to the next voter. Each voter in the series may be an individual or a group, role or users specified on a field. So for each phase of the serial vote there may be several voters voting in parallel.

When a status is entered which has a vote defined for it, the voting process begins for the item whose status has changed. The type of vote, the voters, the votes cast and other information can be seen on the **Cast Vote** dialog for any open vote.

The voters are sent an email to inform them that they need to cast a vote, and a record is added to the vote database for that item to indicate that a vote is opened.

If the vote is a serial vote, only the first voter(s) is(are) emailed.

The vote remains open until one of the following occurs:

- The status is changed
- A serial vote is closed by stop criteria

While a vote decision has not been reached, the status of the item may not be changed, but voters may cast a vote for one of the statuses which is a valid progression from the current status.

There are a few exceptions to this however:

- For an advisory vote, the status may be changed at any time providing the vote has no deadline defined, and there are no outstanding mandatory voters
- A vote may be ignored by a user with a role which has the **ignorevote** permission for the status

Details of votes cast may be found in the [Vote browser](#) and on the Votes tab of the items' viewers.

A decision is reached by one of the following events:

- When all mandatory voters have voted, plus sufficient optional voters to enable a decision to be reached.
- The deadline expires
- A serial vote's stop-criteria returns true (stop-criteria are a part of the vote definition)
- A vote is cast for the decision override status (the Decision override status is part of the vote definition)

When a decision is reached, an email is sent (if defined to do so in the vote definition) to the users defined to indicate either that a decision has been reached, or that a decision cannot be reached, i.e. the vote is blocked.

The decision is calculated according to the **Decision Type**.

The decision may be inconclusive, i.e. blocked, for each **Decision Type** as follows:

- **Specified number**: no status has the required number of votes
- **Largest number**: all statuses have an equal number of votes
- **More than half**: no status has more than half of the votes
- **Two-thirds**: no status has two-thirds of the votes
- **All but one**: no status has all but one of the votes
- **Unanimous**: the votes are not unanimous

An advisory vote always has a conclusive decision.

While a vote is open users may alter or rescind votes cast. This may change the decision reached, or change whether the vote is blocked. An email is sent, on a decision being reached, to the recipient as defined in the vote definition. Additional emails are sent if the decision changes due to votes being cast, altered or deleted.

For a serial vote once all mandatory voters have cast a vote within a set of voters, the vote must be passed on to the next set of voters. This is achieved using the **Next** button on the **Cast Vote** dialog. This will only be enabled if you have permission to pass the vote on (by default one of the current set of voters or the vote_manager) and if all mandatory voters in the current set of voters have voted. Once passed on to the next set of voters optional voters from previous sets may no longer cast a vote. When the vote is passed to the next set of voters an email is sent to inform them that they should cast their vote. If all possible voters for a stage in the series have voted the vote will be automatically passed to the next set of voters.

If a conclusive decision is reached, then the item may be progressed to the decision status (in the case of an advisory vote all statuses are permitted). This may be achieved by holding the SHIFT key while double clicking the status in the cycle viewer, or right click on the status in the cycle viewer and select **Move Status**.

If a vote is blocked then all statuses are permitted as with an advisory vote.

If the vote is defined with **Auto-progress** and a single status is the decision, then the status change is made (and the vote closed) as soon as a decision can be arrived at.

Information on open votes, blocked votes and votes awaiting voters may be reported on from **AllChange** using the XXvtopen, XXvtblocked and XXvtawaiting reports, where XX is either cr, bl or pa, for CRs, Baselines or Parts respectively.

Votes may also be [monitored](#) using the baseline/cr/part event and vote/altvote command .

Casting a Vote

Users may cast a vote either by email or within ACE:

Casting a vote within ACE

Users may cast a vote by means of the **Vote** dialog available from **Vote | Cast Vote** or context-menus for CR, Baseline and Part browsers and viewers. Votes may also be cast from the Cycle Viewer, by double-clicking on the current status or the status they wish to vote for.

Vote

Vote Details

ACMD: RFC00008
Status: AssessEval

Date	User	Vote	Representing	Comment
16/11/2009 15:44:37	Malindi Lamb	(Vote S...		Start vote added b...

Vote Definition

Valid Voters:

Voter
Malindi Lamb (Optional)
Paul Vickery (Optional)

Users yet to cast a vote:

Voter
Malindi Lamb (Optional)
Paul Vickery (Optional)

Deadline: (None) Remaining voters: 2
Decision Type: Advisory Decision Reached: Advisory: any status allowed

Vote Details

You have not yet cast your vote for this item.

Representing: Assessors

Vote: Authorise

Comment:

OK Cancel Help

The **Vote** dialog is opened where the user can view existing votes for the item, see which users have voted and are left to vote, see information on the vote definition, and also cast or change their vote.

The **Vote Definition** information includes:

- **Valid Voters:** A list of all the voters who may vote
- **Users yet to cast a vote:** A list of the users who have not yet voted. Optional voters are indicated. For a serial vote, voters not yet valid to vote are shown disabled/greyed.
- **Deadline:** If specified this is the date by which all votes must be cast. After this date the vote a decision deemed to have been reached and no further votes may be cast or modified.
- **Decision Type:** This is the type of decision defined for this vote.
- **Remaining Voters:** The number of voters still to vote
- **Decision Reached:** This shows what decision has been reached if any.

To cast a vote or modify an existing vote the following information must be entered in the **Vote Details** section of the dialog:

- **Representing:** this is the capacity with which the user is casting their vote. The drop down list will show valid entries according to the vote definition of which the user is a member. (Self) will be shown to represent a named individual voter voting as themselves.
- **Vote:** this is the vote that is to be cast.

For any **Decision Type** other than serial this may be any of the possible progressions from the current status. In addition the following may be specified:

(No Preference) - this counts as a vote cast but does not contribute to the decision that will be reached

(Not Voted) - this will remove a vote previously cast by the user

For serial votes the **Next** button may be used to pass the vote on to the next voter(s) in the series. The button will be disabled if you do not have permission to pass to the next voter or if there are still mandatory voters to vote in the current set of voters.

- **Comment:** this is optional and may be used by the voter to make a comment about their vote

Votes may have arbitrary fields defined, and if so, these may also require values to be entered in this dialog on the **Fields** tab.

Casting a vote by email

Users may cast a vote by email if this has been enabled and configured by the AllChange administrator, see the [AllChange administrator manual](#).

The simplest method of casting a vote by email is to reply to a vote initiation email sent by AllChange. This contains information as to how to vote by email. Simply reply to the mail box being used to submit votes by email, your **AllChange** administrator can tell you what this is. This may be the email address that the vote initiation email came from or may be another email address depending on how the feature has been configured.

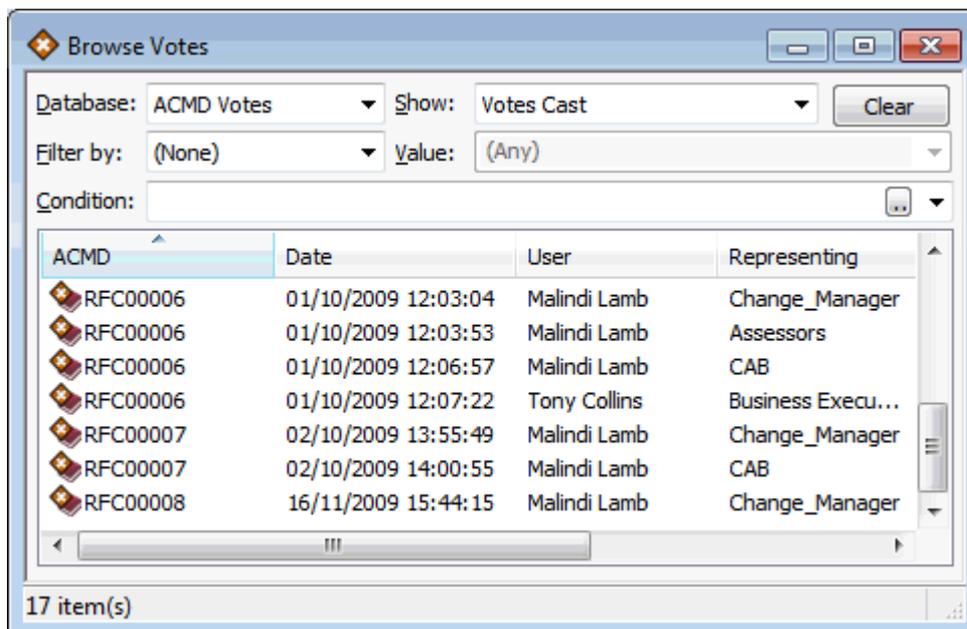
The email sent must have the vote cast a the first line of the email body, and must have the following information elsewhere in the email together with any mandatory fields (e.g. Comment or arbitrary fields) specified in the form *field=value (this is provided automatically in the vote initiation email)*:

```
RemoteCommand=Vote  
Database=database  
Item=item
```

The user to vote as and in what capacity they vote (e.g. role) is calculated from the sending email address. If the user has already voted then their vote is altered, otherwise a vote is cast.

Votes Browser

The Vote browser (available from **Vote | Browse Votes**) shows the votes that have been cast.



The precise information shown for each vote is user definable (see [Customising the User Interface](#)).

Database is used to select whether to browse CR, part or baseline votes.

Show may be used to filter the votes shown to:

- **All Votes** - this will show all entries in the votes database, this include Start and End vote records
- **Votes Cast** - this will show just the votes cast by users
- **Open Votes** - this will show all votes for votes which are still open (i.e. a decision has not been reached)

Filter By may also be used to restrict what is displayed according to the **Value**.

The browser supports the following:

- The information displayed may be modified (i.e. which columns) by right clicking on a column title and selecting **Add/ Remove Column**.
- The columns may be reordered by dragging them and dropping them in the required position.
- The information may be sorted by any column by selecting the column title.

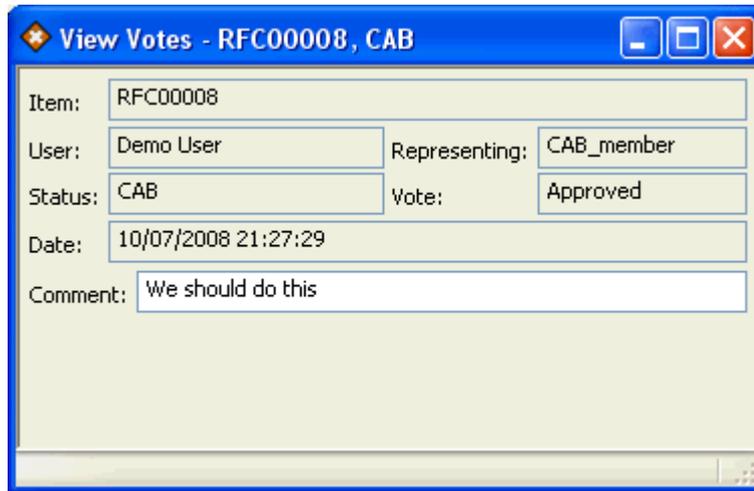
These features are common to all browser windows, see [Browsers](#) for a more detailed description.

Double clicking on a vote in the browser will show the details of the selected vote in [Vote viewer](#) window.

Vote Viewer

Vote details may be viewed in the vote viewer by double-clicking or selecting **Vote | View** on a vote in one of the vote browser lists.

This will show information about the vote and allows some of the information, e.g. the comment, to be altered.



The information shown is:

Item:

This is the item which has been voted on. This may be a CR, baseline or part

User:

The user who cast the vote

Representing:

The role or group that the user casting the vote is representing. This will be blank for Start/End votes

Status:

The status that the item was in when the vote was cast

Vote:

The vote that has been cast. This will normally be a status which is a valid progression from the vote status. As a special case this may be (Vote Start), (Vote End), the stop/block word defined for a serial vote, the next/pass word defined for a serial vote (see the **AllChange** Administrator Manual).

Date:

The date and time that the vote was cast.

Comment:

Any comment made when the vote was cast.

Arb1..Arb40:

Any arbitrary fields defined for the vote. These will be defined by the **AllChange** administrator, see the **AllChange** Administrator Manual

Monitors

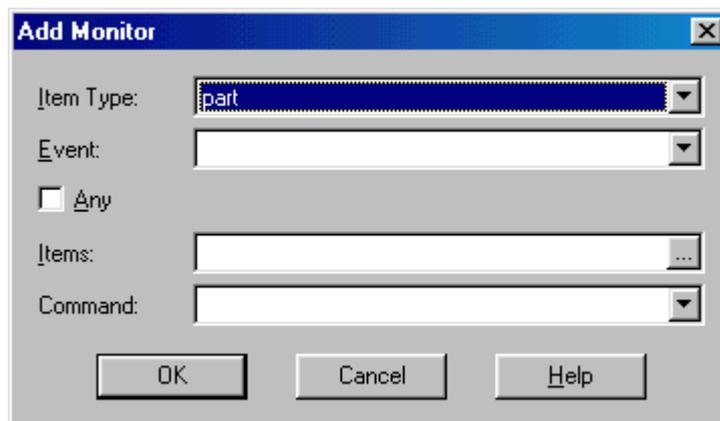
Monitors

AllChange has a facility for monitoring events affecting objects which are of interest.

Whenever a monitorable event occurs within **AllChange** the system looks to see whether any users have placed monitors for that event on the object being affected by the event. If there are any matching monitors in existence the placers of the monitors are informed that the event has occurred. The **AllChange** Administrator will have set up precisely what "informing" consists of, but it should mean that the operating system's "mail" facility is used to post an appropriate message to the users.

Monitors may be disabled by the **AllChange** administrator, if they are disabled then all monitor menu options will be disabled.

A monitor may be placed by using the **Monitor | Add Monitor** dialog.



The **Item Type** of the item to be monitored should be selected. This will affect both the events available for that type of item and the items to be monitored.

The **Events** that may be monitored are certain selected **AllChange** commands and some general events (part, cr, baseline). The general events will have no effect unless configured to do so by the **AllChange** administrator. If configured then the monitor arbitrary field will probably also have been configured to specify the specific event to be monitored.

[Figure 12.1](#) describes the events which may be monitored in the default supplied system. The **SubEvent** documents the supplied `arb1` field values that may be specified when placing a monitor on a general event. Site specific events may have been added.

Figure 12.1: Event Monitors

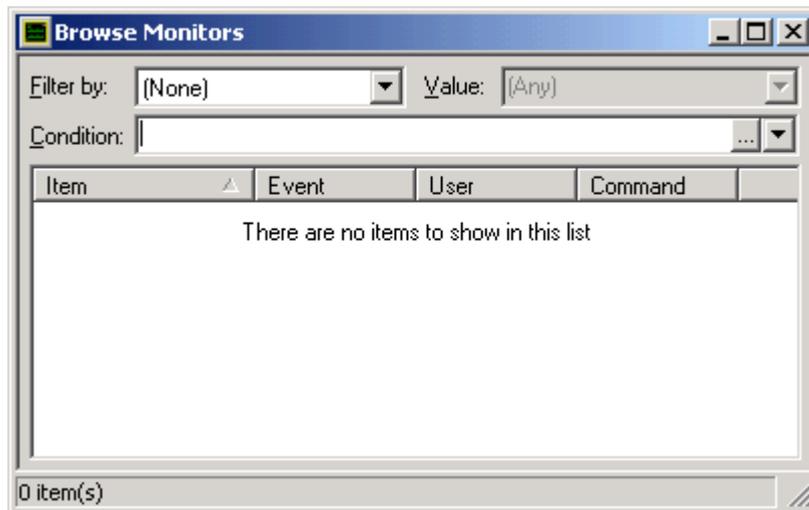
Item	Event	SubEvent	Description
part	part	checkout	Monitors for a part being checked out to a workspace
	part	checkin	Monitors for a part being checked in from a workspace
	newversion		Monitors for a new version of a component being created
	use		Monitors for a part being used
	status		Monitors for a status change on a part
	part	alterobsolete	Monitors for the obsolete flag on a part or version being changed
	part	vote	Monitors for a vote being cast

	part	altervote	Monitors for a vote being modified
	part	rename	Monitors for a part being renamed
file	promote		Monitors for a file being promoted to a pool
	release		Monitors for a file being released from a pool into a release directory
cr	statuscr		Monitors for the status of a CR changing
	cr	assigncr	Monitors for a CR to be assigned to a user
	cr	vote	Monitors for a vote being cast
	cr	altervote	Monitors for a vote being modified
baseline	statusbaseline		Monitors for the status of a baseline to change
	baseline	renamebaseline	Monitors for a baseline being renamed
	baseline	vote	Monitors for a vote being cast
	baseline	altervote	Monitors for a vote being modified

The items to be monitored should be specified. If the user is interested in the event occurring on *any* item of the specified type, the **Any** should be selected instead of specifying the items individually.

Users may set up any monitors that they require and the information about what events and objects are monitored and by whom is stored in the *monitors database*.

This may be browsed using **Monitor | Browse**.



Monitors may be deleted by selecting the monitors to be deleted in the browser list and then selecting the delete record toolbar button.

Reports and Queries

About Reports and Queries

Full facilities are provided within **AllChange** for informal querying and formal reporting from the **AllChange** database.

Many simple on-line queries may be able to be answered by the information provided in the various browsers and viewers which provide a fixed format view of most aspects of the **AllChange** database.

For example:

How do I know what is checked out to a particular workspace and/ or user?

The check-outs browser window can provide this information very simply. Alternatively, if the workspace you are interested in is the current workspace, then one of the **Check-out** filters on the part browser could be used to show those parts checked out to the current workspace.

How do I know what CRs are currently assigned to me?

The CR browser will provide this information with the use of the **Filter by** and selecting **Assignee** followed by a **Value** of *me*. Alternatively the **Condition** may be used but in this case that would be a less efficient method.

The contents of browsers may be printed as a fixed format reporting mechanism using **View** | [Print](#) or from context menus from the browsers.

Some information is only available as a report or is best accessed via a report. A wide variety of report types are supported catering for everything from summaries to metrics and graphical output, see [Types of Report](#) for details.

AllChange supports two methods for accessing the **AllChange** reporting facilities:

1. From the report wizards/dialogs available from ACE.
2. From a wizard accessed from within MS Word

There is a [Report Wizard](#) available from each database menu. This will guide you through the process of generating a report from within ACE.

When you become more familiar with reporting the **Reports** | **Report** menu provides access to the underlying reporting and querying facilities available within ACE, options available are:

[Report](#) <database>: provides facilities to generate reports in any site specified format with any query required. A number of sample report formats are supplied with the system.

Two types of report are available, textual reports and acreports. Textual reports will appear in the output window in a fixed width font format or may be sent to a text file or printed.

acreports are produced using the GUI reporting tool ACREPORT, results are formatted more prettily than the textual reports and may be printed.

[Report Wizards](#) Provides access to various report wizards which guide you through the process of generating a report

[Column Report Wizard](#): allows tabular report formats to be generated for output to Word or Excel

Report for each database is also available from the corresponding database menu.

For details of creating your own reports see:

- ACREPORT which is available from the **Reports** menu. This allows GUI based report format generation.

- **AllChange** Administrator Manual for textual reports: it is only intended that AllChange Administrators should need to use this facility

Types of Report

AllChange's flexible reporting facilities allow for several different types of report to be produced.

Brief

A textual summary of a database contents usually in a tabular form. The example shown below is produced using ACREPORT.

Number	Status	Priority	Target	Assigne	Summary
CR00001	released	Medium	SRC;1	demo	Initial import of source
CR00004	complete	High	SRC;2	demo	Problem that must be solved
CR00005	complete	Medium	Unallocated	demo	Hardware elements fault when temperature low
CR00006	complete	High	SRC;2	demo	Add AllChange Keywords to Source Files
CR00007	released	Medium	SRC;1.1	demo	Bug in product software
CR00008	complete	High	SRC;2	demo	Test for Year 2000 Compliance
CR00009	complete	Medium	SRC;2	demo	Edit required on module1.c to remove local print error

Detail

A textual report giving a lot of detail about each item reported on. The example shown below is produced using ACREPORT.

Full Change Request Report

CR Number: CR00000

Summary: Edit required on module1.c to remove local print error

Originated by: demo **Assigned to:** demo

Current Status: complete **Class:** cr

Reference:

Type: Error

Priority: Medium

Complexity: Low

Keywords:

Introduced:

Discovered:

Date Due:

Target Release: SRC2

Project:

Locked:

Top Part: /productsw/source

Parts Affected: /productsw/source/module1.c

Versions Solved: /productsw/source/module1.c;003

Blines Affected:

Blines Solved:

Files Affected:

Status log:

Status	Date	User	Assignee
submitted	06/02/00 12:41:44	demo	
accepted	08/02/00 12:41:44	demo	
Assigned	08/02/00 12:41:45	demo	demo
fixing	09/02/00 12:41:45	demo	
complete	10/02/00 12:41:45	demo	

Text:

Description/Reason For Change::
 A line regarding a local print function has been incorrectly added to module1.c

Operational Impact::
 Extra blines caused by this module could bring down the WAAH. Consequential losses to the organization potential reaches £2 Billion

Identify any Risk Areas::

Generated on 2002/11/14 02:06:06 by demo 1

Textual Metrics

Metrics may be produced and presented in a textual form. These may be used, for example, to analyse quality metrics and improve quality processes. The example shown below is produced using the text report facilities.

Overview of CRs for Project: AC6.0

Total	12		
	Current State	Today	This Month
Raised	2	0	1
In progress	7	0	4
Closed	3	0	0

Word Table

Summary data may be produced and output directly to Word as a table. The example shown below is a summary of the CRs.

<i>Number</i>	<i>Class</i>	<i>Status</i>	<i>Assignee</i>	<i>Summary</i>
CR00001	cr	released		this is an example CR
CR00002	cr	fixing	demo	this is another example CR
CR00003	cr	rejected	devgroup	User has a problem with the functionality
CR00004	cr	released	demo	Problem that must be solved
CR00005	cr	pass	demo	Upgrade PC1 Memory
CR00006	cr	released	demo	Add AllChange Keywords to Source Files
CR00007	cr	released	demo	New version of supplied stuff to implement new features
CR00008	cr	systest	demo	Test supplied progs in environment 1 for YR 2000 compliance
CR00009	cr	test	demo	Need a new global variable

Excel Table

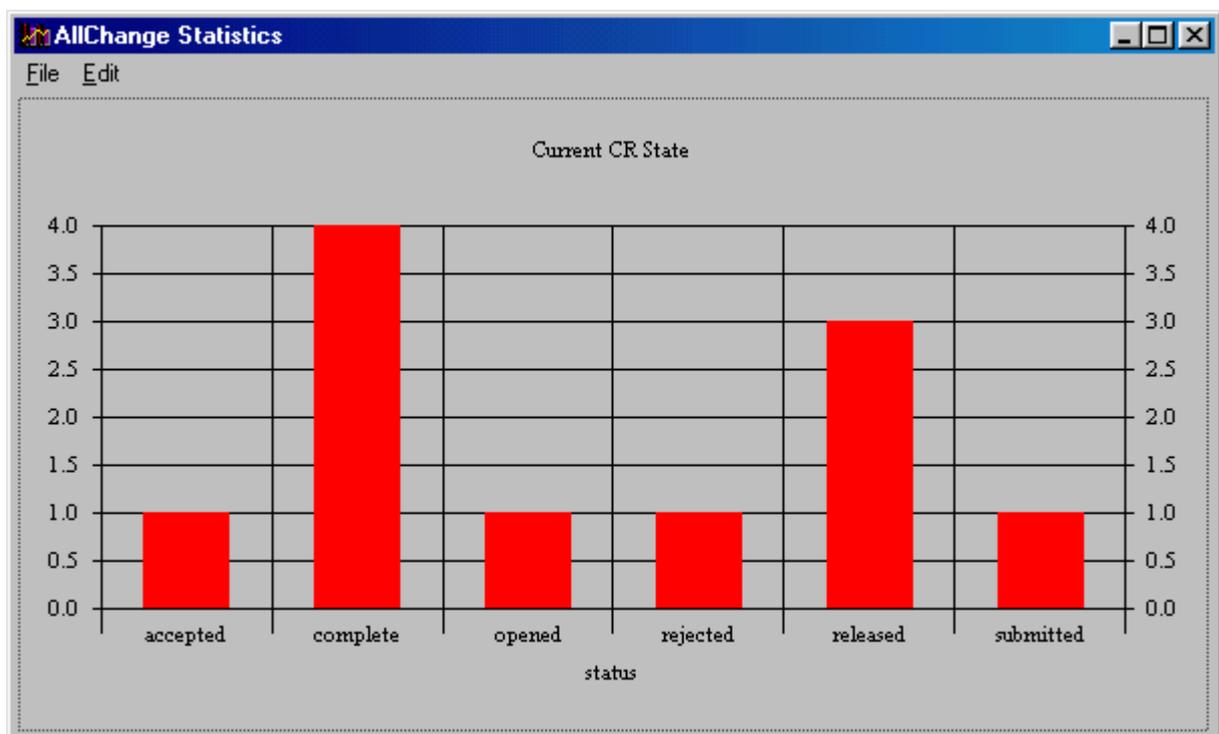
Summary data may be produced and output directly to an Excel spreadsheet.

	A	B	C	D	E	F	G	H
1	Name	Class	Status	User	Comment			
2	DOOCR1	release	FitForUse	demo	Documentation Baseline for Release 1			
3	HWR1	baseline	FitForUse	demo	Baseline for Hardware Release 1			
4	Release1	baseline	FitForUse	demo	Baseline for Release 1			
5	Release2	baseline	created	demo	Product Release2			
6	SRCR1	baseline	FitForUse	demo	Baseline for source development			
7	SRCR1_1	baseline	NotFitForUse	demo	Bug Fix to Source Release1			
8	SRCR2	baseline	created	demo	Baseline for source development Release 2 -			
9	Salamander	baseline	created	demo	3rd Party application			
10	Tutorial_Baseline			demo				
11								
12								

Note that **AllChange** replaces the contents of an open workbook, if there is one, in an already running Excel. You should therefore make sure that any changes you have made in Excel are saved before running an **AllChange** report which produces output in Excel.

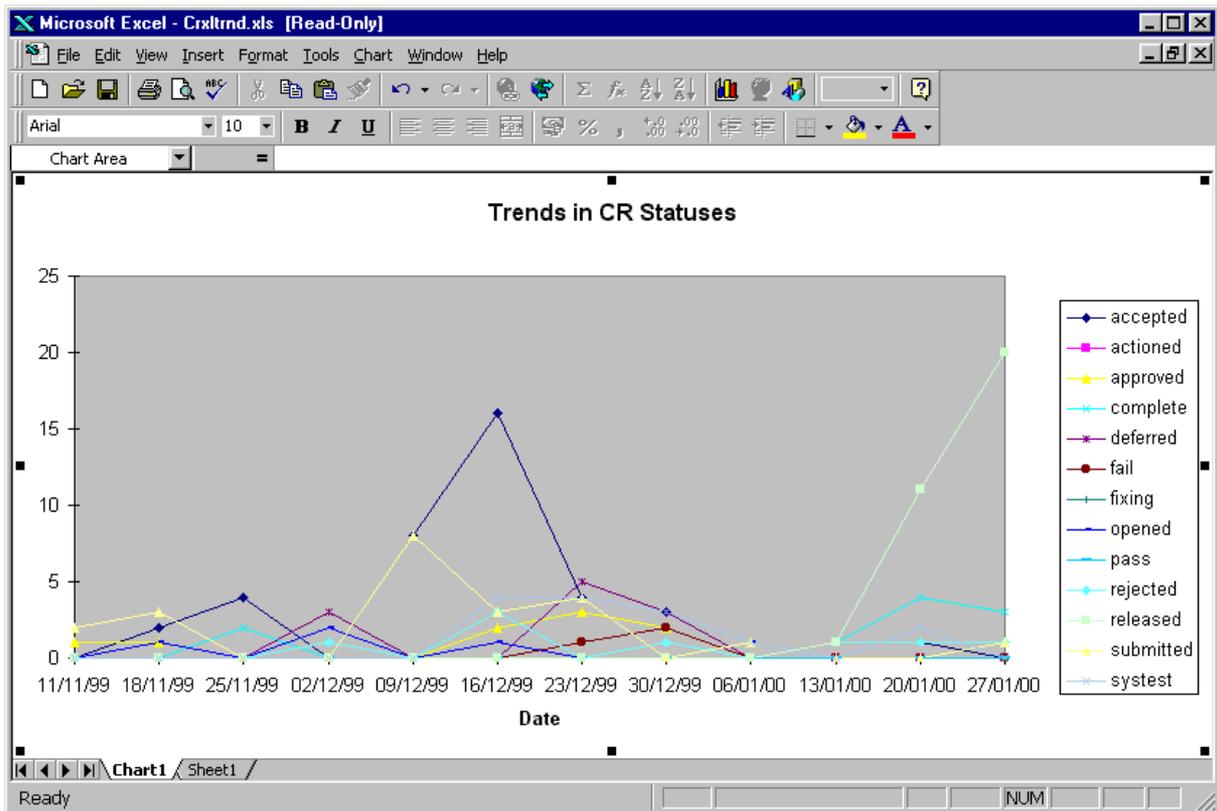
Graphical Statistics

A graph drawing application is supplied with **AllChange** together with some sample reports which use it to display metrics related to CRs in a graphical form. The example below shows the numbers of CRs in each of the different possible statuses.



Excel Graph

Metrics may be produced and output directly to an Excel spreadsheet and presented as a graph.



(See also note above)

Word Wizard

Reports may be generated directly in Word by use of the Word Wizard.

AllChange Report Produced on Tuesday, January 11, 2000

CR Number: CR00003

Summary: User has a problem with the functionality

Originator:	demo	Assigned To:	devgroup	Reference:	myref3
Curr. Status:	rejected	Class:	cr	Type:	Off Spec
Priority:	High	Complexity:	High	Keywords:	Functionality
Introduced:	Build	Discovered:	Test	Date Due:	
Target Release:		Project:		Arb10:	

Description::
 The functionality does not meet expectations

Action::
 Implement required functionality

Status Log	Date	User	Assignee
submitted	06/06/95	demo	
rejected	06/06/95	demo	
Assigned	23/11/99	DEMO	devgroup

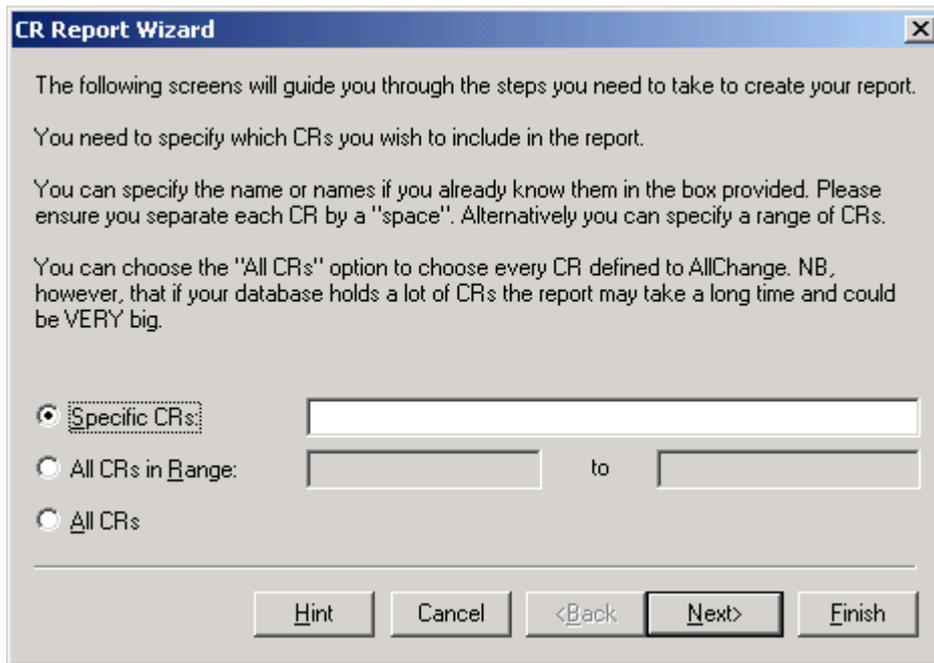
All of the above report types may be generated by the methods described in the following sections.

Report Wizard

There is a **Report Wizard** option available on each of the **Part**, **Baseline** and **CR** menus. This will guide you through the process of generating a report from within ACE on each of the corresponding databases.

The first wizard screen will allow you to specify the items that you wish to report on. Any items selected in a relevant browser or viewer may be used.

The example below shows the first wizard screen for reporting on CRs.

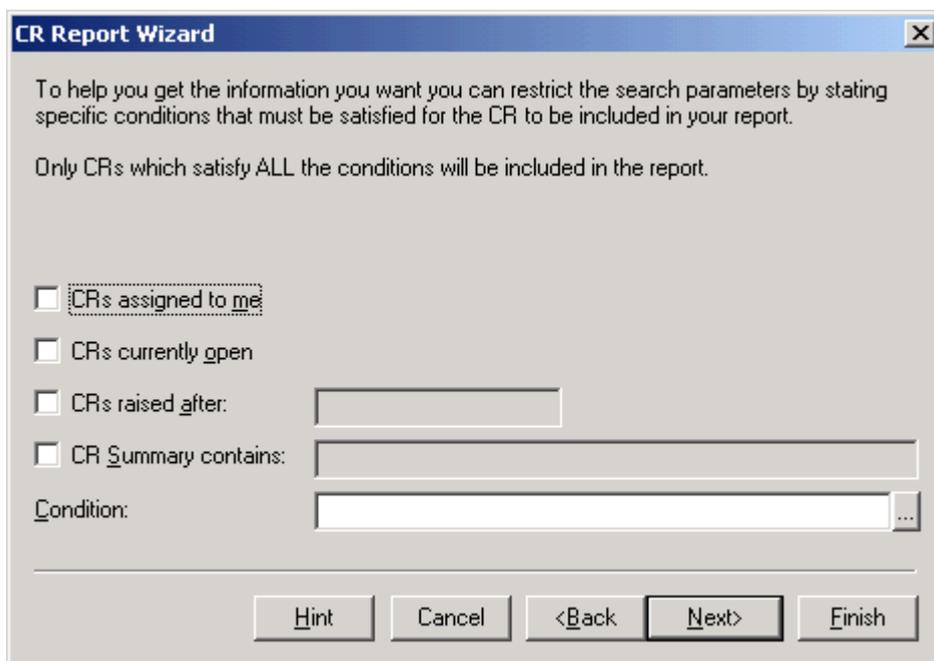


Any CRs selected in a browser or viewer will be copied as the **Specific CRs**, but these will only be the CRs reported on if the **Specific CRs** button option selected. For CRs a range may be specified, or all CRs may be reported on.

Select **Next** to progress to the next screen.

The second wizard screen allows you to express conditions which the items to be included in the report must satisfy. Certain common conditions are shown which may be selected, any additional criteria may be specified by using the ACCELcondition editor accessible from the **Cond** button.

The example below shows the second wizard screen for reporting on CRs.



Select the **Next** button to progress to the next screen.

The third wizard screen allows the index that the report is to be generated against to be selected.

Both the index to be used and a specific value may be specified. The example below shows the index wizard screen for reporting on CRs.

The **(Default)** entry indicates the default index for the database.

The values that may be selected for a specified index will include the special values:

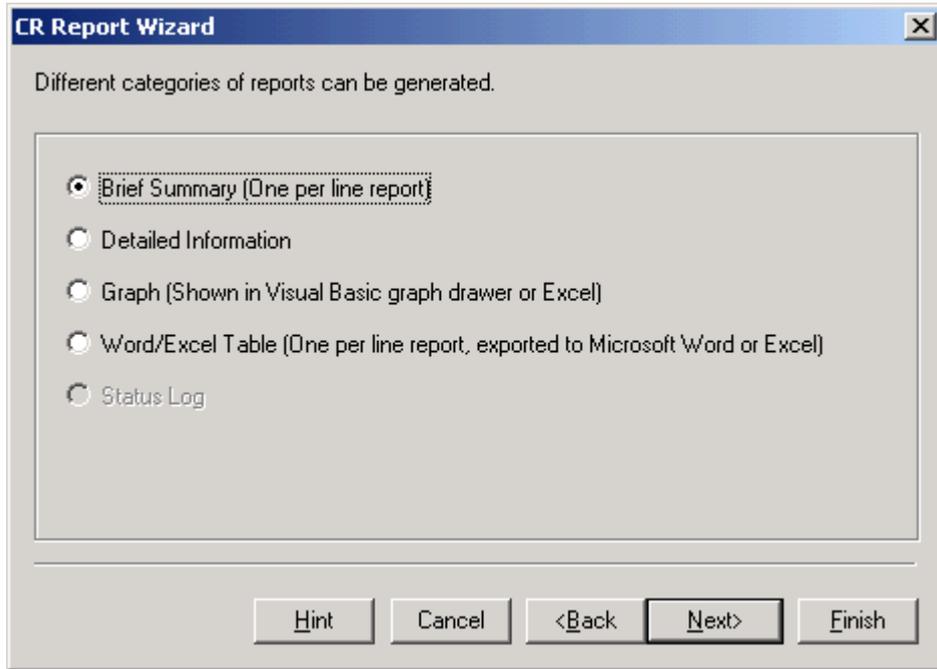
- **(Any)** to indicate all values for the index are required
- **(Empty)** to indicate values where the field indexed has no value

Selecting an index provides a means of sorting the report output by the indexed item and a fast search mechanism if a **Value** is given.

Select the **Next** button to progress to the next screen.

AllChange reporting facilities allow for several different types of report to be generated. This screen allows you to select the category of report that you require.

The example below shows the category wizard screen for reporting on CRs.

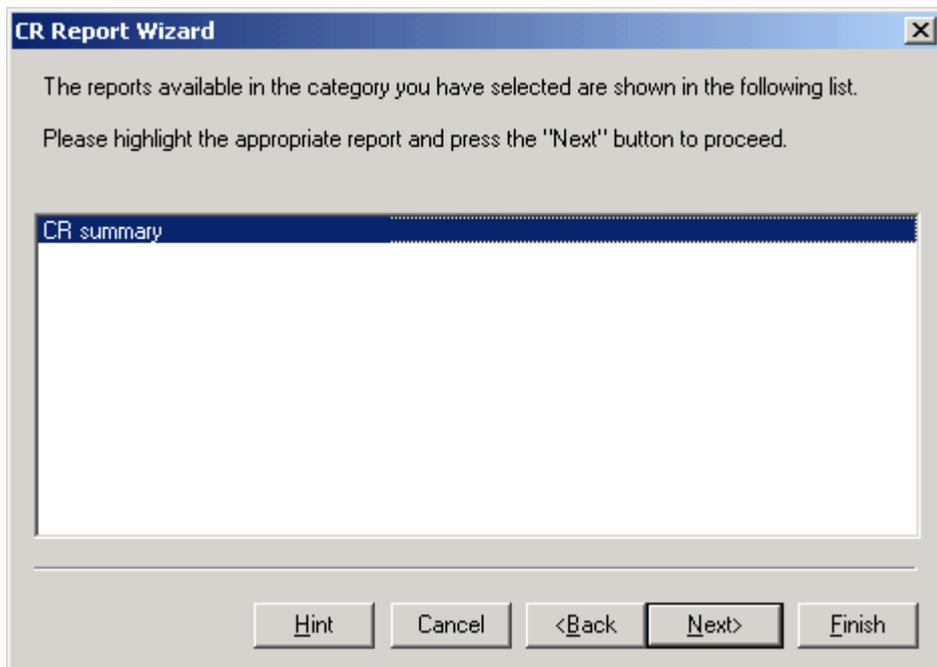


The selection that you make here will determine the contents of the next wizard screen.

Select the **Next** button to progress to the next screen.

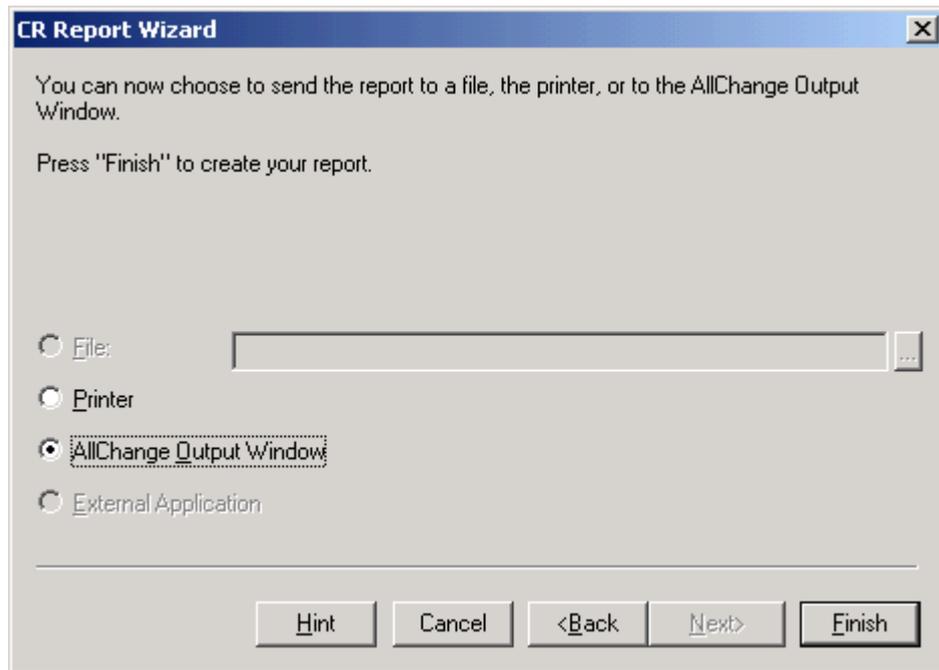
This screen provides you with a list of the reports available in the category selected in the previous screen.

The example below shows the list of reports available for the **graph** category for reporting on CRs.



Select the report required and select the **Next** button to progress to the next screen.

The final screen allows you to specify where you wish the report output to be sent. The options available will depend on the type of report being generated.



At any stage in the wizard process you may go back to a previous screen, when you are happy with all your selections select **Finish** and your report will be executed.

Whilst the report is being generated a dialog will appear allowing the report to be cancelled . This is useful if you wish to abandon a report which may be taking longer than expected, or if you entered an incorrect condition.

Generating AllChange Reports from MS Word

Certain **AllChange** reports may be generated from a Word Wizard which is supplied with the **AllChange** You should run a workstation install selecting the Word integration option in order to enable this facility.

In Word if you select **New** from the **File** menu, the **AllChange Wizard** (`allchang.wiz`) will appear in the list of templates offered.

Select this and the Wizard will start. If **AllChange** is already running then this is the **AllChange** that the wizard will communicate with, otherwise it will start a new ACE.

Select the report that you require from the list and specify the item name(s) for the items you wish to include in the report (if none are specified then all items for the database reported on will be included).

An example is shown below of the result of running the **Full CR Details** report from the Word Wizard.

AllChange Report Produced on Tuesday, January 11, 2000

CR Number: CR00003

Summary: User has a problem with the functionality

Originator:	demo	Assigned To:	devgroup	Reference:	myref3
Curr. Status:	rejected	Class:	cr	Type:	Off Spec
Priority:	High	Complexity:	High	Keywords:	Functionality
Introduced:	Build	Discovered:	Test	Date Due:	
Target Release:		Project:		Arb10:	

Description:
 The functionality does not meet expectations

Action:
 Implement required functionality

Status Log	Date	User	Assignee
submitted	06/06/95	demo	
rejected	06/06/95	demo	
Assigned	23/11/99	DEMO	devgroup

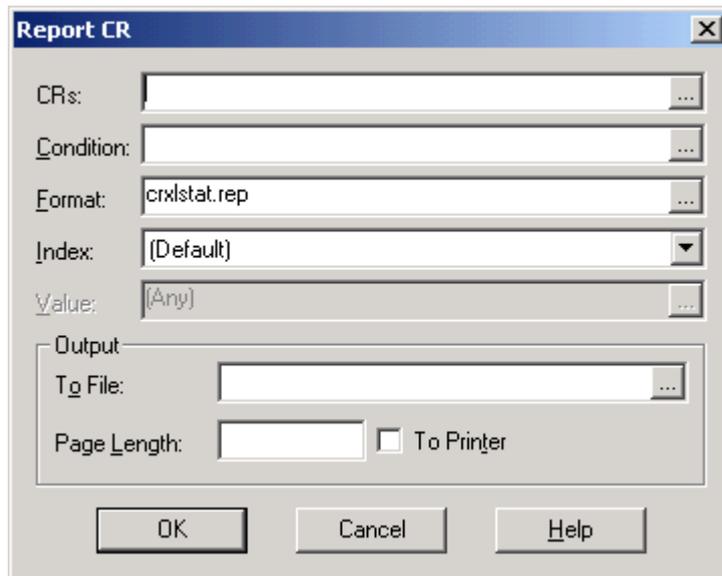
Additional reports may be added to the list as required for your project, see the *AllChange* Administrator Manual for details of how to do this.

Report

About Report

In order to report on a particular database select **Reports | Report | Report database** or **Report** from the appropriate database menu. Alternatively select the toolbar icon or use the context menus when an appropriate database window is the current window and the **Report** dialog relevant to the current window will be shown.

The report dialog for the CR database is shown below:



The general format of the report dialog for each database is common, certain options however are not available for all databases. The common options are available on the CR report dialog shown above and are described below. The additional options specific to each database are described in the appropriate subsection below.

Whilst a report is being generated a dialog will appear allowing the report to be cancelled. This is useful if you wish to abandon a report which may be taking longer than expected, or if you entered an incorrect condition.

Common Options

Items

The items that are to be candidates for inclusion in the report should be specified as the **CRs**, **Parts**, **Blines** etc. If no items are specified (or the word any) then all items in the database will be candidates for inclusion in the report.

Certain databases permit a range of items to be specified, see the section **Limiting the Items Reported On**.

Cond

Each item which is a candidate for inclusion in the report is submitted to any specified **Condition**. The condition must be satisfied for the item to be included in the report. The condition may be any valid ACCEL expression. The ... button provides access to the [ACCEL condition editor](#) which provides a user friendly interface to the a subset of the full language facilities

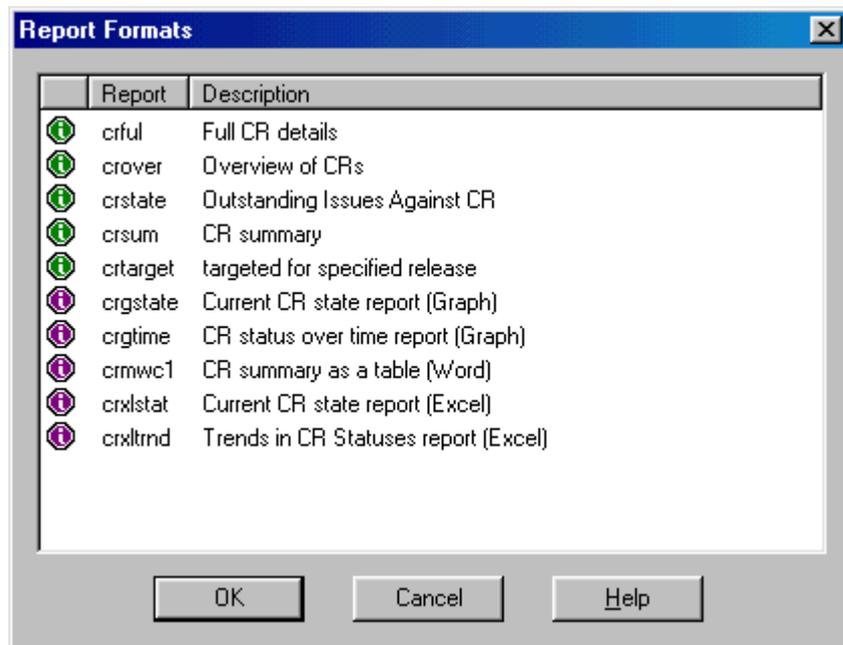
The condition is parsed and, if possible, corresponding clause(s) injected into the query sent to SQL. Where this happens the search can be fast.

- The parser can only handle a limited subset of ACCEL. The simpler the condition, the more likely it is to be included in the SQL query, e.g. a single simple <field><-operator><value> is likely to succeed.
- The parser is designed to handle the typical ACCEL generated by the ACCEL condition editor. It knows about the code generated for the various operators offered there, and common in-built ACCEL functions (e.g. `match_wild()`) and user-defined functions (e.g. `call(IsArbFieldDefined, ...)`).

- The parser can handle an ACCEL **and** operator in a condition, but only some **or** and **not** operators.

Format

The name of the file containing the desired specification (both content and layout) of the report *must* be given in the **Format** option. The format file may be selected from a list:



The names of appropriate format files for the database to be reported on is shown, together with a summary description of what that report will show. A large number of sample reports are supplied with the system and your Administrator may well have added some site-specific ones.

Reports may be ACREPORTS (acr's) or may be textual (rep's) which may produce text based report or may be passed on to other applications such as graph drawing applications or MS Word. The Windows version of the software provides a simple graph drawing package which may be used with reports.

The format will define the content and layout of a report. Furthermore, it may also define other report options and possibly even **Conditions** for the report. Any of these options selected in the **Report** dialog will *override* any values specified in the format file.

For details on creating your own report format files 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 **Condition** and the default index.

The **(Default)** index is the primary index 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

The results of the report may be:

- placed in a file if an **Output File** is specified. This is only valid for textual reports
- sent to the printer if the **Printer** option is selected
- displayed on screen if neither of the other options is used

The **Page Length** should be used to specify the number of lines required per page. If no value is given then the page length is taken to be unlimited. See the section on page breaks later on. **Page length** is only valid for text based reports.

Limiting the Items Reported On

The search for candidates to be included in a report can be limited to a range of items when using the primary index for the check-out, monitor, CR or baseline databases: this is achieved by specifying *item-->* *item* (two hyphens) as an item (there must be no space on either side of the -->).

For example, if

```
CR01000-->CR01500
```

is entered as the **CRs** for a CR report then only CRs CR01000 to CR01500 (inclusive) will be included as candidates for the report.

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.:

```
-->CR00020
```

will include CRs from the start to CR number CR00020.

Alternatively

```
CR10000-->
```

will include CRs from CR number CR10000 to the last CR.

Where a database holds a large number of items this facility will (drastically) reduce the search time compared to searching the whole database.

Page Breaks

Report automatically inserts a header at the start, and a footer at the end, of each page of a report. A new page is thrown whenever:

- the page length is exhausted; or
- a page break instruction is encountered in the *formatfile*.

A Page Length may be specified for textual report formats (*.rep*). If the page length is not specified then a page will be thrown only when a *-Pagethrow* instruction is encountered (i.e. the page length is unlimited). The page length may be specified in the report format file or on the command line.

Reporting on Parts

The candidates for inclusion in a report on parts may be modified by the following options:

Versions:

Allows you to select which (if any) versions of parts are to be candidates.

None no versions are included.

All all versions (except those which have the **No_Version** flag set) are included.

Default the default version is included.

Registered the registered version is included.

Top the top version is included.

Edit all versions including those with the **No_Version** flag set are included.

Follow Uses

If selected then the report will follow any uses type parts encountered to the part used when searching for candidates.

Recursive

If selected then all children of each specified part are included. This **must** be selected if any **Versions** have been selected for inclusion.

These options may be selected in the format file used; any option selected in the **Report Part** dialog will override any format file settings.

Reporting on Instances

Individual instances may be specified, and will be reported on.

If a version is specified then all instances of the version are reported on.

If a subsystem is specified then all instances of all versions of all components in the subsystem are reported on

Reporting on Baselines

A range of baseline names may be specified as the baselines to be included in the report when using the primary index; e.g. `BL1-->BL5` would report on all baselines with names alphabetically between `BL1` and `BL5`. Note that if a range is specified then the **Index** may not be used.

Baselines may also be limited by specifying the number of versions of the baselines to be included in the report using the **Show last** option. This will only include the latest *how-many* versions of the baselines, where **how-many** is the number specified. Note that **Show last** may not be used in combination with **Index**.

The **Recursive** option will cause a report on a meta-baseline to recursively report on all the sub-baselines contained within the meta-baseline

Reporting on Check-outs

If any partnames are specified as the **Parts** then all check-outs of those parts will be candidates for inclusion in the report (i.e. if a part is checked out to several different workspaces then details of all these check-outs will be candidates).

A range of partnames may also be specified when using the primary index.

If the **Recursive** option is specified then any children of the specified **Parts** which are checked out will be included as candidates for the report. This provides a useful mechanism for finding out what parts within a particular subsystem are checked out.

Reporting on Change Requests

The numbers of those change requests of interest may be specified as the **CRs** (full CR numbers must be specified, e.g. `00017` may not be abbreviated to `17`). If no change request numbers are given then all change requests will be included.

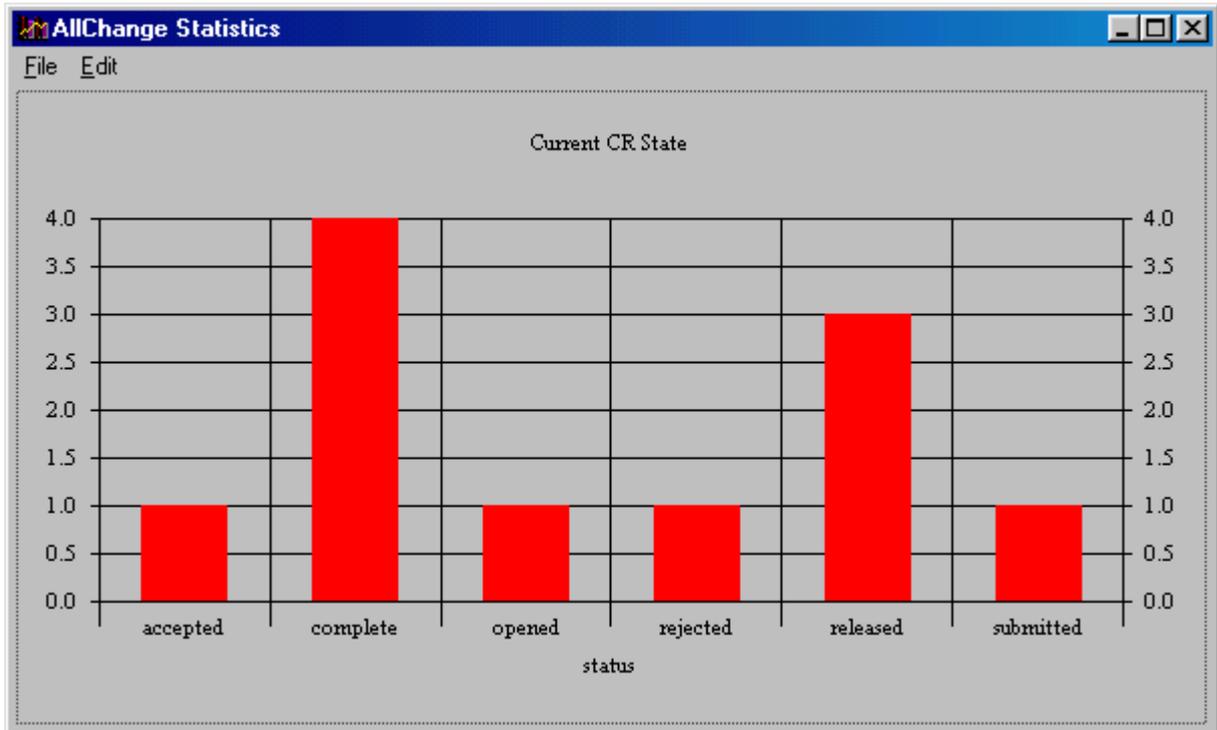
A range of CR numbers may be specified (e.g. `00100-->00999` will include only CRs with numbers 00100 to 00999). Note that if a range is specified then the **Index** may not be used.

Under Windows two graphical report formats are supplied for CRs:

crgstate

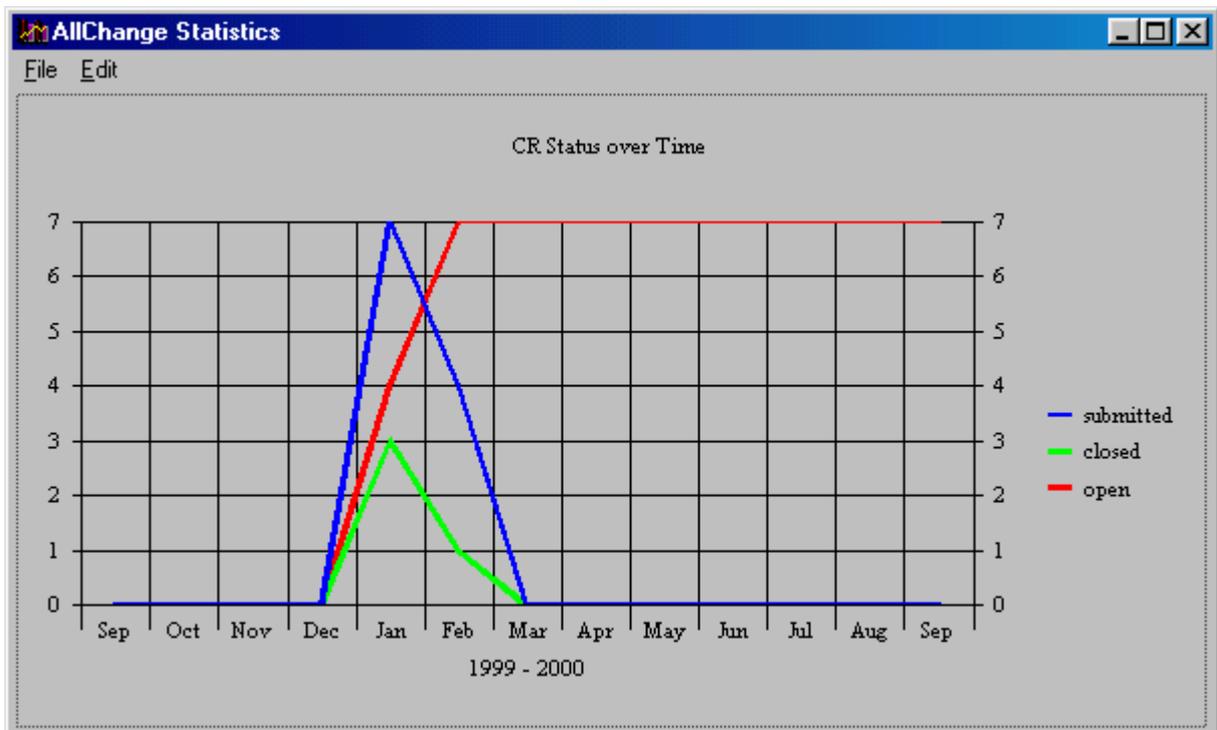
This displays a graph showing the numbers of CRs which are in a particular state. The user is prompted for what should appear along the X-axis (i.e. what state is of interest). In the supplied format, assignee, class, status and arb1 may be selected; however, this may be tailored to offer other options. The number of CRs in each possible value for the selected state will be plotted. For example, if assignee is selected then the number of CRs assigned to each individual who has at least one CR assigned, will be shown.

Another example is shown below where the number of CRs in each life-cycle status is shown.



crgtime

This displays a graph showing the number of CRs *Submitted* and *Closed* each month over the last 12 months. It also shows the number of CRs that were *open* for each month.



In addition, under Windows, the following sample report formats are supplied which export data to an Excel spreadsheet.

crxlstat

This is the equivalent of `crgstate` but exporting the data to Excel.

crxltime

This is the equivalent of `crgtime` but exporting the data to Excel.

crxltrnd

This exports data to Excel to display trends in CR statuses showing the number of CRs in each status at the end of each week.

crxlactv

This exports data to Excel to display CR activity showing the number of CRs which passed through each status during each week.

crxlTimeForClearance

This exports data to Excel to show the average time to close CRs per month and the number of CRs closed per month

Reporting on Votes

The **Items** reported on should be CRs, baselines or parts as required. If none is specified then all votes for that item type will be reported on.

For each of CR, baseline and part the following reports are supplied:

`xxvtawaiting`

Shows all items awaiting a vote

`xxvtopen`

Shows all items with open votes

`xxvtcurrent`

Shows all current votes and their decision

`xxvtblocked`

Shows all blocked votes for items

Where `xx` is `cr`, `bl` or `pa` for CRs, baselines and parts respectively

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 General Information

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. Four useful reports are included with **AllChange**:

gistats

Shows general database statistics such as the number of records in each section of the **AllChange** database as well as various environment information.

giconfig

Shows information about the current **AllChange** project configuration.

giuser

Shows information about registered AllChange users including their roles, workspaces and groups.

gibt

Pretty prints a BT file.

These reports are available from **Reports | Report | General Info** report dialog.

No items should be specified for `giconfig`, `gienv` or `gistats`. The full path to a file whose BT is to be reported on should be specified for `gibt`

Generating New Reports

Generating New Reports

New reports are created by creating a new *format file* and adding this to the list of available reports.

New reports may be saved in your home directory in which case they will only be available to you, or in the **AllChange** project directory where it will be available to all users. Only **AllChange** administrators may save reports to the project directory.

As an **AllChange** administrator, it may be useful to save reports initially to your home directory whilst you refine them and then later promote to the project directory to make generally available: this may be done using [Promote Report](#).

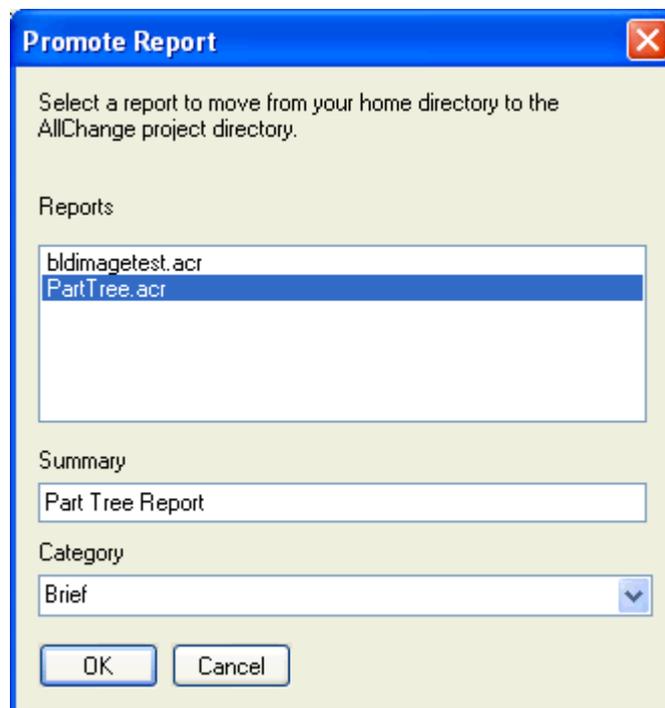
If you wish to generate a summary tabular report for Word or Excel, this may be achieved using the [Column Report Wizard](#).

Standard reports for viewing the result in **AllChange** or printing may be created and modified using ACREPORT. This provides a GUI based method of creating report layouts and content including the ability to populate tabular report layouts from the information selected to be shown in the ACE browsers.

Textual report formats including those which export data to third party applications require editing the format file using a text editor. Full details about writing these format files may be found in the **AllChange** Administrator Manual.

Promote Report

Promote Report allows report format files to be promoted from a users home directory to the AllChange project directory to make them available for general use. Promote Report is available from the **Report | Report** menu.



Reports shows a list of all **AllChange** report format files found in the home directory.

Select the report you wish to promote, if it is defined with a **Summary** and **Category** in your home directory then these controls will be completed with this information.

The **Summary** should provide a brief description of the purpose of the report

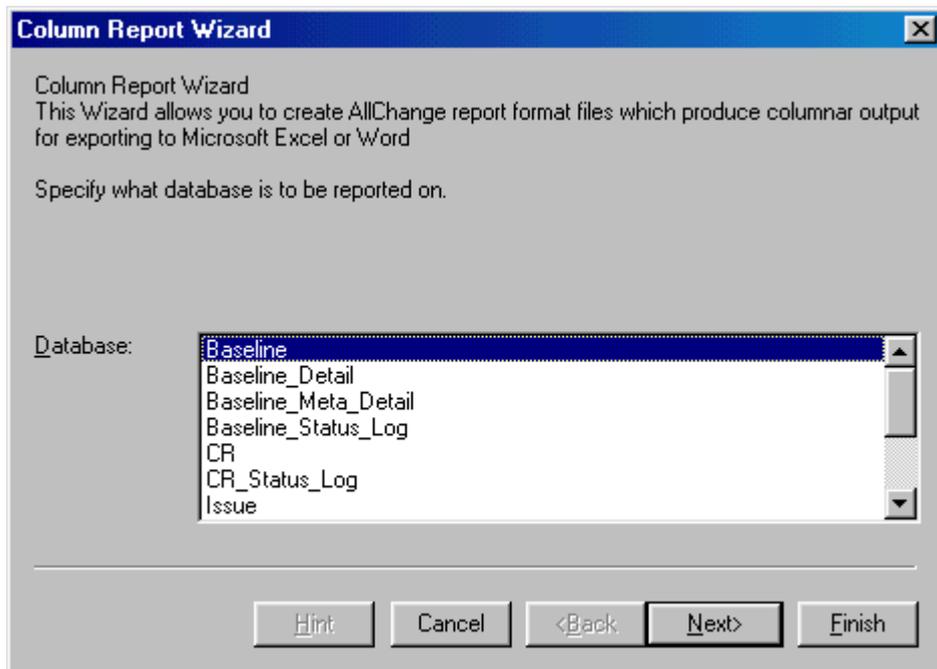
The **Category** should specify the type of report that it is.

On selecting **OK** the file will be copied to the project directory and the appropriate summary and category information added to the report format information for the project directory.

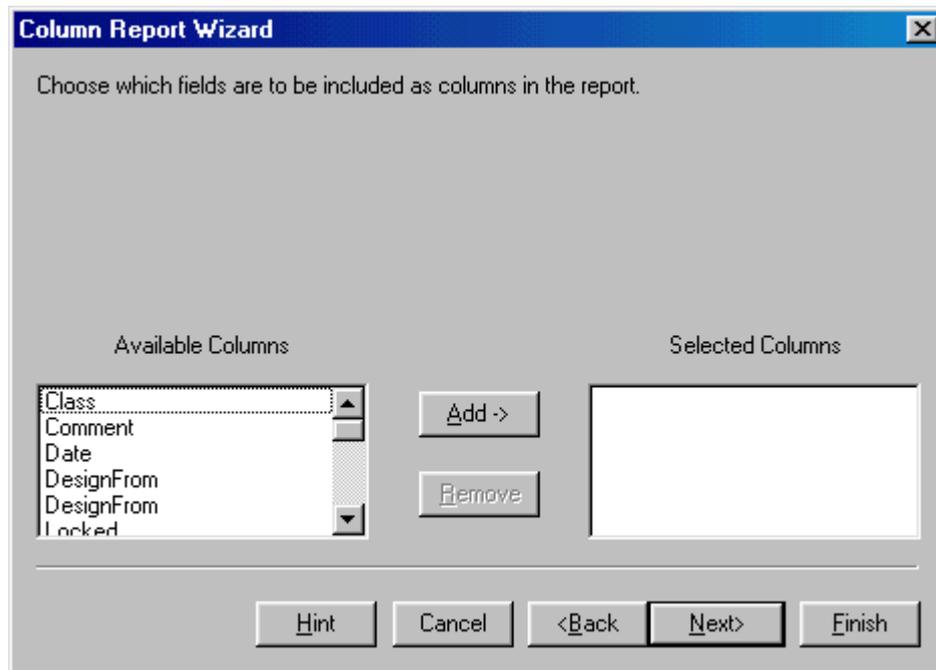
Column Report Wizard

The **Column Report Wizard** is available from the **Reports | Report** menu. It allows you to generate a new report format file which may be accessed from the **Report** dialogs. The report formats generated are for column based reports output to Word or Excel.

The first screen requires that you select which database is to be reported on.



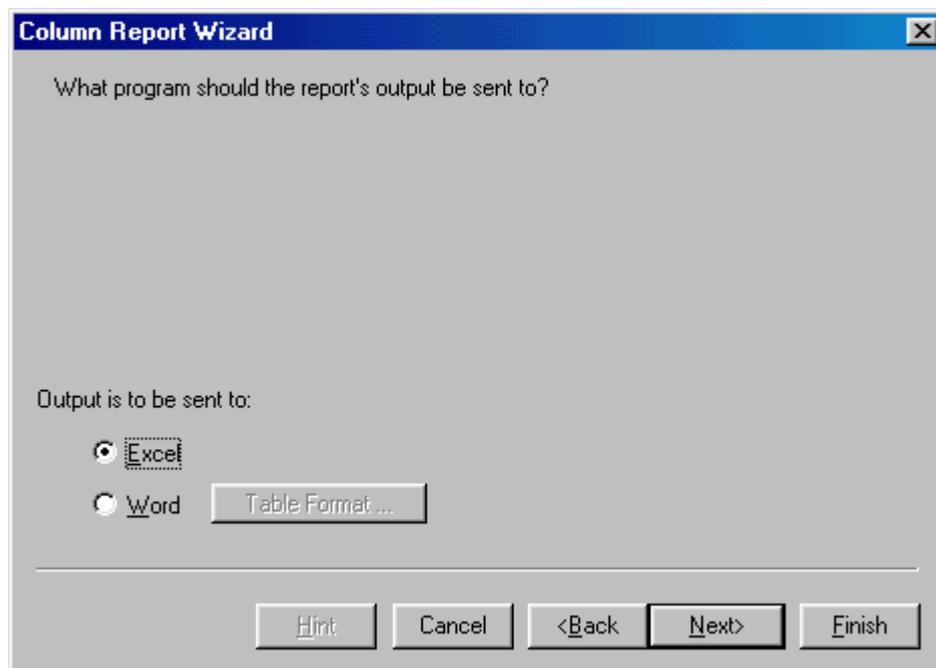
The next screen allows you to select which fields of the database are to be columns of the report.



Select one field at a time and use the **Add** button to add fields to the **Selected Columns**. 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).

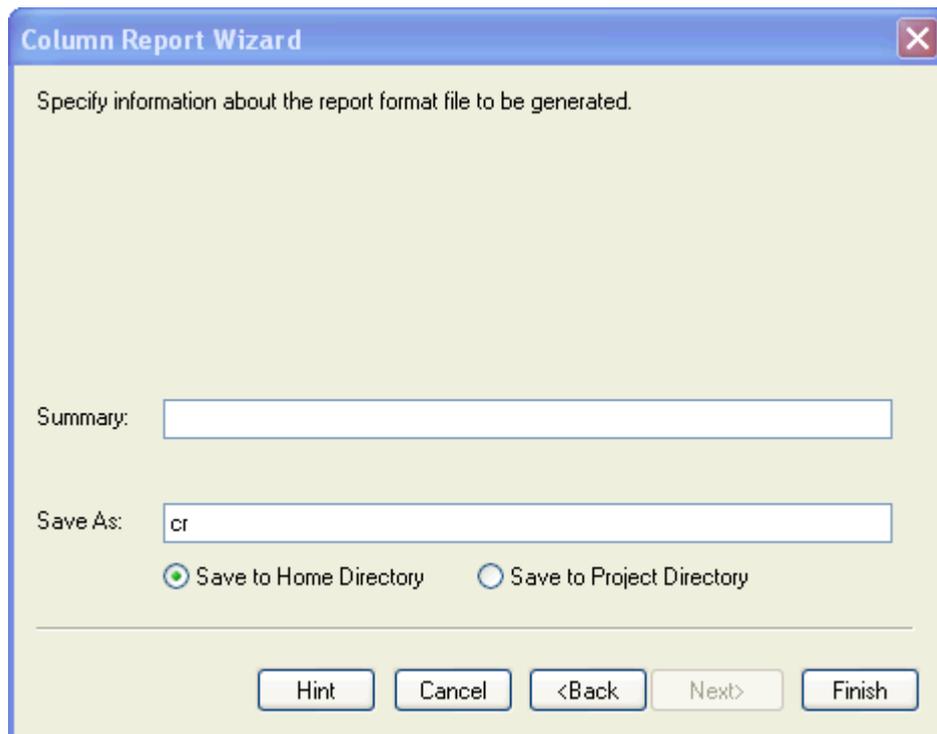
The order may be changed by removing entries with the **Remove** button and then adding them again in a different order.

The next screen allows you select whether you wish the output of your report to go to MS Word or MS Excel.



If Word is selected then the format that the word table should have in the resulting reports may be selected using the **TableFormat** button. This presents a list of the table formats supported by Word and whatever is selected will be used when the report being generated is run.

The last screen requires that you enter some information about the report format to be generated.



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

A file name must be specified for the report format to be **Saved As**. The name used will determine when the report is offered on format lists of available reports. 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.

If you are an **AllChange** administrator then you may choose whether you wish to save the new report format in your **Home Directory** where it will only be available to you, or in the **Project Directory** where it will be available to all users. If you save to your home directory and then later wish to make it available to all users, it may be promoted to the project directory using **Report | Report | Promote Report**

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.

To run your new report use:

either the Excel/ Word category of the **Report Wizard** for the appropriate database

or the format list of the **Report** dialog for the appropriate database.

If your new report does not appear in the report list/ format list, then ensure that the file name you specified started with the appropriate 2 letters for the database being reported on.

Access Control and User Roles

Access Control and User Roles

User Roles may be used to implement access restrictions within **AllChange**. Roles effectively define groups of users with permission to perform various actions on certain areas of the database.

Roles are assigned to you by your **AllChange** Administrator.

A user of **AllChange** has access to commands and information based on whether the user has the required role for a part accessed. The **AllChange** Administrator will have defined what role a user must have in order to execute most **AllChange** commands — note that this is totally site-defined, not hard-coded into the system (i.e. the precise access controls implemented will depend on how the system has been configured by your **AllChange** Administrator).

If you attempt to perform an operation on an item for which you do not have permission, an error message will be issued, for example:

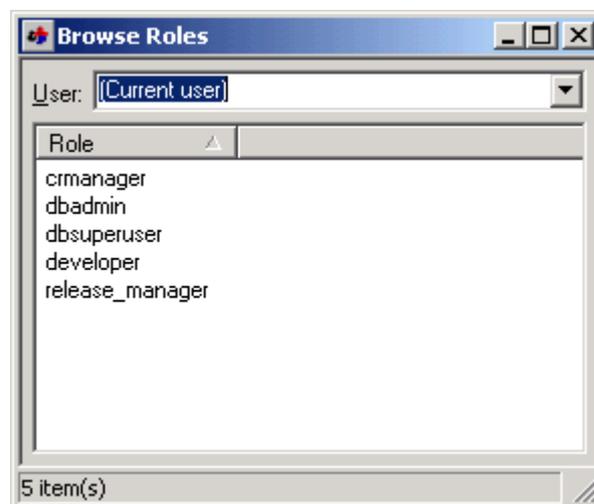
```
Developer permission required for part (part)
```

This would indicate that you require a developer role for the *part* being accessed.

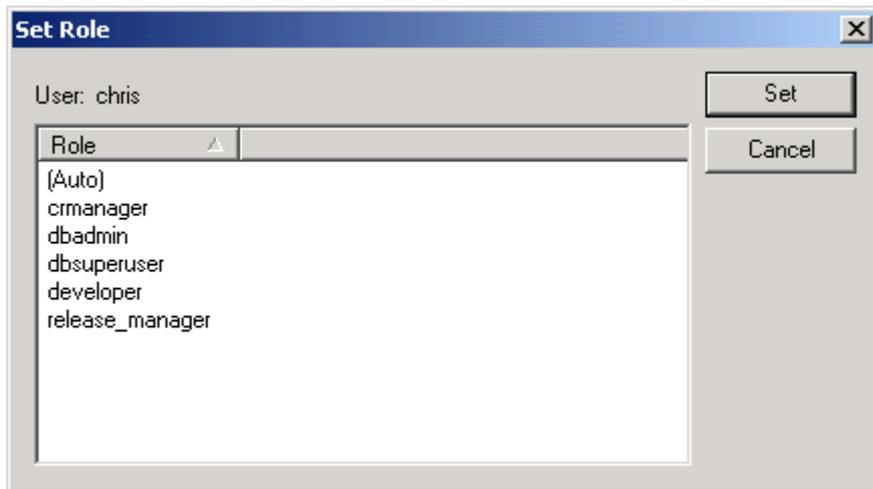
The **Role** status bar item shown at the bottom of the ACEWindow shows the role that you have currently selected.

If a role is shown then you may only have permissions for that role. If **(Auto)** (or no role) is selected then you may have permissions for all the roles that you are a member of. If you wish to restrict yourself to the permissions for one role only, then a role should be selected. For example, if you are a member of both `manager` and `developer` roles and you wish to ensure that you do not inadvertently perform any operations that only a `manager` may perform.

You can find out what roles you may have by using the Role Browser. This may be accessed by **Misc | Browse Roles**.



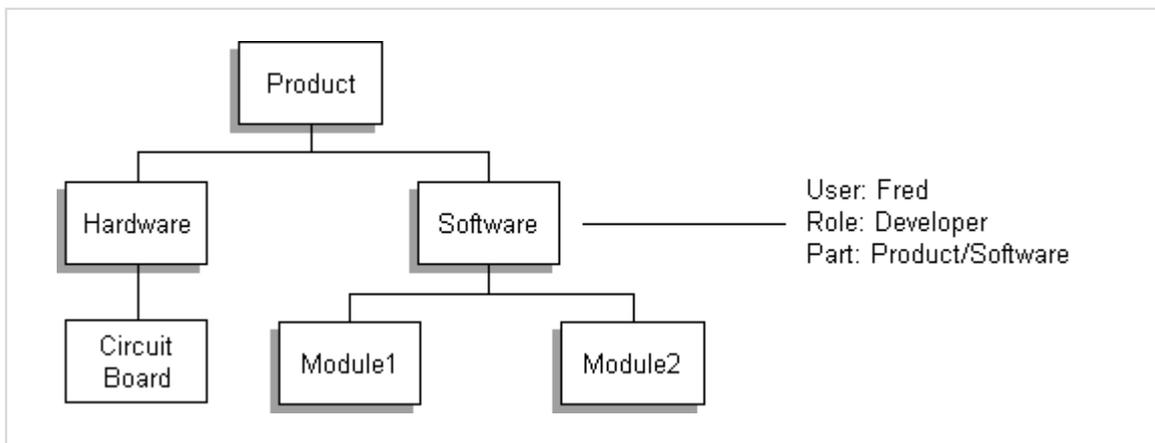
In order to set the role that you wish to use either the role browser or the **Set Role** dialog – select **Misc | Set Role** (or double click on the role status bar item).



In both cases select the required role and from the browser select **Set Role** from the context menu, from the dialog select the **Set** button. This will set the current role to the selection and show this in the status bar.

If the role **(None)** is selected then this will set the current role to no role and **(Auto)** will be shown in the status bar.

Since roles are associated with parts, then just because you may have a `developer` role, this does not mean that you may have a developer role for all parts, you may be a developer for a source code for example but not for design documents.



Various attributes are given in the definition for each role and these may be accessed for reporting/ querying purposes as fields of the roles database.

Customising your Workspace

About Customising your Workspace

A workspace is a directory in which workfiles reside and editing/ compiling etc. takes place. For a full description of workspaces see [About Workspaces](#)

You can customise workspaces to:

- Define the [Default Version](#) for each workspace [using registrations](#).
- Define which pools are to be used for each workspace [using registrations](#)
- [Define which workspace is to be the default on startup](#)

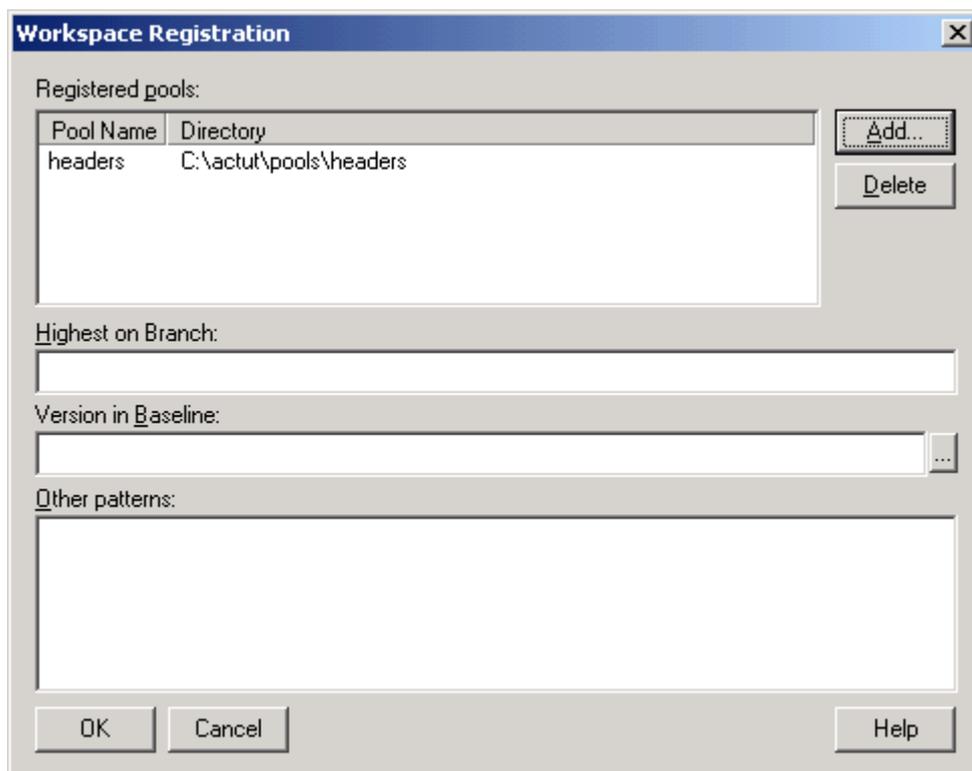
Workspace Registrations

About Workspace Registrations

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.

Registrations are so called because they are used to "register an interest" in particular versions and pools.

Registrations are defined in a register definition file which is a text file created and edited using either **Work-space | Workspace Registrations** or a standard text editor.



See [Registering to Pools](#) and [Default Version Registration](#) for details.

Register definition files are called `register.ac` and must reside in the *workspace directory*.

Workspace Registrations is a standard supplied ACCELfunction which may be modified to suit individual site requirements.

Registering to Pools

If pools are being used, then different workspaces may need to use different pools for build purposes.

The pools that are to be used should be selected using the **Add** button on the **Workspace Registration** dialog. They may be unregistered using the corresponding **Delete** button.

Any pools that are registered to a workspace will be used for build purposes and will also be used for auto-update pool purposes.

These are stored as the first line of the `register.ac` file in the form:

```
pool-names
```

Pool-names is a whitespace-separated list of pools associated with the workspace. If no pools are applicable the line should be a blank one.

Default Version Registration

The **Workspace Registration** may be used to register an interest in specific versions of components. This is achieved by defining what the *default* version for components is to be.

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** specifies 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.

Thus `/project/file;001` would mean simply that: when attached to the workspace and referencing the part `/project/file` without specifying a particular version, then the default version will be version 001.

The pattern `*.c;003` would mean that version 003 would be used for any parts ending in `.c`.

The pattern `/project/*.pas;!bln1` would mean that for any part ending in `.pas` anywhere within the `/project` hierarchy (including any subsystems) the version appearing in baseline `bln1` would be used.

It is also permissible for the version pattern to specify just a branch without the final version number in which case the highest version on that branch will be used, so that `*;branch1` would ensure `branch1` was the default for all parts.

Parts are matched against patterns in the order the patterns appear in the register definition file with the first match encountered being adopted. Hence, for example, an entry of `*.c;001` should come before an entry of `*;002` to set the default version for `.c` parts to 001 and that of all other parts to 002; putting them in the opposite order would not make sense.

All the lines in the `register.ac` file apart from the first are used to define the rules for determining the default version of parts.

Attaching to a Workspace/ Role Automatically

If you wish to automatically attach to a particular workspace whenever you start up **AllChange**, or select a particular role, then this may be achieved by specifying which workspace and/ or role you wish to always start with in the **Misc | Options**, see [Options](#).

You may also automatically attach to the same workspace and role that you had last time you were in ACE by saving your settings in your registry. Ensure that **Misc | Save Settings on Exit** is selected and these values will be saved.

The next time you start **AllChange** your workspace and role will be set to the value last saved.

Customising the User Interface

About Customising the User Interface

Various aspects of the user interface may be tailored from the standard which is initially supplied.

Some of these customisations need only be performed by the **AllChange** Administrator, some may be set up for each user individually.

The following aspects of the user interface may be customised:

Browser Contents

The precise content of the browser lists may be defined as required.

Shortcuts

User defined shortcuts may be created and placed on the shortcut bar or the folder list

Views

Which panes are displayed may be tailored as well as the style of list displayed for windows containing list views. Select [Cycle View Bar](#) to display a graphic representation of the currently selected life-cycle.

Options

A variety of different options are available for users to tailor the GUI to their preferences

Environment

Various aspects of the environment used by **AllChange** may be altered.

Menu Options

The content of the pull down menus may be configured to provide access to user defined ACCEL functions (Administrator Only)

Menu & Toolbar Item Availability

The items on toolbars and menus may be configured as to when they are available (i.e. not greyed) (Administrator only).

Arbitrary Field Labels & Content

Both the labels for arbitrary field and the valid values may be configured (Administrator only).

Those items marked as **Administrator only** are only expected to be configured by your **AllChange** Administrator and are project specific (not user specific) configuration options. If you would like to see changes to these aspects of ACETHen you should discuss your requirements with your Administrator.

Other configuration options may be specified for each user and you may therefore change these without affecting other users.

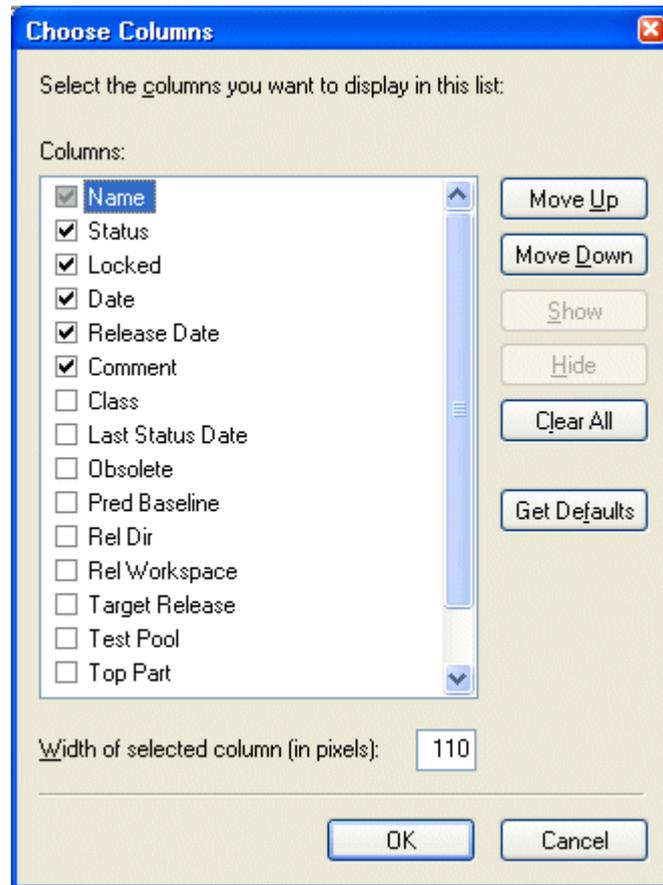
Defining Browser Contents

The contents of browser windows and other lists may be tailored to individual preferences.

Which columns are shown, the order in which they are displayed and the column widths are all tailorable as described below.

Adding and Removing columns

Right click on the column heading bar and select **Add/ Remove Column**



The columns displayed are those which are selected (ticked) in the **Column Selection** dialog. They may be added and removed from the browser by changing the selection.

Certain columns in the **Column Selection** dialog may not be changed (i.e. they may not be unticked). These column(s) represent the name/ identifier for each item/ database record shown in the browser. Generally just the first column is fixed but for some databases (e.g. check-outs), more than this is required to identify the item (the part name and the workspace checked out to for check-outs).

New columns selected will be added as the last column in the browser which may then be moved to the desired position.

Reordering browser columns

In order to move a column to a new position simply drag it to the required position.

Note that fixed columns may not be moved.

Preview Pane

Each browser may have a preview pane to show a summary of a selected item in the browser. This may be switched on/off using the **View** menu.

It is also possible to change the sort order of most browsers by clicking on the column to be sorted by. This facility is only available for the CR browser if the **Allow CR browser header sort** Option is selected. If sorting is permitted then it should be noted that the sort will only take place within the current buffer loaded into the Browser. For full database sorting the **Filter By** should be used.

Shortcuts

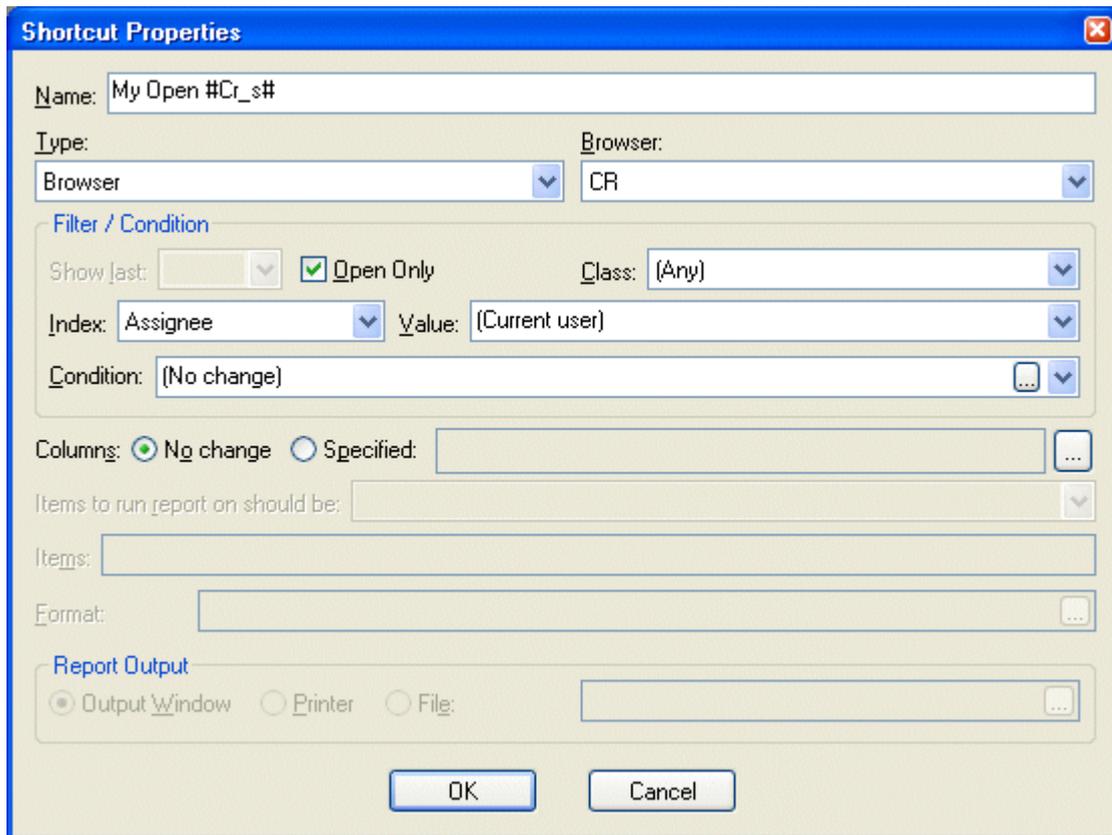
Both of the Shortcut bar and Folder list contain user configurable icons, which can be used to open a browser window, run a report, or invoke an ACCEL command or operating system command. This can be very useful for fast access to frequently run reports or queries run in a browser.

Both the Folder List and the Shortcut bar may be closed if not wanted using the **View Menu**.

The Shortcut Bar includes groups, which each contain a number of items. The user may add, remove or rename groups by using the bar's context menu, found by right-clicking on a group label or in the bar's free space.

The user may also add new shortcuts, change the icon view-style or hide the bar, from this menu.

Shortcuts may have the following properties:



Name: this is name that will be shown with the icon

Type: this determines the type of function that the shortcut performs. It may be one of:

Browser: this will cause a browse window to be displayed

Report: this will allow a report to be run

Eval Command: this allows an ACCEL expression to be evaluated

OS Command: this allows an operating system command to be executed

Browser: defines which browser is to be displayed for *Browser* type short cuts.

Database: defines which database is to be reported on for *Report* type short cuts

The **Filter/Condition** group specifies the filter and condition for a browser or report shortcut:

Show Last: specifies the number of baseline versions to be shown for a baseline browser short cut

Open Only: specifies whether only open CRs should be shown

Type: specifies the baseline type, for the baseline browser, out of any combination of Design/Release and Meta/Non-meta

Class: specifies a class by which to filter the CRs or Baselines in the relevant browser

Check-outs: specifies a filter for the Check-outs browser to show **Any**, **For Edit** or **Read-only** check-outs

Index: specifies the index to be used for a browser or report short cut

Value: specifies the index value for a browser or report short cut.

Condition: defines any conditions to be applied to a browser or report short cut

Columns: defines the columns to be shown in the browser. This may be the current browser column settings by selecting **No change** or may be selected by selecting **Specified**. If **Specified** then the columns and their order may be selected by selecting the ... button. On selecting the ... button the columns initially selected will be as per the current open browser if there is one, otherwise the default for that browser.

Items to run report on should be: specifies how the items which are to be reported on should be specified for a report short cut. This may be:

- **All the items in the appropriate database**
- **Selected from the appropriate browser or viewer**
- **Prompted whether to use selected or all items**
- **Specified in the shortcut**

The default is "Prompted...".

The 'Items control is disabled unless "Specified..." is chosen.

Items: specifies the items to be reported on for a report short cut. This may only be specified where the **Items to run report on should be** selection is **Specified in the shortcut**

Report Output: specifies where the report results should be sent for a report short cut. This may be:

Output Window — If the report is a native **AllChange** report (*.acr) then the report output will be displayed in its own window. If the report is a textual report (*.rep) then the report output will go to the Output Window. This is the default option.

Printer — this will cause the user to be prompted for a printer to send the output to. The user can also select the page size, and orientation. Note that for a native **AllChange** report (*.acr) the page size and orientation are set in the report itself, so the choices made here are ignored.

File — this is valid only for textual reports (*.rep), and causes the output to be written to the specified file

Format: specifies which report format is to be used for a report short cut.

Right-clicking on icons offers the user a context menu where items may be opened, removed, renamed, copied or have their properties edited.

The Folder List offers the same facilities as the Shortcut Bar, except that the items are presented as a tree, allowing the user to organise them into a hierarchy.

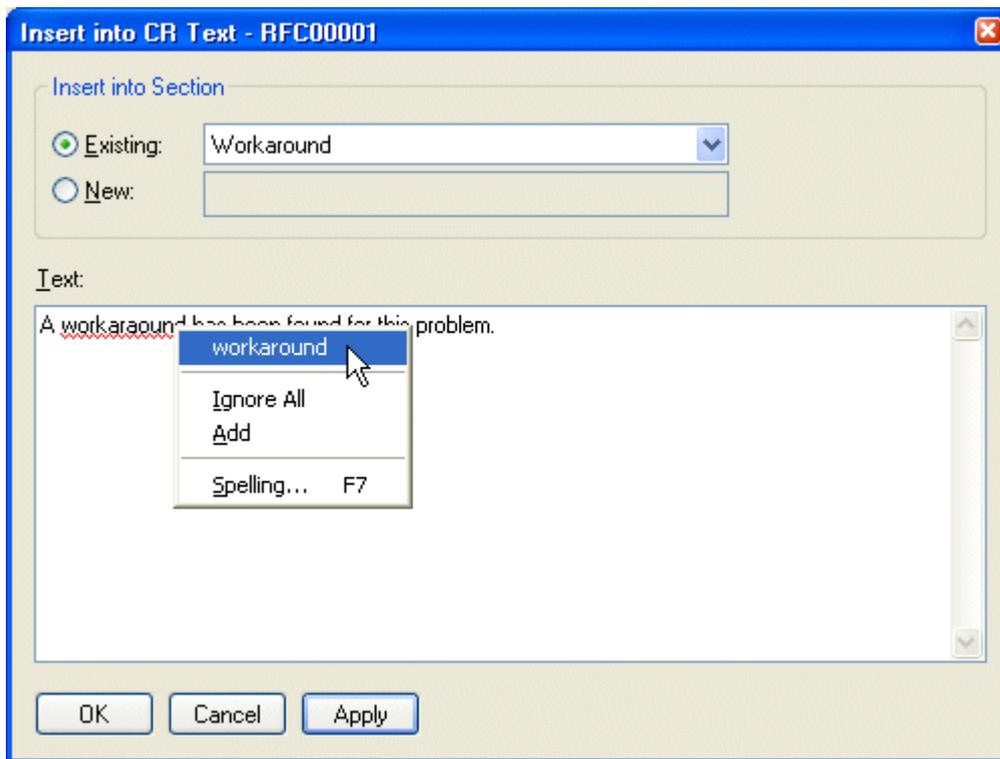
Icons can also be moved or copied within the folder list or shortcut bar. To copy an item, hold down the Ctrl key. Items may be dragged between the shortcut-bar and the folder list, in either direction, in which case the item is copied by default. To achieve a move, the Shift key should be held down. Folders in the shortcut-bar may be moved or copied only within the shortcut-bar itself. If dragging with the right mouse button then a menu is displayed on release, where the user may choose to copy, move or cancel the operation.

Changes to the bars are personal to each user and are stored in the user's registry profile, however defaults may be created by **AllChange** administrators by selecting **Save Shortcuts** from the right click context menu for the shortcut bar or folder list. This will save the current shortcut and folder list settings to the **AllChange** project directory. These default settings will be used by any user who has not saved personalised settings. To restore to the project defaults, select **Misc | Clear Settings** and select **Short Cuts**.

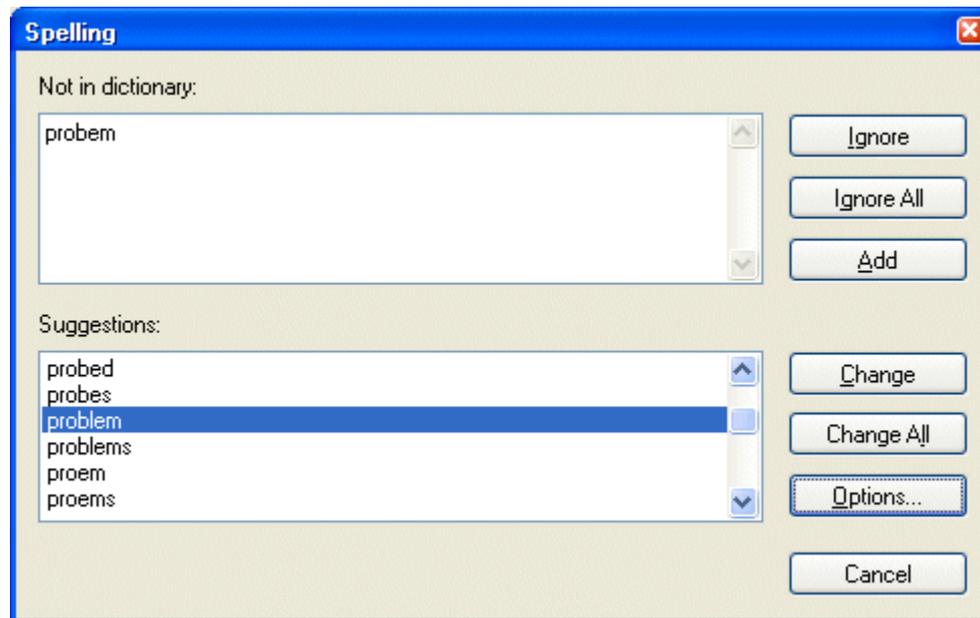
Either of the bars may be turned on or off independently of the other from the **View** menu.

Spell Checking

Spell checking is available in the text tabs of the CR, Baseline and Part viewers, in other text fields for which the Administrator has enabled it, and the Insert into Text dialog. Depending on user options, misspelled words are shown underlined by a red wavy line. Right-clicking on the misspelled word shows a list of suggested correct words. The word may also be ignored, or added to the user's dictionary.



A spell-checking dialog may be opened from the text field's context menu or by pressing the F7 key. The dialog finds each misspelled word, suggests replacements, and allows the user to ignore the word or add it to their dictionary. If a misspelled word is not found in the dictionary, and appropriate suggestions are not found, then the misspelled word may be over-typed in the 'Not in dictionary' field, and "Change" clicked to replace the word.



A Spelling options dialog may be opened from the spell-checking dialog, or by pressing Ctrl+F7 from the text field. The spelling options also appear on the main Options dialog available on the Misc menu. See the [Options](#) topic for more information.

Note that, on the text tab of viewers, if the text is protected against direct editing then spell checking is not available. It is still available on the Insert into Text dialog however.

Views

The **View** menu provides access to facilities to select which windows/bars are to be shown and display options for the windows.

The **View** cascade menu provides options to select the display type of list views as:

- Large Icons - shown as a large image, with a simple caption
- Small Icons - as above, but with small images, listed left to right, top to bottom
- Details - small images and captions, with columns to show additional information
- List - as small icons, but listed top to bottom, left to right
- [Folding](#) - as details, but able to show related items as children of listed items

The style of list view may be selected independently for each Browser window. For speed, when showing views other than Details or Folding, column information is not retrieved. The columns are only evaluated on first switching to a view with columns.

Show Obsolete Items determines whether items marked as obsolete are to be shown in browser windows. If this is selected then any obsolete items will be shown dimmed to distinguish them from normal items

Shortcut bar, Folder list, Output window respectively close/remove from the display the named pane/window

Clear output window allows the output window to be cleared of any text

Cycle View Bar allows the life-cycle for items in **AllChange** to be viewed and updated. The [cycle-viewer](#) is displayed in a docking window, which may be attached to any side of the main **AllChange** window.

Preview Pane if selected will show a *preview* or summary of the content of an item selected in a browser window. The preview pane may be switched on and off for each browser independently.

Toolbar allows items on the toolbars to be added to and deleted from

Folding View

The folding view allows a list to be displayed as a tree. It is supported by various browsers and viewers.

The folding view displays placeholders for elements of the tree which are not in themselves items in the list. These placeholders will be shown using a grey icon, and grey text.

Items which can be expanded will have a '+' button to the left of its icon. AllChange will only evaluate children of items when the item is first expanded. Therefore, items may be shown with a '+' button regardless of whether the item has any 'child' items. On expanding the item, the '+' is removed if no children were found. Items for which the lack of children can be determined without additional database access (e.g. a part baseline as opposed to a meta baseline) will be shown without the '+' button.

The information shown by the folding view (i.e. the items that are folded) vary depending on the type of item being shown.

Versions Folding View

The Part Browser's right-hand pane shown in a folding view shows the version tree for the component. Components will be expandable to a list of the (non-branched) versions of the component. Branches are shown as placeholders which may be expanded to show the branched versions.

Parts List Folding Views

Lists of parts shown in a folding view show the hierarchy of the path to the parts contained in the list. Sub-systems which do not actually exist in their own right in the list are shown as placeholders (i.e. as a grey icon) whilst subsystems which are items in the list are shown as normal in blue.

For the Check-outs browser components are also shown as placeholders with the expansion showing the specific component versions that are checked out.

Parts folding views are supported for the Check-outs browser and all lists of parts displayed in viewer tabs such as the Parts tab of the CR viewer, the Find window also supports a folding view.

CRs Folding Views

Lists of CRs shown in a folding view show the relationship between CRs. If a CR has any CRs affected, then these are shown on expansion.

CRs folding views are supported by the CR browser and all lists of CRs displayed in viewer tabs such as the CRs Solved tab of the Part viewer

If the [Show only orphans at top level in folding view](#) option is set for the CR browser then only CRs which have crsaffected and CRs which are not referred to by any other CR will be shown.

Baselines Folding Views

Lists of Baselines shown in a folding view show the relationship between baselines. Expansion of a meta baseline shows the baselines contained within the meta baseline.

In addition, if the [Fold baseline versions in folding view](#) option is set then placeholders for the baseline basename are shown, and on expansion, if the baseline has versions, each version of the baseline is shown.

Baselines folding views are supported for the baseline browser and all lists of baselines displayed in viewer tabs such as the Meta tab of the baseline viewer.

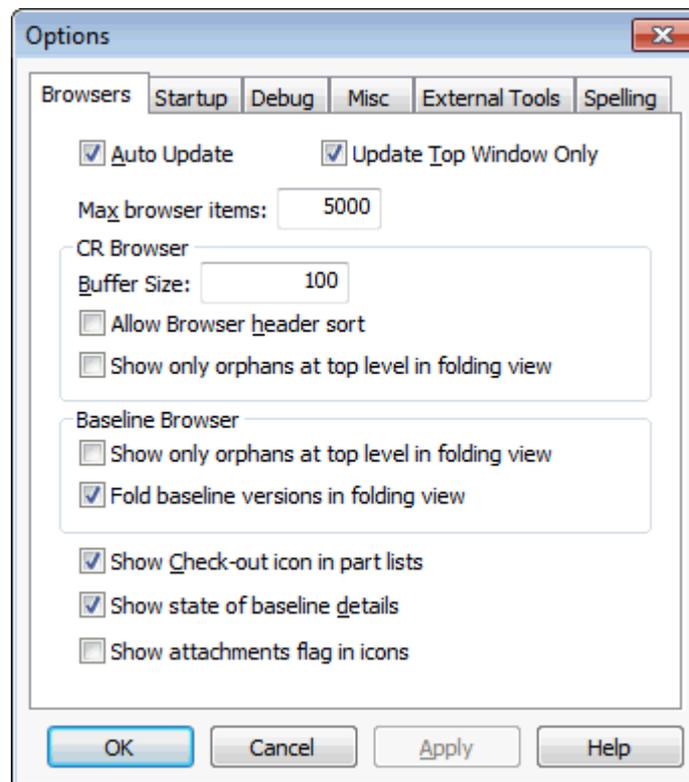
If the [Show only orphans at top level in folding view](#) option is set for the baseline browser then only meta baselines and parts baselines which do not appear in any meta baseline will be shown.

Items Affected Folding Views

When Items Affected browsers are shown in Folding view, the list contains placeholders for the Item Affecting, with a child for each of its Items Affected.

Options

Misc | Options may be used to define various personal settings which tailor the behaviour of ACE.



Browsers

This tab allows certain features of browser windows to be configured.

Autoupdate determines whether to update all open child windows showing databases which have changed after every **AllChange** command has completed. If selected this ensures that whatever changes are caused by any command are reflected immediately. However, if there are many open windows, the database is large or system load is heavy then performance may suffer a noticeable degradation: there may be a pause at the conclusion of certain commands. Switching off autoupdate may be desirable in this situation: the current *active* top window will still be automatically updated, while the **Refresh** toolbar button (or option from the **Misc** menu) may then be used from time to time to cause the current state of the databases to be updated when required. This may also be used to see any changes caused by other concurrent users of the system.

Update Top Window Only determines whether all open windows are updated after every All-Change command or just the current *active* top window, and other windows are updated when they become active. This only has an effect if **Autoupdate** is selected.

Maximum browser items: the **Maximum** number of browser items to be read into any browser. If this number is reached a message is issued and the user is given the opportunity to improve

the filter being used. If this number is very large then the resulting list can be too large to be useful. The default value is 5000.

CR Buffer Size: this allows the buffer size for the CR browser to be specified. It should be the number of CRs to be read in at one time in the CR browser. The default buffer size is 100.

Allow CR browser header sort: If sorting is permitted then it should be noted that the sort will only take place within the current buffer loaded into the Browser. For full database sorting the **Filter By** should be used.

Show only orphans at top level in folding view: if selected ACE will show only items which have other associated items in the list, or items which are not associated with any other item in the list (i.e. they are orphaned). This option may be set separately for the baselines and CRs.

Fold baseline versions in folding view: if selected ace will show baseline versions folded. i.e. placeholders are shown for baseline basenames, the expansion of which will show the versions of the baselines.

Show Check-out icon in part lists: if selected, ACE will show the check-out state of a part in the part browser

Show state of baseline details: if selected, ACE will show an icon displaying the existence - or otherwise - of each detail in a baseline in the baseline viewer parts tab. If **Show check-out icon in part lists** is also selected then the check-out state of each part will be shown.

Show attachments flag in icons: if selected, ACE will show a paperclip on the icon in browsers to indicate the existence of an attachment to a CR or a baseline. Note that this can have an impact on the speed with which the list is filled, especially on systems running over a slow network or WAN.

Startup

This tab allows various ACE startup options to be specified.

Initial Role : the role that should be always selected on startup. This overrides any saved setting.

Initial Workspace : the workspace that should be always selected on startup. This overrides any saved setting.

Always use this project forces ACE and ACCONFIG to always start up using this **AllChange** project and not prompt for the project to use.

Disable splash screen on startup prevents the display of the splash screen while ACE is starting up.

Debug

Debug mode causes ACE to start up in *debug mode*. This means that trace and echo are switched on.

ACCEL warnings allows the ACCEL warnings level to be set. This determines whether warnings are issued for certain operations within ACCEL code. Values may be:

- 1 : warn on attempt to access an undefined variable
- 128 : treat any warnings as errors

Misc

This tab provides access to various miscellaneous options.

Prompt whether to send further errors to output: if selected the error dialog used by ACE allows multiple error messages to be diverted to the output window.

Send Errors To: allows the user to control where error messages are displayed.

Both: sends error messages to both a dialog and the Output Window (this is the default)

Dialog: sends error messages to a dialog only

Output: sends error messages to the Output Window only

Show Output Window: allows the user to control the circumstances under which the Output Window is shown (up-fronted) after running a command.

Default: show the Output Window whenever explicitly requested by the command executed. This will tend to be whenever there is *interesting* output, e.g. after running a report where results are sent to the Output window.

Always: show the Output Window whenever any output is produced

Had Error: show the Output Window whenever explicitly requested (as for **Default**) or when an error message is output

Never: never show the Output Window

Confirm on update

If this is selected then when a record is updated from information entered on a viewer and you hit **Return** or **Enter** you will be prompted to confirm that you wish to update. If you select Misc | Update then you will not be asked for confirmations. If this is not selected then the update will happen automatically.

Read-only CR Text background colour

Re-sets the background colour of the CR text in CR viewers to a colour of choice. Setting the colour to 'Automatic' will use the colours set in Windows preferences.

Home directory: allows you to specify what should be used as your home directory.

External Tools

This tab allows various external tools to be specified.

Default Editor: allows you to specify your preferred default editor. This will be used when there is no edit action associated with a file.

Default Viewer: allows you to specify your preferred default viewer.

Differencing Tools: allows you to specify your preferred file comparison tools by different file extensions and a **Default Differencer** which will be used if there is no specific one. If no differencing tool is specified then if the file is a Word document then Word will be used, if it is a binary file (one listed in Binary suffixes) then a simple comparison for equality will be performed otherwise the default is **VisDiffs** as supplied.

If using Araxis then the `compare.exe` tool should be specified and not `merge.exe` if the full Araxis interface functionality is required. For full Araxis interface functionality the All-Change administrator must also have enabled it. Full Araxis integration allows:

- Subsystem Comparison
- Hierarchical baseline comparison
- 3 way directory comparison
- Subsystem Merging

in addition to the basic functionality which includes:

- File comparison
- Component version comparison

Spelling

This tab allows spell-checking options to be specified.

Check spelling as you type: this option determines whether misspelled words will be shown with a red under-line in item text fields, and arbitrary fields which have spell-checking enabled.

Ignore words in UPPERCASE: any words which are entirely in upper-case will be ignored if this option is set.

Ignore words with numbers: this option causes words containing numbers, e.g. 'Apollo13', to be ignored.

Language: Specifies the language to use when checking words in the supplied dictionaries. This will default to English (UK) unless the client is running in the US, in which case it will default to English (US).

User dictionary: specifies the path of the user's dictionary, which is where words are added when using the Add function in the spell-checking dialog. This defaults to 'User.dic' in the user's AllChange home directory. Clicking the "Edit..." button opens a dialog where the user may edit the contents of their dictionary.

The Environment

Various aspects of the environment used by **AllChange** are specified by the project definitions and by the values contained in certain environment variables or in the registry.

Personalised settings are also saved in the registry such as the size of windows, which windows are open, browser column settings, the toolbar settings etc.

If you wish to revert to the default project settings then **Misc | Clear AllChange Settings** may be used. This allows selective groups of settings to be restored to the project defaults.

The following groups may be restored:

- **Application settings:** this is the saved sizes and positions of windows, and information such as browser condition history, browser columns, etc. This is equivalent to deleting the section "Ace-Settings" in the user's registry.
If **Restore defaults now** is selected then all open windows will be reset immediately reset all open windows immediately. If not their settings will not be restored until they are closed and re-opened
- **Toolbar settings:** this is any customization that has been made to toolbar buttons, additional toolbars, and their sizes and positions. This is equivalent to deleting the section "AceToolbarSettings" from the user's registry, and resets the toolbars to the default state. This also resets the output window, shortcut bar, and folder list to their original position.
- **Shortcuts:** this will remove all shortcuts from both the shortcut bar, and the folder-list, and restore the default shortcuts.

The selected settings are restored as soon as **Ok** is clicked. Any changes to windows settings from then on will be saved as normal.

For full details of configuring the environment see the **AllChange Administrator Manual**.

AllChange Web Browser Interface

About AllChange Web Browser Interface

This chapter describes the Web Browser interface to **AllChange**. This allows, for instance, users at a remote location to run **AllChange** even though they are not connected directly to the network. Most facilities/functionality offered by the Windows interface is supported by the browser interface including running reports, support for Change Requests, Parts and Baselines and Mailing.

In order to use these facilities your **AllChange** administrator will need to have configured the **AllChange** system and your Web server appropriately, see the **AllChange** Administrator Manual for details.

You should run your web server in the normal way and access the **AllChange** Interface pages.

This chapter describes the initial supplied configuration of the **AllChange** Web Browser interface — it may have been configured/ modified for your particular site requirements.

The supplied interface works with Web Browsers supporting HTML including:

- Internet Explorer (version 6.0 and upwards)
- Mozilla Firefox (Version 2.0 and upwards)

The browser must support Frames, Java and Javascript.

The Java Runtime Environment must be installed on the client machine.

Cookies should be enabled if user settings are to be retained.

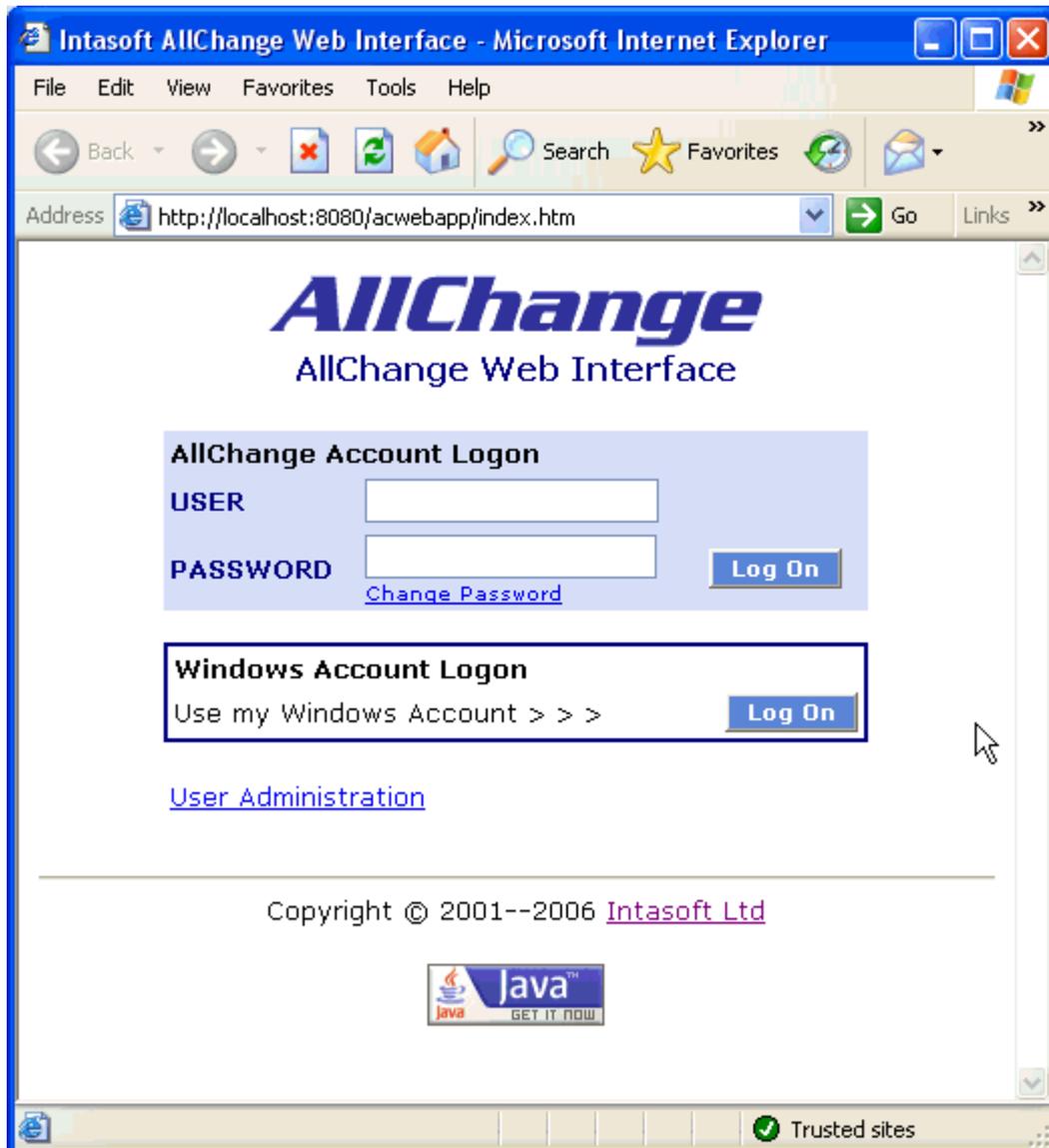
It is assumed that you are familiar with the features of the particular browser you are using.

To access the **AllChange** Web Interface, simply point the web browser to the root location of the **All-Change** web application. This will by default be something like `http://<servername>/acwebapp/`.

This will bring up the [log-on](#) page.

AllChange Browser Interface Logon

The initial page of the Web Browser interface provides for logging on to **AllChange** and for administering the users who have access to the browser interface.



If using Windows Authentication you can logon using your Windows credentials in the Windows Account Logon section. Internet Explorer running in Windows should authenticate and pass across the Windows user automatically. All other browsers, and browsers running in non-Windows environments will prompt the user for their domain user name and password. Windows authentication must have been enabled by your **AllChange** administrator, see the AllChange Administrator Manual.

If using **AllChange** authentication then in order to log on to **AllChange** you must be a registered web browser user, this is in addition to being a registered AllChange user. If you are not registered contact your **AllChange** administrator or see the **AllChange** Administrator Manual.

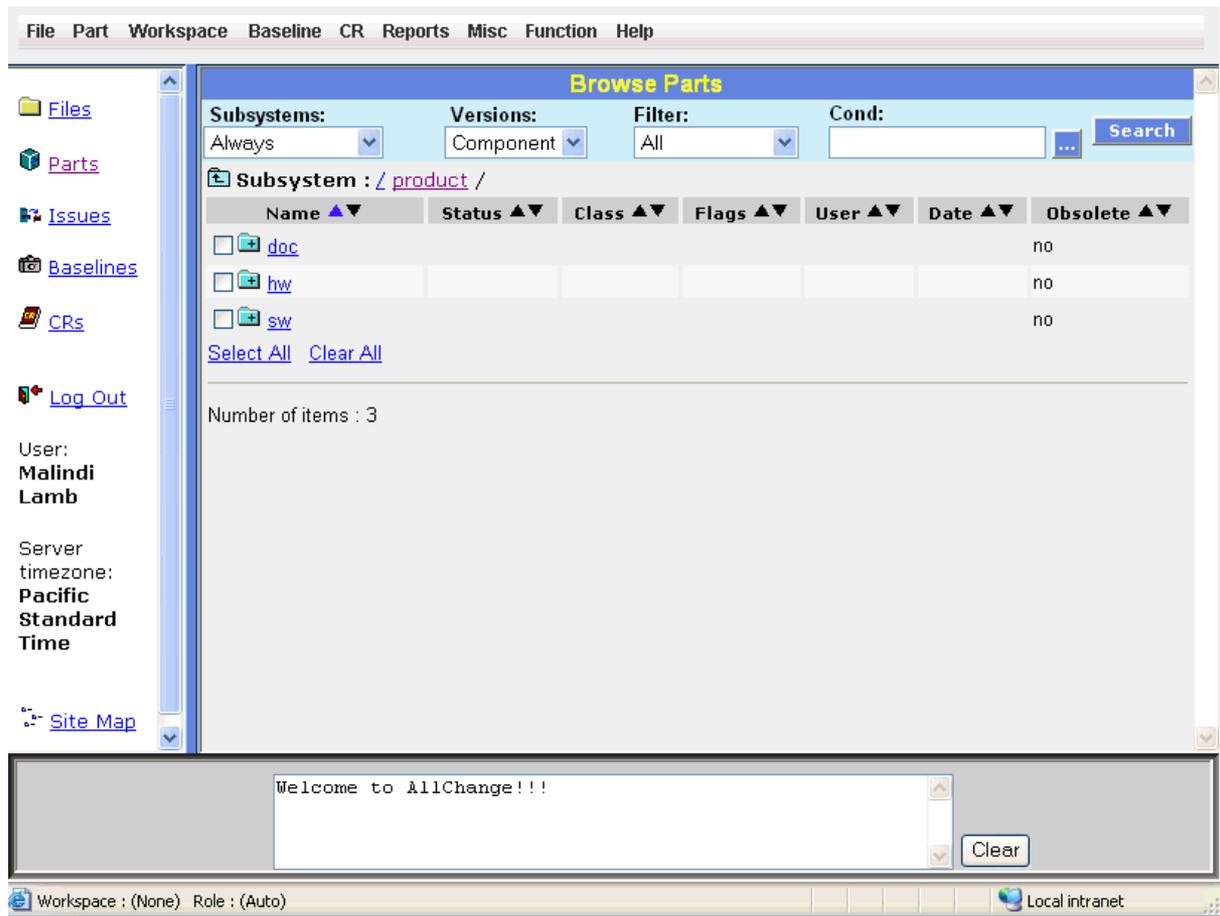
To log on to the web interface User Administration you must be a registered web interface user and enter your **AllChange** user name and password.

When you log on, if there is more than one **AllChange** project that you are permitted access to then you will be presented with a list of these. If only one project is permitted then you will be automatically logged onto that project.

Note that logging on to an **AllChange** project may take a little while.

Browser GUI

When logged on to the Browser Interface, the main page will be loaded. This is split into 4 frames :



Menu Frame - This is at the top and has a menu similar to the standard *AllChange* menu. Click on a button to bring up the corresponding menu. Click on a menu item to perform the named option.

Navigation Frame - On the left of the main screen area is the navigation frame which contains links to the File, Part, Check-outs, CR and Baseline browsers.

The navigation frame also allows you to Log Off from AllChange and displays the user that you are logged on as.

Main Frame - This is where most of the information is displayed. This includes browsers, viewers and dialogs.

Command Output - The bottom frame shows any command output text that might be generated.

The general look and feel of the interface is very similar to the Windows Interface (ACE). There is a browser for each database and a viewer to show the details of an individual item. For full details see the relevant database chapter (Creating and Managing Parts, Creating and Managing Baselines and Creating and Managing CRs)

A few points to note where the interface differs between the Windows and the Browser versions:

Browsers:

- The browser columns may be used to sort the browser in ascending/descending order on that column. The current sort order is shown by highlighting the arrow in blue.
- Conditions may be entered using the browser condition editor or as **ACCEL** expressions and the **Search** button used to filter the browser according to the condition.
- Click on the name of an item to view the details of that item
- Items have a check box beside them. Select (tick) the check box to make the item the subject of a command/action from the Menu or Right Click menu.
- For Internet Explorer users, there is a right-click menu on the browsers. If there are no items ticked, then the menu will offer a link to add a new item on the Part, CR and Baseline browsers. If items are selected, then the menu will offer links to a full list of options, e.g. View, Delete, Status, Assign, Copy and Make Obsolete for the CR Browser. The items selected will be used to populate the dialog for the selected menu item.

CR Browser

The next/previous buffer of CRs may be viewed using the right and left arrows at the top left of the Browser screen.

The default buffer size is 50, but this may be modified using **Misc | Options**.

Part Browser

To view the contents of a subsystem, click on the Parts icon in the navigation frame which will display the browser for the last subsystem visited (default will be /).

To view the contents of a particular subsystem, click on the subsystem icon.

To navigate up a level, click on the  icon at the top of the browser. You can navigate to any level in your current hierarchy by clicking on the subsystem name in the **Location**.

To bring up the viewer for a part, click on the name of the part in the list. To view the root part, select **View** from the part menu before you have selected a part to view. Once you have viewed a part (using the browser), **Part | View** this will show you the last part viewed.

File Browser

Displays the content of a workspace, by default it will show the content of the current workspace if there is one. Only Web workspaces may be browsed.

Navigation through subdirectories is the same as for the [part browser](#).

Files may be uploaded and downloaded to/from the workspace as in the [part browser](#). In addition the **UPLOAD NEW FILE** link may be used to upload a currently unknown file from your PC/Network to the workspace (e.g. in preparation to check it in as a new part or a new version of a part which is not checked out).

Viewers

Viewers are accessed by selecting the hyperlink for the item in the corresponding browser.

Information is displayed in viewers in a very similar fashion to the windows interface with Tabs to show different sections of information about the item.

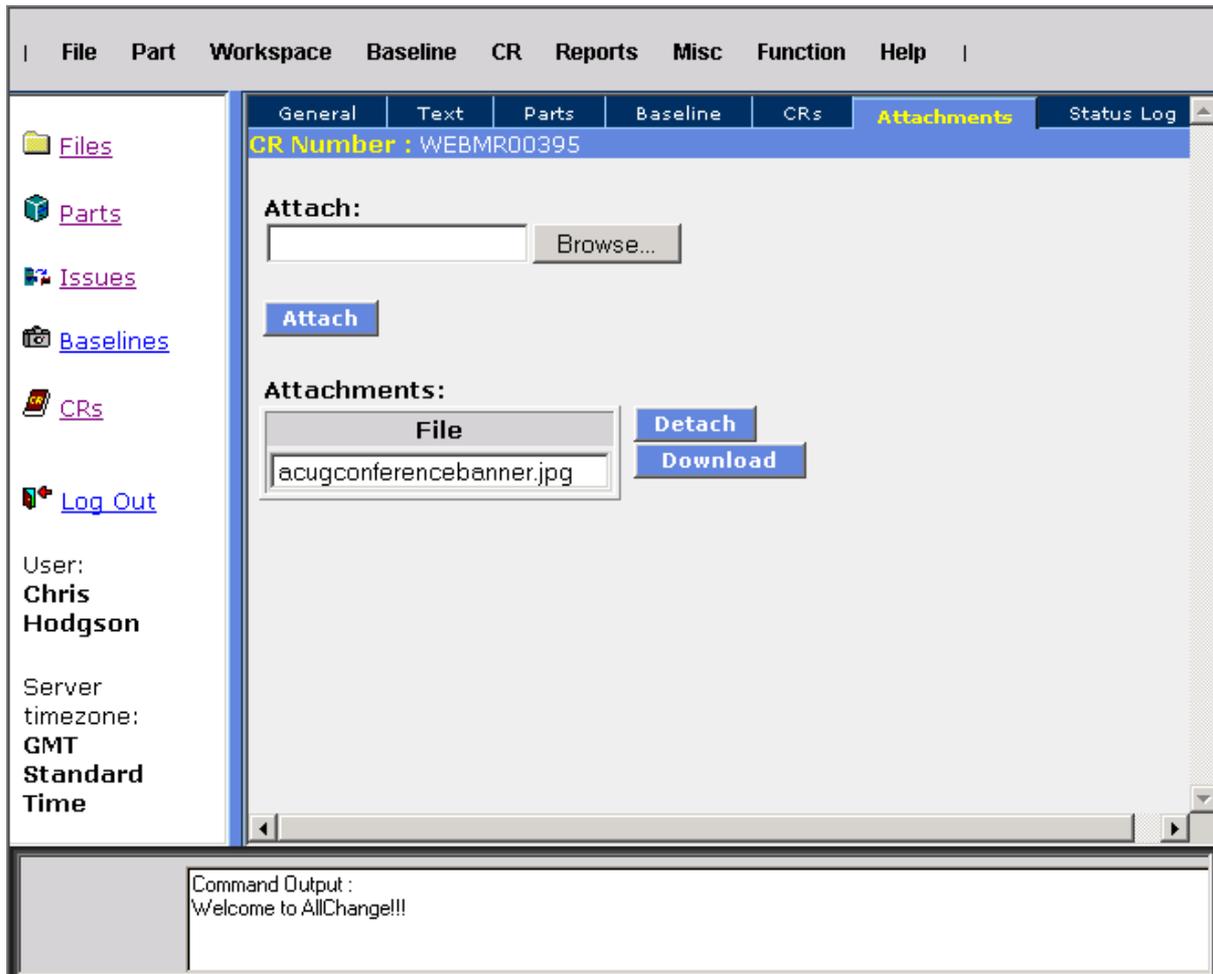
Changes may be made on the page and saved using the **Update** button.

If changes are unsaved when a new tab is displayed an * (asterisk) will be shown next to the item name to show that there are unsaved changes. Selecting **Update** at any time will save unsaved **changes. Parts, Baseline and CRs Affected by a CR**

All the items affected tabs for a CR viewer show the respective items buffered in *pages*. Use **Next/Previous** to move through the pages of information. The default buffer (or page) size is 5 and may be modified using the **Items Affected Buffer Size** option, from the **Misc** menu.

Attachments:

Attachments are fully supported but work in a slightly different way to the Windows Interface.



To attach a new file to a CR/baseline, click the **Browse** button. This will bring up a file selection dialog where you can pick a file to attach. Once selected the file name will be displayed in the **Attach** text box. To actually attach the file, click on the **Attach** button. Note that this may take several minutes for large files. Once attached the file will appear in the list of file attachments.

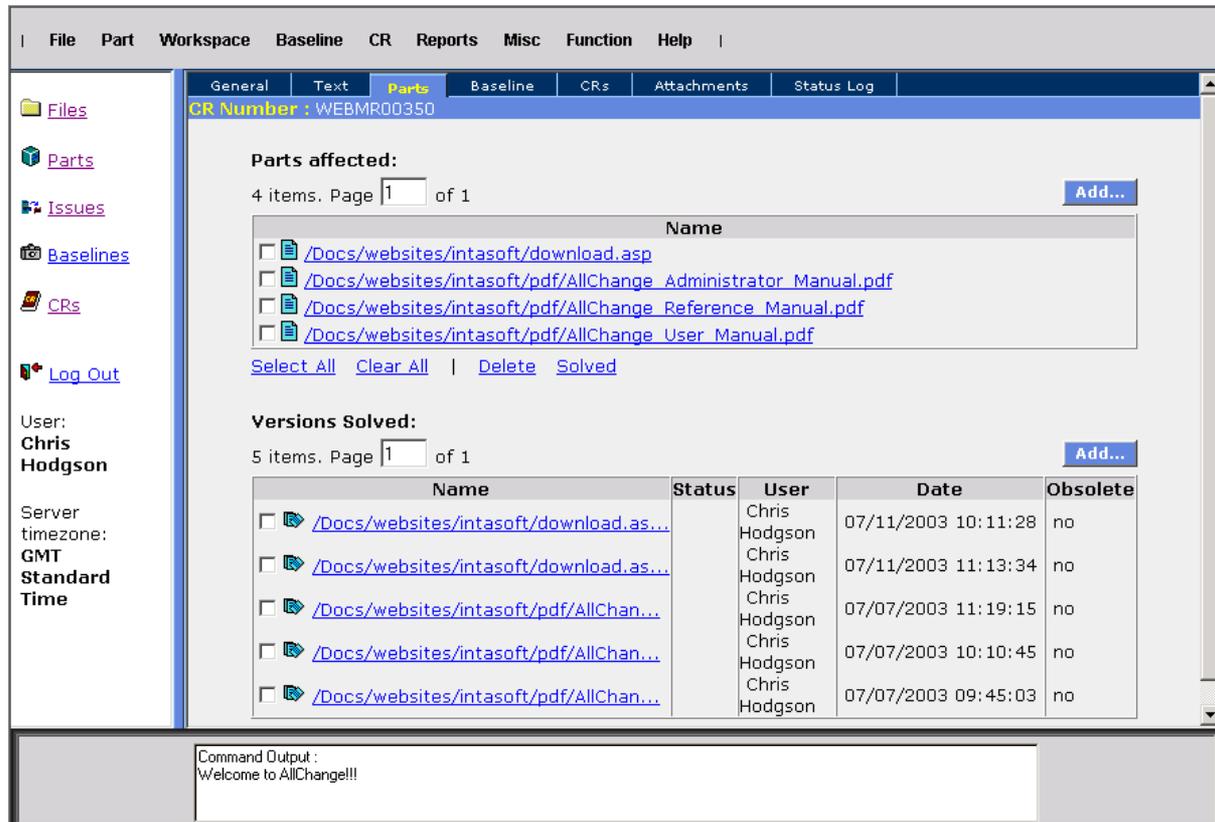
To view an attachment you need to first download the file to your PC - select the file that you would like to download from the list of attachments and select **Download**. This will present a dialog allowing you to **Open** or **Save** the file, opening will allow you to view the file contents and saving will allow you to save the file to your PC or network and then view the file using the appropriate application.

To detach a file from the files affected list, select the relevant files and click on the **Detach** button.

When adding a new CR or Baseline, then file attachments must be specified last, as the filename cannot be saved between tabs.

Items Affected

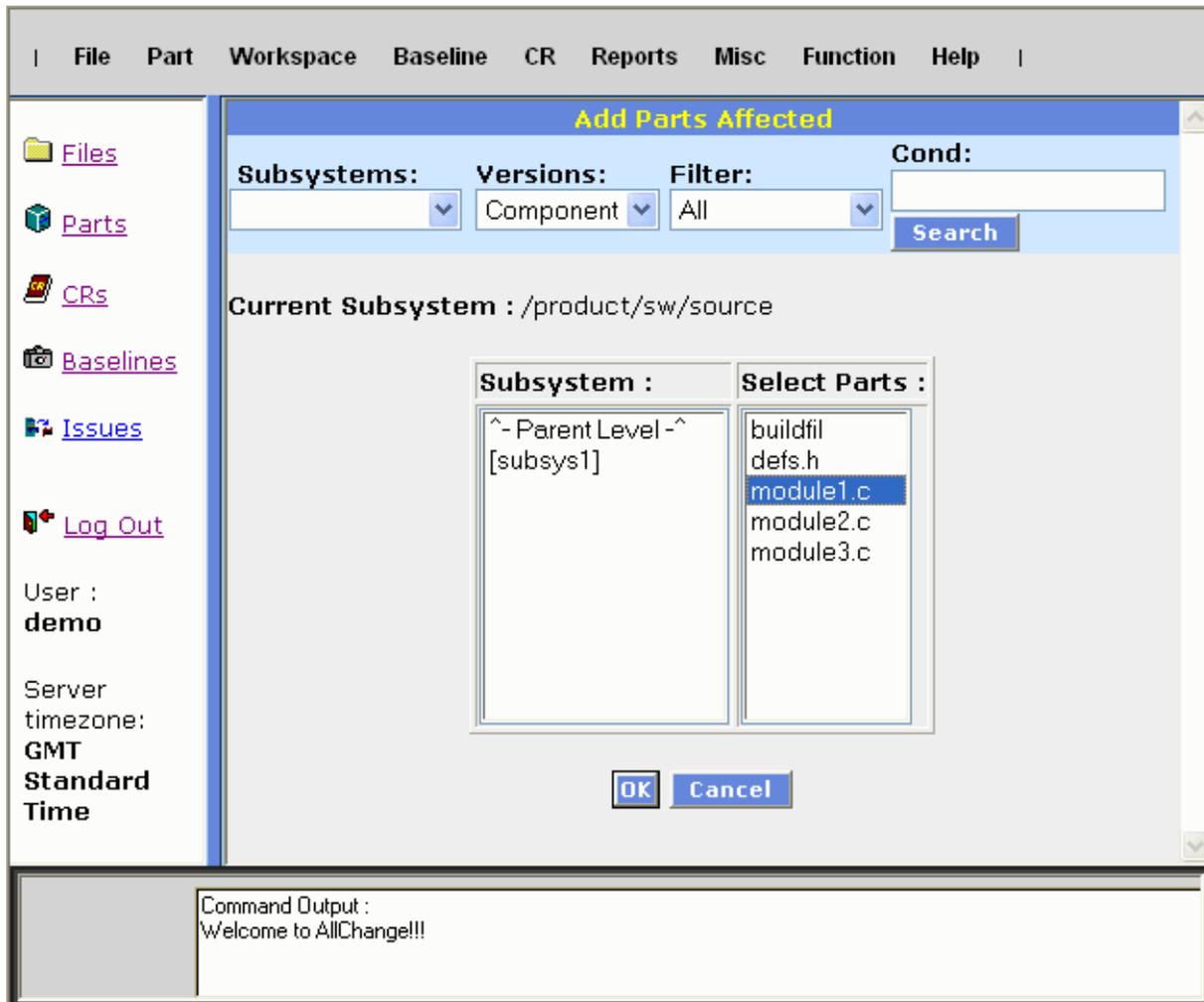
Items Affected (Parts affected, Baselines Affected, CRs Affected etc) are displayed in buffered tables. The size of the buffer is set to 5 as default; this can be adjusted by setting the **Items Affected Buffer Size** in the [Options](#). You can move forwards and backwards through the buffers by clicking on **Next** or **Previous**. You can also go directly to a specific page by entering the page number in the edit box.



CR Parts Affected/Versions Solved

Selecting the **Add** button for either Parts Affected or Versions Solved on the Parts tab of the CR viewer allows parts to be selected for addition.

The selection windows shows 2 lists, the **Subsystem** list and the **Select Parts** list. The subsystem list is used for navigation around the tree and the Select Parts list is used to select specific components or versions which are to be added. Select ^-Parent Level-^ to navigate back up the tree.



Reports

Reports are available in HTML format for each AllChange database. By default only reports that generate HTML are enabled on the report dialogs. To list the plain text reports, click on the **Include Text Reports** option.

The screenshot shows a dialog box titled "Report CR". The dialog has a light blue header and a white body. The fields are as follows:

- CRs:** A text input field containing "ACCR00001" and a blue button with three dots.
- Condition:** An empty text input field.
- Format:** A dropdown menu with "hcrful: Full CR Details (HTML)" selected.
- Include text reports:** An unchecked checkbox.
- Index:** A dropdown menu with "(Default)" selected.
- Value:** A dropdown menu with "(Any)" selected.

At the bottom of the dialog, there are two buttons: "OK" and "Clear".

Web Workspaces

Workspaces can be designated as *web workspaces*. These workspaces, and only these workspaces, are available to the web browser interface. Web workspaces may only be defined using ACCONFIG or the browser interface. They can have the **Allow in ACE** attribute set so that they are also available to the Windows **AllChange** application.

Web workspaces defined with the **Allow Transfer** attribute set have upload and download capabilities. Files can be transferred through the web browser, from the client's machine to the web workspace directory on the server machine (and vice versa).

Workspace directories for web workspaces must be defined as a directory that the server can read/write to. The web server user and the AllChange user will both need read/write permissions for this directory. When uploading a file, the file is simply posted to the web server which writes the file into the appropriate directory.

The transfer workspace should be seen as a temporary directory where files are stored between transferring to/from the client, similar to an FTP workspace.

When checking out to a transfer workspace, the files are written to the web workspace (a server directory). The user then has the option to save the files just checked out to the client machine. Similarly, when checking in, the user has the option to upload the files from the client machine to the server workspace directory before checking in. This provides a mechanism to check in / out via a client.

Files that are checked out read only or for edit can be downloaded. Files that are checked out for edit can also be uploaded. Files not checked out at all can be uploaded to create a new version. New files can be uploaded to the transfer workspace and then checked in to create new parts.

Parts can be downloaded when checking out. After check-out to the workspace, the user will be prompted to download the checked out files to their client machine. The user has the option to download, or continue without downloading. If the user chooses to download, the user will be prompted to pick a directory to save the files to. Any sub-directories will be created as necessary. Files are downloaded relative to the current working part. Files checked out read-only will be downloaded as read-only. Files that are checked out read-only, when checked out for edit, will prompt for transfer as well.

A part, providing it is checked out to a transfer workspace, can be downloaded independently of check out, by selecting **Download** from the Part menu. Files can also be selected in the File browser for download.

Parts can be uploaded when checking in. After selecting parts (or files) to check in that are checked out for edit, if discard is not selected, the user is prompted to upload the corresponding files before check in occurs. The user now has three choices. The first is to upload the corresponding files from the client. In this case the user is prompted for the client directory where the files are stored. This is relative to the current working part, and any sub-directories will be recursed if necessary. The second option is to Cancel the Check In all together, and the third is to Continue the Check In, but skip the upload. Files checked out for edit can also be uploaded in this way from the file browser.

An upload or download can be cancelled at any time (during upload or download) by pressing the Cancel button. This will abort the upload or download.

New files can be uploaded to the transfer workspace via the file browser. Click on the **Upload New Files** link to upload selected files to the directory in the File browser. Files can then be checked in to **AllChange** from the file browser.

The **Add Part** dialog also has the facility to upload a corresponding file from the client to use as the initial version of the part. Click on Browse next to 'Transfer file from client' to select a file. The part name will automatically be filled in from the name of the file. This must be done as the last thing to be filled in on the **Add Part** dialog.

Files checked out to a web workspace can of course be edited in the normal way if the user has direct access to the workspace directory. FTP workspaces can also be used via the web interface.

Users cannot browse above the root directory of a workspace in the file browser.

Options

Some but not all of the ace options are supported by the browser interface. These are available from **Misc | Options**.

In addition there are some browser specific options:

Items Affected Buffer Size: Specifies the size of the buffer on the items affected tabs of the CR Viewer. This determines the number of items affected that will be displayed in one page/buffer.

UK Date Format: Select to show dates in UK format of DD/MM/YYYY

US Date Format: Select to show dates in US format of MM/DD/YYYY

Options

Max Browser Items: 1000

CR Buffer Size: 50

Items Affected Buffer Size: 5

Initial Role: (Auto)

Initial Workspace: (None)

Show Issue icon in part browser

Show state of baseline details

UK Date Format US Date Format

OK Cancel Help

Cancel Current Operation

The Misc menu has a Cancel Current Operation item that can be used to stop the web interface from updating the current page with data coming from **AllChange**. This might be useful if **AllChange** is taking a particularly long time.

Use this option with caution; it will allow you to request another page from **AllChange**, but there is no guarantee as to what state **AllChange** might be in.

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 out read 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. CItemType)

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 occurrences 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-cycle 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

absolute, use	78
ACC	18, 20
ACCEL	28, 184, 290
access control	277
Access, integration with	106
ACCONFIG	18, 21, 300
acdbgwin	22
ACE	18
customising	282
options	289
AceSettings	292
AceToolbarSettings	292
ACheader	137
ACident	138
AClog	138
ACpart	138
ACPROJECT	25
acreport	255-256, 268
ACSERVER	22
action line modifiers	162
ACversion	137
adbgwin	22
add, baseline	231
CR	200
part	61, 68
workspace	87
Admin	26
administering AllChange	21
Against	116
AllChange	14
facilities provided	4
overview	2
project directory	166, 173
user interface	16

AllChange Command	25
Allow CR browser header sort	190, 282, 289
Allow in ACE	300
Allow Transfer	295
alter, baseline	236-238
CR	205
part	64
Always use this project	289
Apply Condition	45
Approval	11, 247
Araxis	97, 115, 291
Arbitrary Field	282
archiving, baselines	245
CRs	214
parts	73
Artisan	106, 108
Ask if more than one match	83
Assign CR	210
Attach	297
attach workspace	86
Attachment	297
Attachments	195, 198, 219, 227, 229, 290
Authentication	294
Author	137
Auto-progress	249
Autobuild	165, 176
Autoupdate	118, 136, 289

B

backupac	22
baseline	
instance	217
Baseline	218, 220, 290
Header tab	221
Status Log Tab	228
baseline items affected database	230
Baseline Relationships	226, 229
Baseline text	223, 66, 223, 206, 223

baseline view	
votes tab	228
baseline viewer	290
baselines	216
adding	231
altering	236-238
archiving	245
browse	218
checking out	245
copying	234
creating	231
design	217
difference between baselines	244
header	221
importing	246
introduction	12
populating	239
release	217
reporting on	270
status	241
status log	228
updating	239
view	220
baselines affected	198
Baselines Affected	229
baselines solved	198
Bill Of Materials	225-226
BOM	225-226
branches	76, 82
browse	
check-outs	113
files	125
part relationships	57
Browse Roles	25
browse, baselines	218
CRs	188
files	70
monitors	253

parts	40
Browser	26, 31, 282, 295
columns	282
customising	282
sorting	283
browser, aborting update	31
baseline	218
CR	188
file	70
options	289
part	40
Buffer	189
buffer size	290
Build	128, 154
build template	166
build template file	165
build thread	129, 132
build tool	174
build, autobuild	165
builtins	173
command line options	175
environment	173
internal macros	159
introduction	12
macros	158-159
options	130
setting up a buildfile	165
build.ini	173
build.tpl	166
buildfile	156, 165, 174
directives	164
builtin rules and macros	173

C

Cancel Current Operation	302
cancelling reports	261, 266
Cast Vote	71, 208, 243, 247
Category	274

change management	8
Changed State	126
Check-out	89, 113
check-out browser	113
check-out state	113, 126, 290
check-outs	
browse	113
part	56
reporting on	270
unlogged	120
Check Check-outs Changes	115
Check In	68, 83, 98, 105, 145, 300
from DOORS	110, 148
from Eclipse	107
from Edit	100
from Explorer	111
from MCSCCI	106
from Microsoft development environments	106
from Read Only	100
from Rhapsody	146
from Word	108
New Parts	102
Not Checked out	101
Check In all Edits In Workspace	104
Check In Check-outs	104
Check In Criteria	104
Check Out	81, 89, 105, 120, 145, 301
baseline	245
for edit	92
from DOORS	110, 148
from Eclipse	107
from Explorer	111
from MCSCCI	106
from Microsoft development environments	106
from Rhapsody	146
from Word	108
parts checked out	55
read-only	91

Check Out Criteria	91
check spelling	286, 291
checked out	113
Checked out For Edit	119
Checked out Read Only	119
CheckIn	71
CheckOut	71
Clear AllChange Settings	25, 292
Clear Download Area	295
Clear output window	30, 287
Clear Settings	27, 285
Closed	35
Colours in the Cycle View	37
Column Report Wizard	255, 274
command line options, autobuild	176
build	174
prep	177
Compare	114, 137, 244
Compare Versions	114
Comparing Subsystems	115
components	39, 48
creating	61-62
concurrent	22
Condition	32
condition editor	184
Configuration Options	80
confirm on update	289
Containing	117
Copy	27
Copy Baseline	234
Copy CR	202
CR	80
Attachments	195, 227
CR browser	190, 290
CR Buffer Size	289
CR fields	190
arbitrary	214
CR Number	191

cr only	24
CR Relationships	194, 198
CR text	223, 66, 223, 206, 223
CR view	
votes tab	197
CR view, Baselines Tab	
General Info Tab	191
Status Log Tab	196
Text Tab	223
CR Viewer	297
create, baseline	231
CR	200
part	61, 68
CRs	188
CRs Affecting	198
CRs For Baseline	239
CRs Solved	198, 288
CRs, altering	205
archiving	214
assigning	205, 210
browse	188
copying	202
creating	200
creating by email	202
creating from DOORS	212
creating from web browser	203
creating from word document	203
importing	215
life-cycle	207
locked	81, 210
related to other CRs	210
reporting on	270
requirements	203
status	207
status log	196
updating	205
view	190
crxlactv	272

crxIstat	272
crxItime	272
crxITimeForClearance	272
crxItrnd	272
current directory	160, 166, 173
current working part	75
Customising the User Interface	188
Cycle view	89
Cycle View	35, 37, 99, 105
50% 100% 200%	36
colours	37
print	36
visibility of statuses	37
zoom	36

D

Debug	290
decision override	248
Decision Type	247
Default	89
default project	289
default editor	291
default version	77, 89, 235, 280
default viewer	291
design baseline	217, 233
Design baseline	232
detach	297
detail state	289
Details	287
Differencing	114, 137, 144-145, 244, 291
files	114
versions	114
Differencing Application	97, 289
Diffs	114
directives, prep	178
Directory	115
Disable splash screen on startup	289
dockable	26

document	108, 139, 145
document stamping	139, 145
DOORS, check in	110, 147
check out	111, 148
creating CRs	212
interface	147
DOORSCRCClass	212
download	296
Dreamweaver	109

E

Eclipse	107, 148
Edit	126
Edit Hyperlink	33
editor, default	289
email	202
Email submission of vote	250
Enable Word document stamping	139
End CR	188
environment	24, 282
environment variables	22, 173
environment, customising	292
error dialog, customising	289
error messages	290
Escape key	31
eval	29, 184, 255
Eval Command	25
Excel	108, 139, 145
Excel, AllChange reports	256, 270, 274
Existence	126
Explorer	137
Explorer interface	144
Autobuild	145
Build	145
check in	145
check out	145
differences	145
keywords	145

promote	145
report	145
update workspace	145
versions	145
Explorer, check in	111
check out	111
Export	36
Export to Image	36
expressions, prep	180
External Tools options	291

F

fetch file	20
fetch files	78
File	125
browse	70
check in	68, 98
check out	89
checking in and out	81
comparing	145
differencing	114, 137
editing	126
get version unlogged	120
import new	98
importing	68
support for access on remote machine	122
viewing	126
file browser	70, 125
file stamping	137
file type	125
Filter	44
Filter By	32, 189, 219, 251, 283
filter, parts	44
Find	42, 117, 288
find/replace	50, 194, 224
Firefox	293
Fold baseline versions in folding view	287, 289
Folder List	26, 287

folding	30, 287
Folding View	40, 113, 188, 218, 288
Format	268
FTP	84, 97, 105, 115, 122-123, 127, 134
full install	24
full path	75

G

general info, CR	191
Get Baseline to Directory	120
Get Version to Directory	120, 295

H

header, baseline	221
help	37
hidden files	105, 112
hierarchical CRs	210
hierarchy	39
home directory	25, 273, 276, 291
HP Testdirector	109
HTML	123
hyperlink	30, 33-34, 50, 193, 223

I

I-Logix	106
Ident	138
ignorevote	248
impact analysis	39
import, CRs	203
requirements	203
Import, CRs from Email	202
Import, Votes from email	250
ImportFromEmail	202-203, 250
importing	
new versions	121
Importing Parts from CSV File	61
importing, baselines	245
CRs	214

directories	68
files	68
new files	102
new versions	99
parts	73
incremental release	245
index	31
inference rules	162
initial role	289
Initial Workspace	289
Insert Into Text	50, 68, 193, 205, 207, 224
install	24
full	24
workstation	24
Instance	39, 41
Instance Baseline	232, 234
Instances	43, 53, 59, 217, 225, 231, 270
Integrations, HP Testdirector	109
interfaces	
Explorer	144
interfaces, DOORS	147
Rhapsody	146
Word	145
internal macros	159
Internet Explorer	293
introduction to AllChange	2
Item Affected	57, 198
Item Affecting	59, 200, 231
Items Affected Buffer Size	295
ITIL	3, 195, 210

J

Joint Editing	80, 92
---------------	--------

K

Keep checked out	99
keyword	137

L

large icons	30, 287
latest version, updating to	118
Licensing	22
life-cycles, baselines	241
CRs	207
introduction	10
parts	70
using to check in and out	105
Life Cycle	35
Life Cycles	
changing status	35
Line	138
List	26, 287
lock, CR	210
parts	217, 233
Locker	138
locking baseline	233
Locking Models	81
Log Off	295
logon	293
Look In	117

M

macro	158-159
Macromedia	109
macros, prep	179
Main Menu	26
Managing Files on a Remote Machine	122
maximum browser items	289
MCSCCI	106, 110, 146
Menu	282
Merge	82, 95
Merging Subsystems	97
meta-baselines	226, 270
creating	231
Microsoft Common Source Code Control Interface	106

minimised	23
miscellaneous options	290
modification time	154
monitors	253
browse	254
introduction	11
reporting on	272
Move Check-outs to Branch	95
Mozilla Firefox	293
MS Excel	108, 145
MS Project	213
Must be affected by part	80
Must be assigned	80

N

Never	45
New CR	201
New Instance	59
New Search	117
new version	81, 296
new, baseline	231
CR	200
part	61, 68
next	29, 190
No Change	235
NOACTION	27
NOCMD	27
NOCOND	27
norestorestate	23

O

object, building	128, 154
obsolete	91, 99, 121, 201, 236, 238, 287
baselines	223
CRs	191
parts	49, 52, 239
Only list versions in results	117
Open CRs Only	188, 211

Optimistic	92, 113
optimistic locking	52, 78, 81, 105, 119
Options	29, 225, 282, 289, 296
options, ACE	289
build	130
debug	290
OS Command	25
Other Settings	117
output window	26, 30, 255, 287
customising	289

P

parallel development	76, 81, 92, 95
Parent Level	298
part browser	40, 290, 296
check-out state	290
filters	44
Part Column State	125
Part Comparison Settings	117
part fields	45
Part Item Affected Viewer	58
part paths	42
advanced	75
Part Relationships	54, 57
Part State	126
Part text	223, 66, 223, 206, 223
part view	
Check-outs Tab	55
votes tab	56
part view, Baselines Tab	
Status Log Tab	55
Version Tab	51
part, part view	47
viewer	46
parts	39
check out	89
difference between versions	114
parts affected	57, 198, 298

parts, adding	61, 68
altering	64
arbitrary information	49, 53
archiving	73
browse	40
check in	98
checking in and out	81
class	48
creating	61
flags	49, 52
hierarchy	39
import new	98
importing	74
in a baseline	75
life-cycles	70
locked	53
locking	222
part fields	47
reporting on	269
status	53, 70
status log	55
status, changing	71
type	48
usage relation	78
version fields	51
wildcards	75
Paste	27
Pattern	104, 126
Pattern matching	32
Pessimistic	92, 113
pessimistic locking	52, 76, 81, 92, 105, 119
pool	125
pools	122, 127
associating with workspaces	280
updating	136
using for building	133
populate baselines	239
Powerbuilder	106

preferences	29
Prep	176
prep, command line options	177
directives	178
expressions	180
macros	179
preprocessor tool	176
preview	33
Preview Pane	30, 33, 283, 287
previous	29, 189
Print	30, 33, 255
product design	5
product development	6
programs	22
running	22
Project	18, 23, 25, 166, 173, 290
promote	128, 134
Promote Report	273, 276
prompt send errors to	289
protected text	223, 205, 223
pseudo workspace	86
Putaway store file time	51

Q

Quick View	127
------------	-----

R

Read	29
Read Only	119
readonly	23
Real Time Studio	106, 108
Recursive	117, 270
refresh windows	28
Registered	89
Registered Version	235
Registration	280
registrations	279
introduction	11

registry	292
Relationships	
baseline	226
CR	194
part	54
relative paths	75
relative, use	78
Release	13, 136, 217, 241
release baseline	234
Release baseline	232
remote files	122
managing	122
pools	134
selection	105
workspaces	83
Rename Branch	95
Reordering browser columns	283
Report	26, 255
Report Wizards	255
reporting	255
introduction	13
running reports from Word	265
reports, cancelling	261, 266
new	273
requirements, importing	203
Revision	138
Rhapsody	106, 110, 146
check out	146
interface	146
role	27, 277, 290
roles	277
initial	290
setting	281
S	
Save Shortcuts	285
Save Windows On Exit	25
scan	117

SCCI	106, 110, 146
Scroll current status into view	37
Search	50, 194, 224, 296
search path	161
autobuild	169
section	66, 206
select all in list	25, 28
send errors to	289
Serial Votes	247, 250, 252
Set Role	25, 277
Set to Defaults	35
setup requirements	24
Shortcut bar	26, 287
Shortcuts	282
Show attachments flag in icons	290
Show issue icon in part browser	289
Show Issue icon in part lists	220, 225, 289
Show Last	219
Show Latest Solved Only	195
Show Obsolete	121, 236, 238
show obsolete items	30, 91, 99, 287
Show only orphans at top level in folding view	287, 289
show output window	289
Show state of baseline details	220, 225, 289
small icons	30, 287
SolidWorks	112, 149
SolidWorks Show inTaskpane	150
SolidWorks Symchronise	150
SolidWorksDoc	151
SolidWorksDrawing	151
sorting browsers	282
sorting CRs	188
Specify Criteria	98
spell checking	286, 291
splash screen	290
Stamping	137
start cr	289
StartCR	188

startup options	290
Status	89, 99, 106
Status bar	26
status log, baseline	228
CR	196
part	55
status, baseline	241
CR	207
part	53, 70-71
statusbar	27
Statuses	
changing status	35
Statuses in the Cycle View	35, 37
stop-criteria	248
Substitution	137
Subsystem Comparison Settings	114
subsystems	39, 48
creating	62
Subversion	61
system directory	166, 173

T

targets, special	158
template	63, 166
TestDirector	109
text	223, 66, 223, 205, 223, 236
text file, stamping	137
text, Baseline	223
text, CR	223
text, Part	223
Today	138
toolbar	26-27
configuring	27
Top	89
Top Version	235
TopPart	81, 93
Try current workspace	83

U

Undo	27
unlogged check-outs	120
Update	29, 34, 118, 136, 145, 169, 235-237, 296
CR	205
part	64
Update Options	119
Update Top Window Only	289
Update Workspace	80, 118
upload	296
User	9, 277
uses	48, 78

V

valid for change	81
Value	32
variant	95
VCfile	138
VCpath	138
Version Control Command	25
Version text	223, 66, 223, 206, 223
version, part view	51
Versions	43, 76
altering	52, 64
comparing	145
default	77, 280
differencing	114
importing existing	121
status	53
Versions Solved	198, 298
view	
part	45
View	26, 31, 33, 40, 126, 287
View in new window	33
view, baseline	220
CR	190
issue	118

viewers	33
Viewing Files	126
Views	33, 282
VisDiffs	114, 137
visibility	36
VisMerge	96
Visual Basic, integration with	106
Visual C++, integration with	106
vote_manager	248
Votes	272
Voting	35, 56, 70-71, 197, 208, 228, 241, 243, 247
voting, by email	250

W

Web	203
Web Browser	293
Web development, support for	123
web workspaces	87, 296
wildcards	75
parts	75
Windows Authentication	294
Word	137, 139
Word file, stamping	139
Word interface	145
check in	145
check out	145
compare versions	145
exit AllChange	145
insert AllChangefield	145
list versions	146
version information	146
Word, AllChange reports	256, 265, 274
check in	108
check out	108
integration	139, 265
requirements facilities	203
workbook	108, 139, 145
Workfile	126, 137

workfiles	84
Workspace Registration	280
Workspace/Pool	126
Workspaces	25, 83, 87, 122, 125, 289, 296
attaching to	280
autoupdate	118
creating	87
customising	279
deleting	87
initial	290
modifying	87
moving parts between	118
registrations	279
updating	118
workstation install	24

Z

Zoom	36
Zoom to Fit	35